Improving Novelty Search Sample Efficiency with Models or Archive Bootstrapping

Elias Hanna

Ph.D Thesis
Improving Novelty Search Sample Efficiency with Models or Archive Bootstrapping

Research laboratory:
Institut des Systèmes Intelligents et de Robotique, Sorbonne Université
Research Unit:
Architectures and Models for Adaptation and Cognition

Author:
Elias Hanna
Supervisors:
Stéphane Doncieux (Director)
Miranda Coninx

Reporters:
Alain Dutech
Cyril Fonlupt
Examiners:
Pascal Morin (President)
Sao Mai Nguyen

May 3, 2024

Abstract

In the context of policy search for robotic systems, being sample-efficient is a most. Evolutionary Algorithms have been used in the past ten years to achieve significant results in the robotics domain as their Darwinist approach to optimization allows them to bypass problems often encountered by gradient-based optimization methods like Reinforcement Learning. Nevertheless, such methods remain sample greedy and almost impossible to transfer directly onto a real robotic system. This Ph.D thesis takes interest in solving that sample-efficiency problem, especially for the Novelty Search algorithm, a novelty-driven Evolutionary Algorithm. Incorporating learned models in the optimization process has been a solution towards sample-efficiency for many years, but few works address this within the Novelty Search framework. Three research axis within that framework are explored in this manuscript. Firstly, the impact of pre-training the learned model with data gathered using random processes of varying time-correlation is evaluated. It is shown that the impact is negligible on a state-of-the-art model-based Evolutionary Algorithm, but that it is significant on a model-based Reinforcement Learning algorithm with returns with up to ten-fold differences between the best and worst random process used. Secondly, a preliminary study is made on a new approach aiming at biasing the initial population of the Novelty Search algorithm towards a more behavioraly diverse one using random dynamics models ensembles. It is shown that this approach successfully reduces the number of evaluations required by Novelty Search to cover the environment of a two-wheeled robot. It is also shown that this approach fails on a more complex locomotion environment of an hexapod robot, and the lack of diversity captured by the random models ensembles used is determined as the cause. Finally, a new model-based Evolutionary Algorithm, dubbed Model-Based Novelty Search, is proposed, with the aim of preserving the strong exploration capabilities of Novelty Search while reducing the numbers of evaluations needed to reach the same coverage of the Behavioral Space. Results on three robotic tasks show a reduction in sample usage compared to Novelty Search of 30% up to 75% depending on the considered task, a significant advance towards a more sample-efficient Novelty Search algorithm.

Résumé

La robotique est un domaine vaste et le degré de complexité du contrôle d'un robot peut varier grandement selon le système à contrôler, la tâche à accomplir et l'environnement dans lequel cette tâche doit être accomplie. Pour faire face à cette complexité, des méthodes permettant au roboticien d'abstraire de nombreuses parties de la tâches ont émergées, notamment les méthodes d'apprentissage. Ainsi, le problème du contrôle du robot passe d'une tâche d'ingénierie à un problème de design d'un algorithme d'apprentissage, ayant pour objectif l'apprentissage d'une politique. Dans le cadre de l'apprentissage par renforcement [151], l'environnement étant défini comme un processus de décision Markovien, une politique est une stratégie de décision qui pour un état du système renvoie l'action que l'agent (le robot) doit effectuer, généralement avec un objectif précis en tête qu'il soit de résoudre la tâche ou d'explorer l'environnement du robot. Nous nous intéressons dans cette thèse uniquement à des stratégies déterministes, mais une distribution de probabilité sur l'action à effectuer peut également être retournée par la politique pour un état donné.

De nombreuses méthodes de recherche de politique existent mais, dans le cadre de cette thèse, deux nous intéressent particulièrement. Premièrement, l'apprentissage par renforcement consiste en interactions successives avec l'environnement suivant une politique, chaque interaction donnant un signal de récompense à l'agent. La récompense peut-être positive ou négative, et l'agent se sert de ce signal de récompense pour améliorer sa politique, de telle sorte qu'à terme l'agent prenne l'action optimale de telle sorte à maximiser la récompense reçue le long d'un épisode, un épisode étant une suite d'interactions avec l'environnement. L'apprentissage par renforcement s'est fortement développé ces dernières années à travers de nombreux succès en robotique [81] et dans d'autres domaines [173, 118] . Néanmoins, l'apprentissage par renforcement se base sur le gradient du signal de récompense, qui en robotique est souvent trompeur ou épars [12] et peut mener à des apprentissages de politiques sub-optimales.

Les algorithmes évolutionaires [110] sont une autre famille de méthode d'optimisation pouvant être utilisés pour trouver une ou plusieurs politiques permettant de répondre à un problème donné. A l'inverse de l'apprentissage par renforcement, ces méthodes n'utilisent pour se guider que de la performance d'une politique sur un épisode entier, et sont donc moins sensible au problème de récompense rare. En particulier, les algorithmes de diversité [91, 133] sont un sous-ensemble des algorithmes évolutionaires dont l'objectif est de générer un ensemble de politiques diverse les unes par

rapport aux autres dans un espace dit comportemental, défini par l'utilisateur. Le premier algorithme proposant ce schéma de fonctionnement est Novelty Search [91]. Novelty Search a des propriétés couvrantes de l'espace comportemental [43], ce qui permet en alignant la tâche et l'espace comportemental à cet algorithme de résoudre et d'apporter un ensemble de solutions diverses à des tâches de robotique complexes.

Néanmoins, les méthodes d'apprentissage présentées ci-dessus reposent souvent sur des volumes de données importants pour permettre au robot de trouver des comportements résolvant la tâche qui lui est confiée [96]. Hors, dans le monde réel, l'utilisation d'un robot est coûteuse: temps de développement de contrôleurs stables, temps d'exécution des mouvements, risques de sécurité et usure de la machine sont tous des facteurs à prendre en compte lorsque l'on décide de travailler sur un robot dans le monde réel. Ainsi, dans le monde de l'apprentissage en robotique, l'utilisation de simulateurs s'est d'autant plus démocratisée [30, 157, 28]. Un comportement est appris sur le système simulé, puis transferré sur le système réel, pour un coût d'évaluation sur le système réel grandement réduit, l'apprentissage ayant déjà été effectué en simulation. Cependant, l'utilisation de simulateurs ne se fait pas sans anicroches et amène également d'autres problèmes, le plus notoire étant le *Reality Gap* [66]. Le *Reality Gap* est l'existence d'une divergence dans la dynamique du système dans le monde réel et celle observée dans sa version simulée. En effet, la simulation est une approximation de la réalité et ne prend pas en compte tout les facteurs pouvant influencer le comportement du robot dans la réalité: calibration, bruit dans les capteurs, frottements...

Pour surmonter ce problème, de nombreuses méthodes ont été proposées. Les plus notoires sont l'Identification de Système [30, 157, 28, 1, 83], l'Adaptation de Domaine [69, 48, 101, 21, 76, 16, 67, 159], la Transférabilité [84] et la Randomisation de Domaine [156, 129, 138, 109]. En dépit de ces propositions de solution au problème du *Reality Gap*, d'autre façons de le contourner existent. L'une d'entre elles est au coeur de cette thèse, les approches basées sur un modèle appris de l'environnement [131]. Les approches basées modèle sont centrées non pas sur un simulateur, défini à l'avance et dont les paramètres sont souvent fixés, mais sur un modèle de la dynamique de l'environnement appris au fur et à mesure des interactions avec ce dernier. Cette approche permet généralement, dans le support d'entraînement du modèle d'avoir une représentation fiable de la réalité car directement appris à partir de données du monde réel. Ces méthodes concourrent souvent les unes entre les autres sur un facteur primordial d'un algorithme de recherche de politique basée modèle: la *sample-efficency* ou le nombre d'évaluations sur le système réel nécessaire pour atteindre un comportement résolvant la tâche donnée au robot.

Ainsi, dans cette thèse, nous abordons le sujet de la réduction du nombre d'évaluations nécessaire pour l'algorithme Novelty Search pour couvrir l'espace comportemental défini par l'utilisateur, tout particulièrement grâce à l'utilisation de modèles appris. Nous chercherons donc à répondre à la problématique suivante:

Comment réduire le nombre d'évaluations requise par Novelty Search pour couvrir l'espace comportemental ?

Pour y répondre, nous procéderons en trois volets, chacun attaquant différentes phases de l'apprentissage de Novelty Search ou des méthodes basées modèles. Premièrement, nous étudions l'impact des processus aléatoires de récolte de données initiale pour le pré-entraînement d'un modèle de la dynamique de la tâche ainsi que la manière dont ils se comparent les uns aux autres, en montrant qu'en fonction de l'environnement la différence de performance initiale de l'algorithme peut être jusqu'à 10 fois supérieure avec une méthode de récolte de données initiale appropriée. Deuxièmement, nous proposons une approche préliminaire basée sur l'utilisation de modèles de la dynamique aléatoire permettant de générer une population initiale pour l'algorithme Novelty Search plus diverse que d'utiliser une population de politique paramétrées aléatoirement. Finalement, nous proposons une nouvelle méthode basée modèle autour de l'algorithme Novelty Search, nommée Model-Based Novelty Search, qui divise jusqu'à 5 fois le nombre d'évaluations sur le système réel nécessaires pour atteindre une couverture similaire à celle de Novelty Search classique.

Dans le premier chapitre, nous abordons donc la question du processus aléatoire utilisé pour générer des données issues de l'interaction entre le robot et son environnement, données par la suite utilisées pour pré-entraîner un modèle utilisé avec une méthode de recherche de politique basée modèle. Ce processus aléatoire contrôle le comportement du robot dans son environnement. Nous considérons cinq méthodes de récolte initiale de données, deux venant de l'état de l'art et trois étant des marches aléatoires dans l'espace des actions que peut prendre l'agent. Les deux méthodes issues de l'état de l'art sont les actions aléatoires et les politiques paramétrées aléatoirement. Les trois autres sont des paramétrisations différentes de marche aléatoires générées à l'aide de bruit coloré [130], chaque paramétrisation modifiant le degré d'auto-corrélation des trajectoires générées dans l'espace des actions. Chacune de ces méthodes de bruit coloré est dénotée $CNRW_\beta$, β étant le facteur permettant de régler le degré de corrélation temporelle dans les sequences d'actions générées avec un β plus grand, la séquence étant plus fortement corrélé.

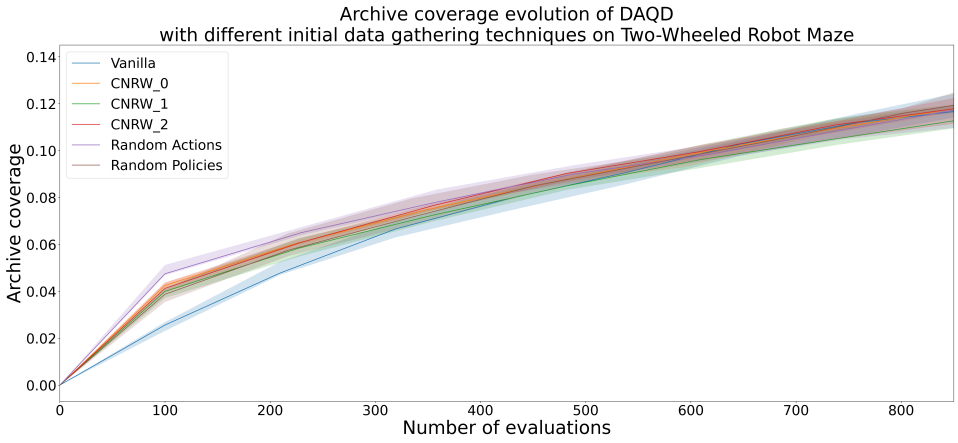
Plus en détail dans le chapitre 3, nous étudions le comportement de chacune de

ces méthodes d'initialisation sur 5 environnements robotique différents que nous caractérisons à l'aide d'une métrique de consistence des actions dans l'espace d'état du système. Cette métrique permet de montrer que les environnements avec une consistance plus grande auront leur dynamique mieux explorée par des méthodes de récolte initiale de données peu corrélés dans le temps, notamment les actions aléatoires. La réciproque est également observée pour les environnements avec une consistance faible, les modèles de la dynamique ayant une erreur de prédiction plus faible lorsque les données ont été récoltées avec un processus aléatoire dont les séquences d'actions sont plus fortement corrélées.

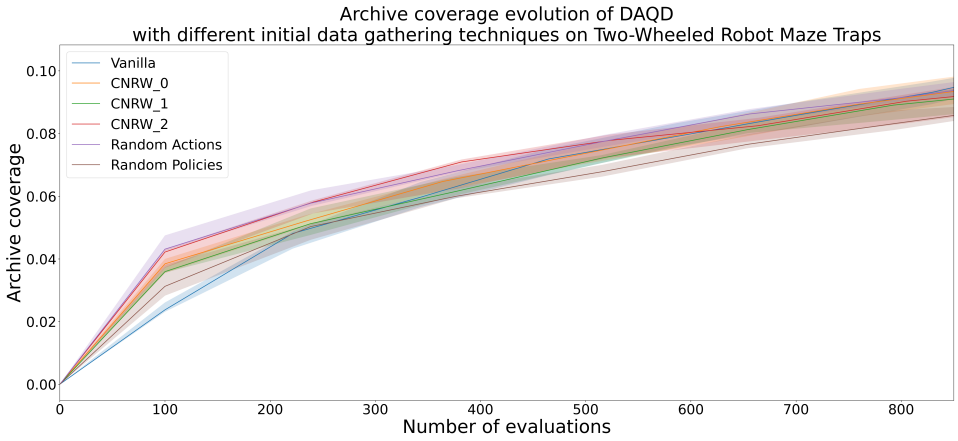
Finalement, nous analysons l'impact de ces méthodes sur deux algorithmes état de l'art: DAQD [98] et PETS [26], le premier étant un algorithme Quality-Diversity basé modèle, et le second étant un algorithme d'apprentissage par renforcement basé modèle. Les figures 1a, 1b et 1c montrent l'évolution de la couverture de DAQD en fonction du nombres d'évaluations avec les différentes méthodes d'initialisation étudiées. L'impact observé n'est pas significativement différent entre les différentes méthodes considérées, et est même rapidement effacé avec le nombre d'évaluations grandissant. DAQD semble ne pas être très sensible à la qualité initiale du modèle, et l'effet de l'initialisation est rapidement supprimé, le nombre d'évaluations supplémentaires étant effectué étant rapidement très supérieur au budget d'initialisation de 10 épisodes.

Les figures 2a, 2b et 2c montrent l'impact des méthodes d'initialisation sur l'algorithme PETS. Cet algorithme effectuant une nouvelle optimization après chaque action prise, l'effet de l'initialisation de modèle devrait être plus prononcé. Effectivement, nous observons sur la figure 2a, que les actions aléatoires donnent une récompense médiane finale jusqu'à plus de dix fois supérieur à celle d'une marche aléatoire à base de bruit coloré. Cet environnement étant le plus consistant, cela correspond bien à nos attentes en termes de correspondance entre la corrélation temporelle des séquences d'actions générées, la qualité du modèle et l'impact sur la méthode de recherche de politique qui s'ensuit. De même, sur la figure 2c, nous observons que la marche aléatoire avec un bruit coloré le plus corrélé dans le temps donne un retour médian en terme de récompense quasiment dix fois supérieur à celui des actions aléatoires. Cet environnement étant le moins consistent, cela confirme à nouveaux les observations faite précedemment.

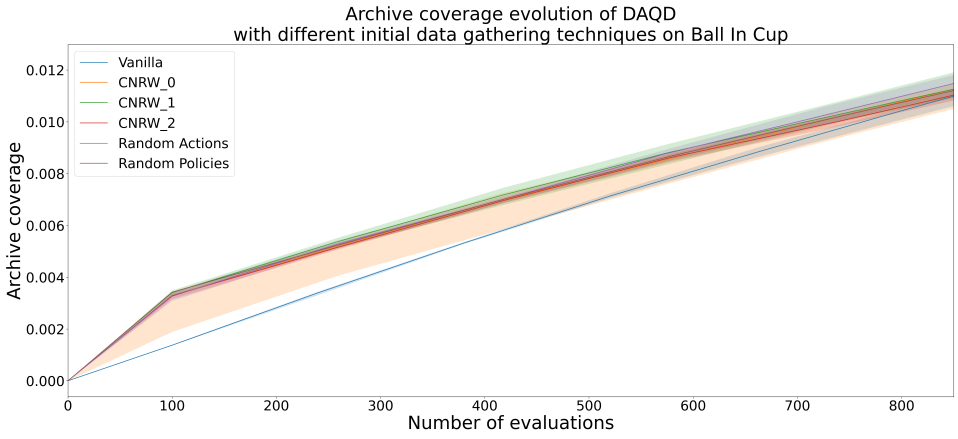
Pour conclure, dans ce chapitre nous explorons le lien entre la consistence des actions dans un environnement, le degré d'auto-corrélation de méthode de récolte initiale de données et l'impact sur la qualité d'un modèle appris avec ces données.



(a) Two-Wheeled Robot Maze



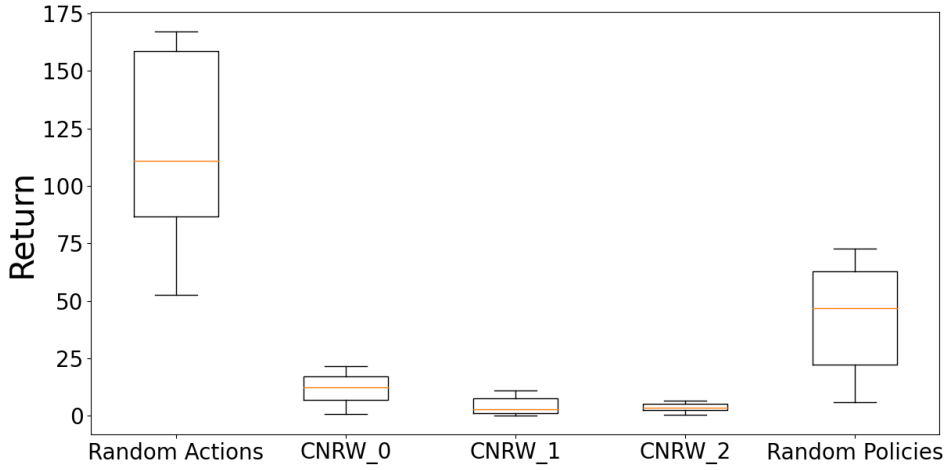
(b) Two-Wheeled Robot Maze Traps



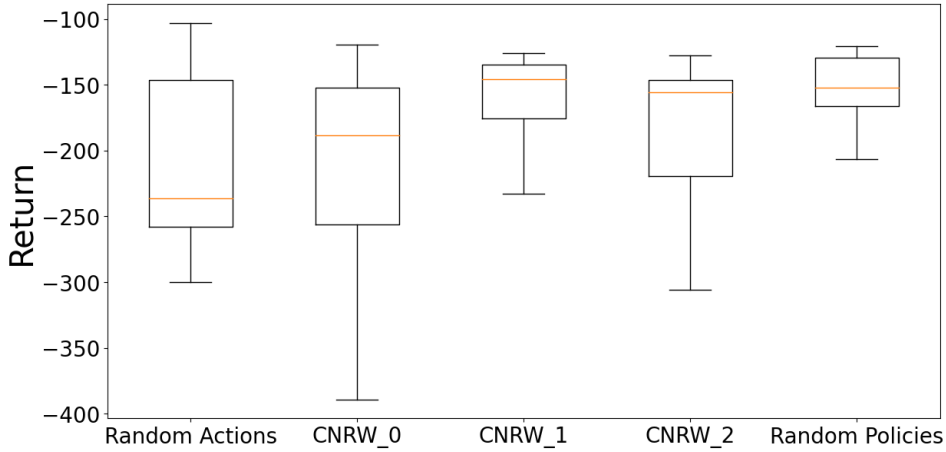
(c) Ball In Cup

Figure 1: Impact of different initial data gathering methods on DAQD algorithm ten first generations

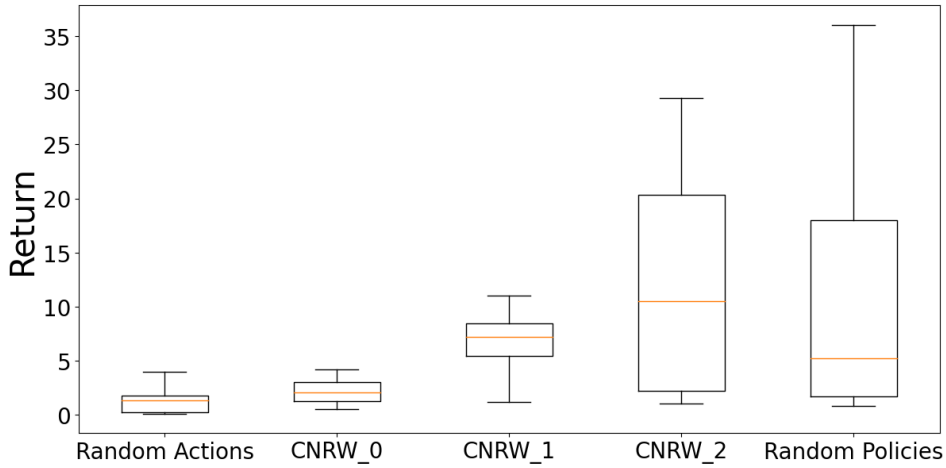
De plus, nous démontrons que la qualité du modèle peut parfois avoir un impact très important sur la méthode de recherche de politique, pouvant entraîner des écarts de performance jusqu'à dix fois supérieur lorsqu'une méthode appropriée est utilisée. Nous montrons également que certains algorithmes sont plus sensibles que d'autres



(a) Cartpole



(b) Pusher



(c) Ball In Cup

Figure 2: Impact of different initial data gathering methods on PETS algorithm first iteration return

à ce phénomène.

Des conclusions du chapitre précédent, nous supposons que si l'initialisation du modèle peut avoir un impact fort sur la méthode de recherche qui s'en suit, d'autres facteurs peuvent également l'impacter. Dans le cas de l'algorithme Novelty Search, un facteur potentiellement important est la population de politiques utilisée comme point de départ pour la recherche de diversité. Usuellement, cette population est initialisée avec un ensemble de politiques paramétrées aléatoirement de manière uniforme dans l'espace des paramètres. Nous proposons dans ce chapitre de biaiser la population initiale, de telle sorte qu'elle ne soit pas uniforme sur l'espace des paramètres mais plutôt concentrée sur des régions prometteuse de ce même espace.

Pour déterminer les régions de cet espace pouvant être prometteuses, nous proposons d'utiliser des ensembles de modèles dynamiques paramétrés aléatoirement pour caractériser des politiques nouvelles sur l'ensemble de ces modèles. Les politiques trouvées étant diverses sur l'ensemble de ces environnements, l'objectif est qu'ensuite ces politiques soient également plus diverses sur le système réel que des politiques paramétrées aléatoirement. Pour trouver ces politiques, nous proposons d'utiliser l'algorithme Novelty Search avec des mesures de nouveauté adaptées à l'utilisation d'ensemble d'environnements (les modèles aléatoires). Nous proposons donc deux méthodes de mesures de nouveauté, N_{min} et N_{sum}, calculant respectivement le minimum de nouveauté d'une politique sur l'ensemble d'environnement, et la seconde la somme de la nouveauté sur l'ensemble des modèles aléatoires. Nous proposons également deux représenations de modèle aléatoires, les Spatial Random Fields [60] et les Réseaux de Neurones Non-Linéaires, les premiers donnant des transitions Gaussiennes et l'évolution des transitions en fonction de l'état-action en entrée étant continu, les seconds n'ayant aucune garantie de continuité pour deux entrées proches les unes des autres mais ayant potentiellement plus de capacité de représentation. Finalement, à l'issue de Novelty Search sur l'ensemble de modèle, un ensemble d'individus (de la taille de la population) est selectionné à l'aide d'une de deux méthodes proposée: soit les individus les plus nouveaux générées au cours de Novelty Search sur l'ensemble de modèle aléatoires sont sélectionnés, avec pour objectif qu'ils soient le plus nouveau possible sur le système réel, ou la population finale à l'issue de Novelty Search est sélectionnée, avec pour objectif que ces individus soient les plus évolvables [44].

L'algorithme proposé, 0DAB, est évalué sur deux environnements, le premier étant une tâche de navigation d'une base mobile à deux roues dans un environnement ouvert, le seconde étant une tâche de locomotion d'un robot hexapode. La première tâche ayant une dynamique uniforme, nous nous en servons comme tâche jouet permettant d'évaluer la viabilité de la méthode proposé et de déterminer quelle

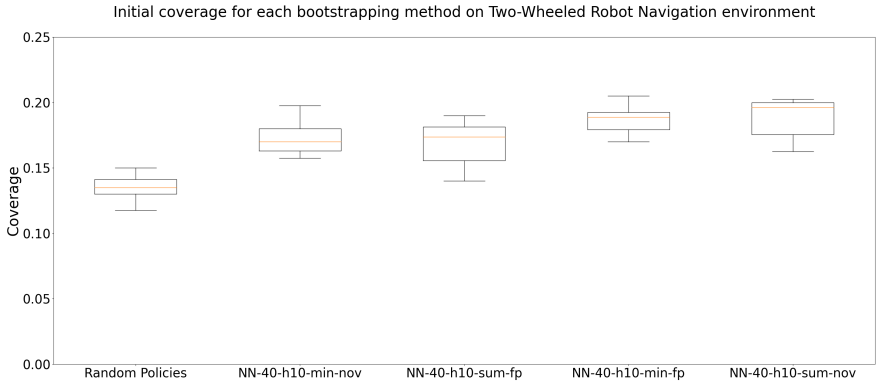


Figure 3: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Non-Linear Neural Network ensembles of size 40 on Two-Wheeled Robot Navigation Task

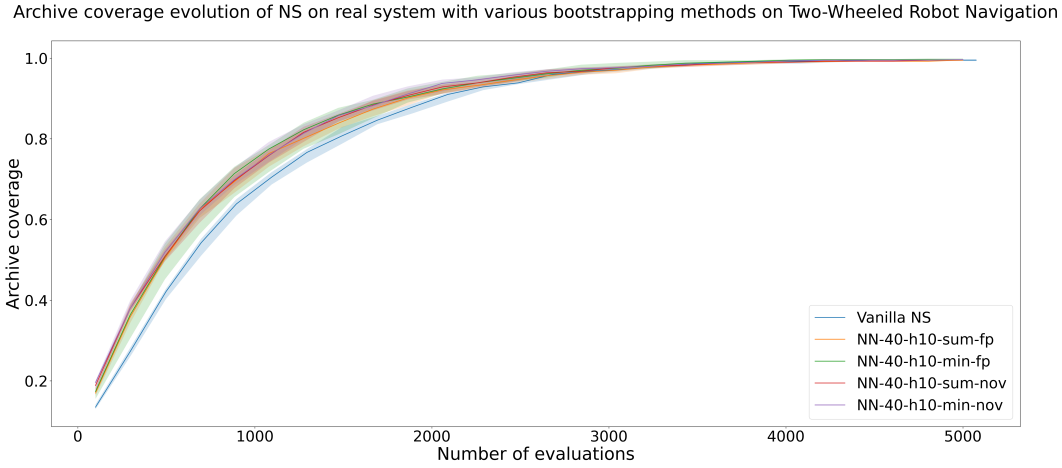


Figure 4: Novelty Search coverage evolution comparison between randomly parameterized policies and 0DAB selected bootstraps obtained with Non-Linear Neural Networks ensembles of size 40 on Two-Wheeled Robot Navigation Task

représentation de modèle peut-être la plus intéressante à utiliser. Il en ressort que les Spatials Random Fields et leur dynamique simple ne parviennent pas à générer des individus qui maintiennent leur diversité sur l'environnement réel, tandis que les réseaux de neurones non-linéaires y parviennent et surpassent la diversité atteinte par une population de politiques paramétrées aléatoirement d'environ 35% commme montré sur la figure 3. De plus, la figure 4 montre qu'utiliser cette population comme point de départ réduit de plusieurs centaines le nombre d'évaluations nécessaires pour couvrir totalement l'environnement considéré.

Les résultats sur le second environnement sont plus mitigés, 0DAB ne parvenant pas à générer une population d'individus ayant une couverture initiale de l'espace

comportemental nettement supérieur à celui d'un ensemble de politique paramétrées aléatoirement, et ce malgré l'exploration d'horizon de planification plus grand sur le modèle comme montré sur la figure 5. De plus, la figure 6 montre que même lorsque la couverture initiale est plus importante, l'utilisation de 0DAB biaise la recherche évolutionnaire dans des zones sous performantes de l'espace des paramètres, et mène à une évolution de la couverture de l'espace comportemental plus lente qu'avec des politiques paramétrées aléatoirement de manière uniforme dans l'espace des paramètres.

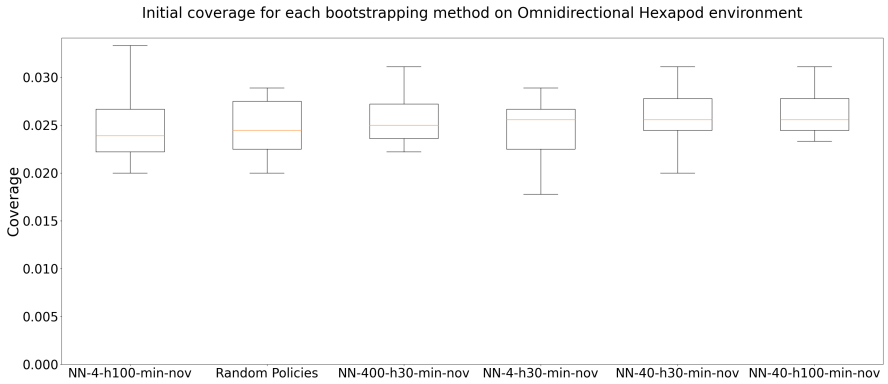


Figure 5: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Non-Linear Neural Network ensembles of size 4 and 40 and a prediction horizon of 30 and 100 time-steps on Omnidirectional Hexapod Locomotion Task

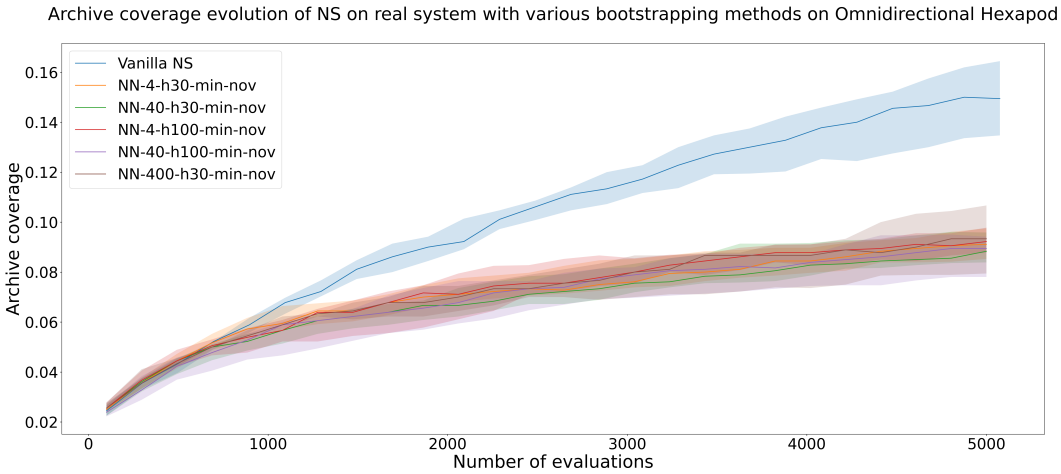


Figure 6: Novelty Search coverage evolution comparison between randomly parameterized policies and 0DAB selected bootstraps obtained with Non-Linear Neural Networks ensembles of size 4 and 40 and a prediction horizon of 30 and 100 time-steps on Omnidirectional Hexapod Locomotion Task

Pour comprendre ce phénomène, une analyse supplémentaire est faite. L'algorithme

Novelty Search est lancé sur le système réel, et les individus ainsi générés sont ensuite transférés sur l'ensemble de modèle et l'on observe leur descripteur comportemental sur l'ensemble de modèle. Ces descripteurs comportementaux sont comparés à ceux obtenus avec 0DAB sur ce même ensemble de modèle, et l'on observe la différence de couverture sur l'ensemble de modèle de chacune de ces deux approches. La figure 7 montre ce qui ce passe lorsque 0DAB fonctionne bien sur l'environnement de navigation d'une base mobile à deux roues. Un alignement des solutions de Novelty Search obtenues sur le système réel avec celle trouvées par 0DAB est observé: l'ensemble de modèles aléatoires choisi permet de capturer un mapping de l'espace des paramètres vers l'espace comportemental qui est similaire, la diversité des solutions étant préservée entre les deux. La figure 8 quand à elle montre ce qui se passe lorsque la représentation de modèle n'est pas la bonne pour générer des solutions diverses sur le système réel. En effet, les solutions issues de Novelty Search sur le système réel s'écrasent toutes dans une petite région de l'espace comportemental sur les modèles aléatoires, rendant la sélection de solutions diverse avec 0DAB très difficile.

Pour conclure, dans ce chapitre nous avons présenté une étude préliminaire sur un nouvel algorithme appelé 0DAB ayant pour objectif de générer une population d'individus plus diverse que des politiques paramétrées aléatoirement. Pour ce faire, nous proposons d'utiliser des modèles dynamique aléatoires et des métriques de nouveauté adaptées. Nous montrons l'efficacité de la méthode sur une tâche de navigation d'une base mobile à deux roues, augmentant la couverture initiale de 35% par rapport aux politiques paramétrées aléatoirement et réduisant de plusieurs centaines le nombre d'évaluations nécessaire pour couvrir complètement l'environnement. Néanmoins, sur une tâche avec une dynamique moins uniforme, la locomotion d'un robot hexapode, 0DAB échoue à produire une population initiale significativement plus diverse que celle des politiques paramétrées aléatoirement et biaise même négativement la population en réduisant significativement la vitesse de d'évolution de la couverture de l'environnement avec Novelty Search. Nous avons ensuite mené une analyse montrant que ce phénomène s'explique par la difficulté à caractériser les solutions considérées comme diverses sur la seconde tâche avec les ensembles de modèles aléatoires de la dynamique considérés.

Le dernier chapitre s'intéresse directement à la réduction du nombre d'évaluations nécessité par Novelty Search directement au cours du processus d'optimisation. L'objectif est de réduire le nombre d'évaluations nécessaire pour atteindre une couverture donnée. Pour ce faire, nous proposons d'incorporer un modèle de la dynamique de l'environnement dans la boucle de Novelty Search. L'objectif est donc de réduire

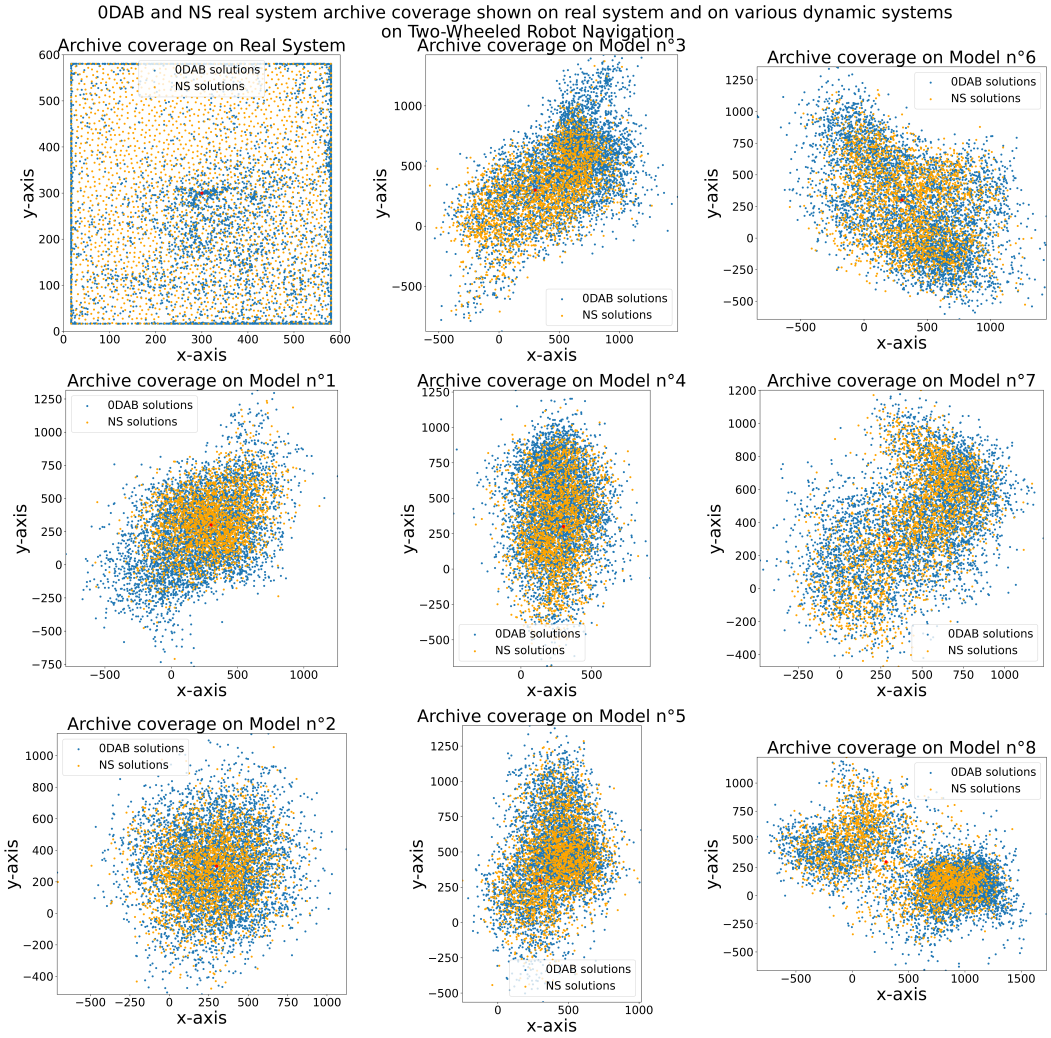


Figure 7: Archive Coverage after 0DAB on the real system and on the models of the Non-Linear Neural Network Ensemble of size 40 compared to that of Novelty Search on Two-Wheeled Robot Navigation Task

le nombre d'évaluations d'individus en pratique peu nouveaux, en estimant *a priori* leur nouveauté grâce à ce modèle de la dynamique. L'algorithme proposé, Model-Based Novelty Search, repose donc en son centre sur l'algorithme Novelty Search classique, executé sur le modèle appris. Novelty Search est exécuté sur le modèle pour un nombre de génération adaptatif, basé sur le nombre d'évaluations sur le système totales actuelles, de telle sorte que mieux le modèle est appris, plus le processus d'optimization de Novelty Search sur le modèle est long. A l'issu du budget de générations défini sur le modèle, les s_p individus les plus nouveaux parmi tout ceux générés sur le modèle sont transférés sur le système réel, s_p étant la taille de la population. Ces mêmes individus sont ensuite utilisés comme point de départ pour

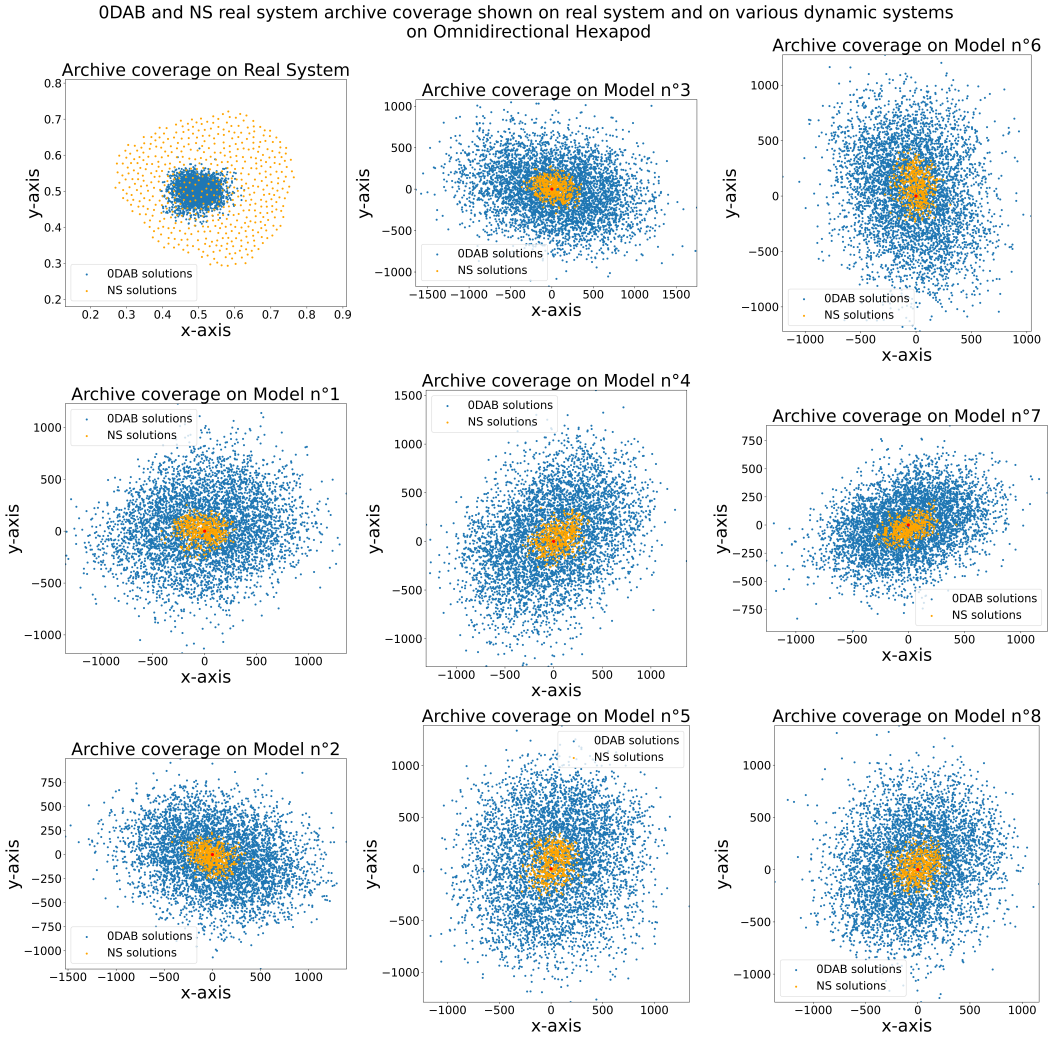


Figure 8: Archive Coverage after 0DAB on the real system and on the models of the Non-Linear Neural Network Ensemble compared to that of Novelty Search on Omnidirectional Hexapod Locomotion Task

la prochaine boucle de l'algorithme Novelty Search sur le modèle, et ce processus est répété jusqu'à exhaustion du budget d'évaluations sur le système réel.

Nous évaluons la performance de Model-Based Novelty Search contre celle de Novelty Search, de Vanilla Quality-Diversity et de Dynamics-Aware Quality-Diversity sur trois environnements différents: une tâche de locomotion d'un robot hexapode, une tâche de navigation d'une base mobile à deux roues dans un environnement ouvert et une tâche de bilboquet. Les figures 9, 10 et 11 montrent que sur les trois environnements considérés, Model-Based Novelty Search réduit le nombre d'évaluations nécéssaires pour atteindre une couverture similaire par rapport à Novelty Search de 30% à 75%, un gain significatif. De plus, Model-Based Novelty Search réduit égale-

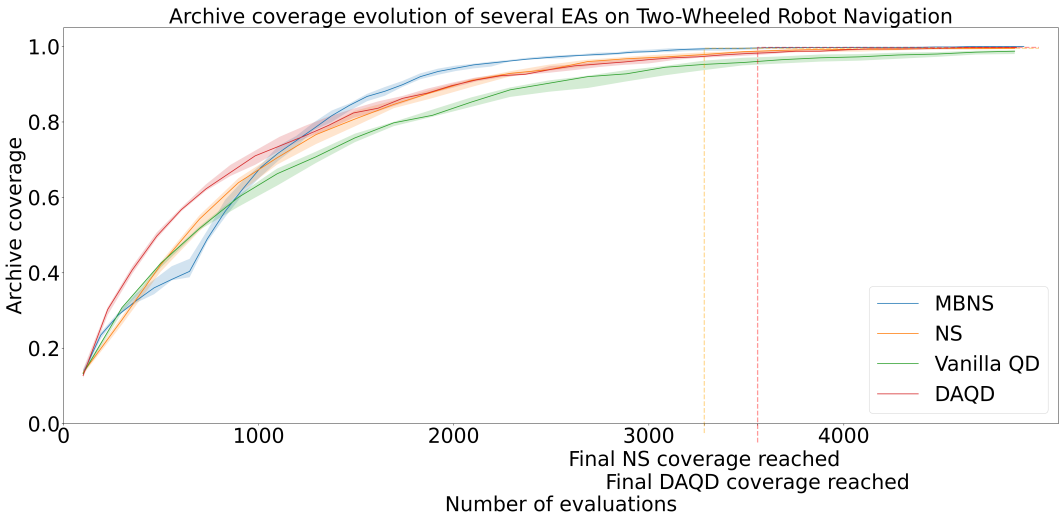


Figure 9: Coverage of Diversity Methods on the Two-Wheeled Robot Navigation task depending on number of evaluations

ment le nombre d'évaluations nécéssaires pour atteindre une couverture similaire à celle de Dynamics-Aware Quality Diversity de 30% à 50%.

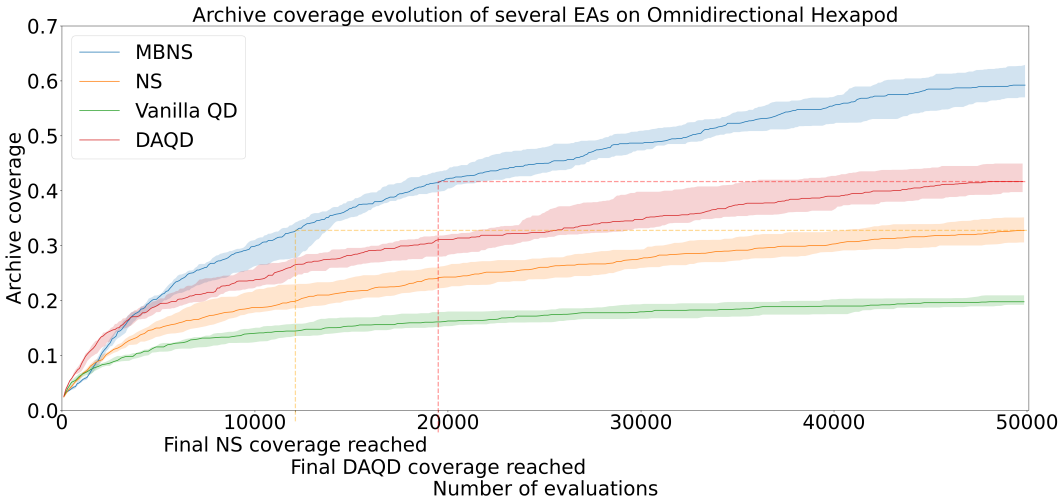


Figure 10: Coverage of Diversity Methods on Omnidirectional Hexapod task depending on number of evaluations

On observe sur les figures 9 et 10 que Model-Based Novelty Search a tendance à sous-performer en terme de couverture atteinte lorsque le nombre d'évaluations sur le système réel est encore faible. Nous montrons plus en détail dans le chapitre 5 que cela est dû a une sur-exploitation du modèle avec l'objectif de transfert des individus les plus nouveaux, ces individus étant souvent au début ceux dont le descripteur comportemental est le plus mal estimé. Dans le chapitre 5, nous proposons donc une nouvelle approche évolutionaire basée modèle, Model-Based Novelty Search, qui

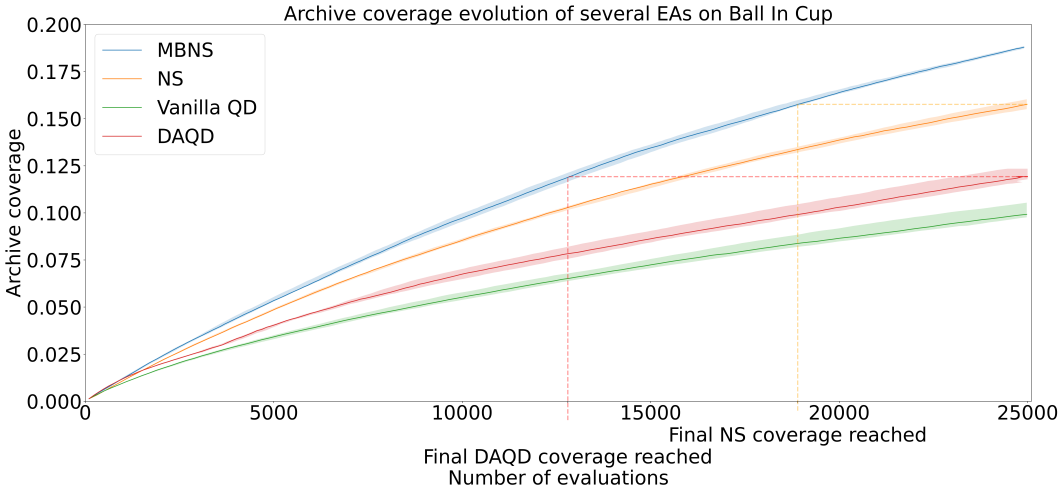


Figure 11: Coverage of Diversity Methods on Ball In Cup task depending on number of evaluations

préserve les capacités d'exploration forte de Novelty Search tout en réduisant le nombre d'évaluations nécessaires de 30% à 75%.

Pour conclure, nous avons dans cette thèse répondu à la question de la réduction du nombre d'évaluations pour l'algorithme Novelty Search a travers trois volets. D'abord, nous avons étudié comment la méthode initiale de récolte de données utilisées pour pré-entraîner le modèle pouvait impacter une méthode de recherche de politique basée modèle, en montrant que la différence de récompense obtenue par une méthode d'apprentissage par renforcement basée modèle pouvait être jusqu'à 10 fois supérieur lorsqu'une méthode appropriée est utilisée. Ensuite, nous avons proposé une approche préliminaire permettant d'augmenter la couverture initiale d'un ensemble de politique par rapport à des politiques paramétrées aléatoirement sur un environnement de navigation simple. Nous avons également étudié les limites de cette technique ainsi que les raisons de ces limites. Finalement, nous avons proposé une nouvelle méthode de recherche de politique basée modèle inspirée de l'algorithme Novelty Search permettant de réduire le nombre d'évaluations nécessaire pour atteindre une couverture similaire à celle de Novelty Search de 30% à 75% en fonction de l'environnement considéré. La question de la réduction du nombre d'évaluations de Novelty Search reste un problème ouvert auquel de nombreuses pistes méritent toutefois d'être encore étudiées, mais cette thèse aura su apporter quelques éléments de réponses et de réflexion pour progresser dans ce domaine.

Acknowledgments

When you start a Ph.D. thesis, you believe that you will revolutionize science and achieve breakthroughs after breakthroughs. However, you also quickly realize that this is not how science works, and that it is all the small steps that you take consistently that actually lead to the findings you are thriving for. Maintaining consistency has been hard, but through hardships and better times a lot of people helped me see the end of the tunnel that a Ph.D. thesis is.

For this, I want to thank here everyone that helped me, through technical and moral support over the 4 years my Ph.D. thesis lasted. Firstly, I would like to thank my supervisors for their unwavering help throughout those years, for believing in me and for sticking with me through the good and bad times I had during my Ph.D. thesis. I would particularly like to thank Stéphane Doncieux, my supervisor, for pushing me towards the finish line when I thought I had already used up all the energy I had stored. Moreover, I would like to thank him for all the precious help he gave me through countless advice and patience shown towards me even when I wanted to rush everything at the risk of misstepping. I would also like to thank Miranda Coninx and all the curiosity-driven conversations we had, helping me grasp better the direction I wanted my work to take. I would also like to thank all the ISIR team, for providing the perfect workspace to develop my thoughts and ideas into concrete algorithms and results. I especially want to thank all the members of the AMAC team, for their kindness and for all the interesting conversations we had over the course of 4 years. Being part of that team made me proud, as the emulsion of thoughts coming from it was like no other, and I hope it will remain like that in the many years to come through the new teams that have been formed. I would like to thank the whole team of the SoftManBot European Project, for all the scientific discussions we had during international meetings solving industrial problems with an academic approach.

I want to thank my family for all the simple moments I shared with them that helped me keep my head up and my thoughts clear during this Ph.D. thesis. I would like to thank especially my mother Isabelle for the support, love and compassion she shown towards me during all these years, even when I remained secluded for work several days in a row. I would also like to thank my second family, my friends, for

whom I know some for more than 10 years. I want to thank them for making me laugh and forget the hardships I was going through, helping me racing towards the end of this Ph.D. thesis. I would particularly like to thank Brian, Ashkan, Alexis, Luca, Leo, Julien and Julian and all the others that are too many to cite here.

Finally, I want to thank the members of the Jury of my Ph.D. thesis defense for accepting to evaluate the work I did over these years. Their opinion will mean a lot to me as it will bring a conclusion to the work I led for 4 years and help me relativize the contributions I made to the Evolutionary Robotics community.

The last thing I would like to bring up is something Tom Platz said, which has struck a chord with me: "Failure has been achieved, thank God. Now the only place to go from failure is to win."

Thanks again to everyone!

Contents

Chapter 1

Introduction

In human imagination, robots have emerged as captivating creations capable of acting in a human-like way in almost any scenario. As robots are seen with a capacity for precise and efficient task execution, Humanity dreams of robots imbued with the same physical and logical capabilities as humans. Towards this robots nowadays tend to be placed in more and more complex task setups to solve, from manipulating objects that have high order dynamics [12] to evolving in open-ended setups where everything is yet to learn [94]. Nevertheless, as of today, controlling a robotic system can still be a difficult task. To deal with such situations, robots have been more and more coupled with learning techniques, allowing them to learn from interactions with their environments to either solve a specific task or discover a variety of skills within their current capabilities. This domain is called robot learning, and is critical domain within artificial intelligence that focuses on endowing robots with the capability to acquire knowledge and skills through data-driven processes. Employing machine learning algorithms, robots undergo training to adapt their behavior based on data gathered from their environment. Various paradigms are employed to facilitate this adaptive process.

A lot of the recent interest in robot learning has stemmed from one of these paradigms, called Reinforcement Learning [151]. Reinforcement Learning is a subfield of machine learning that addresses the challenge of training agents to make decisions in environments to achieve specific goals. Unlike supervised learning, where training data is labeled with correct outputs, and unsupervised learning, where the agent identifies patterns without explicit guidance, Reinforcement Learning operates through interactions with an environment, learning from feedback in the form of rewards or penalties. The Reinforcement Learning framework has proven highly suc-

cessful in applications involving sequential decision-making tasks, including robotics [85, 176], traditional games [146], video games [164] and other autonomous systems [74].

At the heart of Reinforcement Learning lies the Markov Decision Process, a mathematical framework that formalizes sequential decision-making. The Markov Decision Process encapsulates the dynamics of the environment, specifying the state space, action space, transition function, and reward structure. The agent's goal is to learn an optimal policy that maximizes the expected long-term rewards within this framework, a policy being a strategy that maps states to actions.

Another family of optimization methods used for policy search are Evolutionary Algorithms [110]. With origins deeply rooted in Charles Darwin's theory of evolution, Evolutionary Algorithms are characterized by their capacity to mimic nature's iterative process of selection, recombination, and mutation to evolve solutions that solve a given task. Central to this paradigm is the notion of a population of candidate solutions, each representing a possible answer to a given problem. The algorithm iteratively refines this population, advancing it toward increasingly effective solutions through aforementioned processes. The guiding principle is to replicate the way species evolve over generations, where the most successful traits are inherited and accumulate, yielding increasingly fit organisms.

A sub-family of algorithms within Evolutionary Algorithms is Diversity Algorithms [91, 133]. Such approaches have emerged as promising solutions to the problem of exploring the agent environment. These approaches focus on promoting diverse behavior in a user-defined Behavioral Space, leading to the discovery of a broad range of solutions beyond the traditional reward-based exploration methods. One notable diversity-driven approach is Novelty Search [91], which encourages exploration by rewarding agents based on the novelty of their behaviors rather than explicit rewards. The concept of novelty, introduced by Lehman and Stanley [93], quantifies how different an agent's behavior is from previously encountered behaviors in the environment. By maximizing novelty, agents are incentivized to explore new regions of the Behavioral Space, increasing the likelihood of discovering novel and potentially beneficial behaviors. Novelty Search has shown promise in solving challenging tasks where sparse rewards or deceptive landscapes make traditional Reinforcement Learning methods ineffective, such as robotic control tasks [80, 21, 76], maze navigation [93], and multi-agent systems [54].

Despite those advantages, Diversity Algorithms also face their own challenges. In-

deed, the design of the Behavioral Space can significantly impact the performance of these methods. Researchers continue to explore various techniques to overcome these challenges, such as intelligently adapt the diversity algorithms to more complex Behavioral Space representations [115] or directly learning the Behavioral Space [127]. Moreover, diversity-driven approaches share the same hurdles as most data-driven techniques, notably Reinforcement Learning: they require an important amount of environment interactions to find rewarding policies or cover the user-defined Behavioral Space. Such techniques thus call for the usage of a low sample cost simulator, which is not always available depending on the task to be tackled.

Indeed, simulation environments have become indispensable tools for developing and evaluating robot algorithms and control systems [30, 157, 28]. They offer numerous advantages, including cost-effectiveness and faster development cycles. However, a notorious problem arises when the simulator is not close enough to the real environment [80, 107]. This problem is called the *Reality Gap* [66] and poses a significant challenge in robotics, referring to the disparity between a robot's performance in simulation and its performance in real-world settings. Several key factors contribute to the reality gap, such as physics and dynamics, sensor and actuator models, environment modeling and adaptability.

Researchers have explored several approaches to mitigate the reality gap and improve the transferability of robotic skills from simulations to real-world scenarios. Four main approaches have been proposed across the years: System Identification [30, 157, 28, 1, 83], Domain Adaptation [69, 48, 101, 21, 76, 16, 67, 159], Transferability [84] and Domain Randomization [156, 129, 138, 109]. But all these approaches cannot always close the reality gap when the simulation does not hold enough information about the real-world environment. A promising way to be able to overcome the reality gap in most cases is by using a learned model of the real-world environment. Even though model-based methods require interactions with the robot's real environment, which are costly to gather, it allows to directly refine the robot's behavior given the real-world dynamics it will follow.

Model-based approaches in robotics either use a mathematical or learned model of the robot and its environment to control and optimize the robot's behavior. Model-based approaches are crucial for enhancing the robot's performance, as they enable the robot to predict its future actions and outcomes based on the model, allowing for better decision-making and planning. Two prominent model-based techniques in robotics are Model Predictive Control [19] and Model-based Reinforcement Learning (MBRL) [131, 113]. A more recent approach in robotics is Model-Based Evolution-

ary Algorithms (MBEA) [49]. In this thesis, we will focus on approaches using a completely learned model, as mathematical models or simulators can be either too slow to use with data-oriented techniques or too inacurate to successfully transfer onto the real-world.

Both Model-based Reinforcement Learning [113] and Model-Based Evolutionary Algorithms [49] are approaches where the robot learns a model of the environment dynamics from data collected during exploration. This learned model is then used for planning and decision-making. The main difference between both techniques resides in the output of the Policy Search algorithm, MBRL giving a single policy while MBEA outputs a repertoire of policies given a user-defined Behavioral Space. One of the main advantages of Model-Based techniques based on a learned model is their sample efficiency [37].

However, using a learned model can also bring some hurdles. The accuracy of the learned model is critical for the success of MBRL algorithms. Errors in the model can lead to inaccurate predictions and sub-optimal decision-making. Additionally, MBRL requires a balance between exploration and exploitation. While it uses a learned model to guide planning, the robot must still explore the real environment to improve the model's accuracy and avoid over-fitting. On the contrary, MBEA techniques require more samples to converge than MBRL algorithms, which is inherent to the goal of Policy Search methods whose goal is to cover a an outcome space that can be hard to explore completely [43].

The landscape of Model-Based techniques has still not be completely explored, especially in the field of Diversity Algorithms. Exploring how to design and enhance such methods in the context of robotic tasks is thus an interesting domain that deserves to be explored, as Diversity Algorithms have proven to propose a strong paradigm helping solving various complex robotic tasks, one of the last bastion preventing it from wider adoption being its sample-efficiency.

In this context, this Ph.D thesis revolves mostly around Diversity Algorithms, the novelty driven sub-class of Evolutionary Algorithms. It studies various ways of increasing their sample-efficiency, especially for the Novelty Search algorithm through the usage of learned models. More precisely, the general question that will be answered in this manuscript is:

How to reduce the number of evaluations required by Novelty Search to cover the Behavioral Space ?

Each chapter will answer this question in different ways, aiming at tackling several phases of the considered Model-Based Policy Search algorithms.

Enhance Initial Model Performance for Robotics Tasks

In chapter 3, the focus is made on the model bootstrapping part of Model-Based techniques. We study the impact on learned model performance of the random processes used to gather initial training data before the Model-Based Policy Search algorithm is used. The question answered is thus:

Does randomly gathered initial training data for model initialization has an impact on Model-Based Policy Search, and how those random processes compare ?

To answer that question, we demonstrate a link between a consistency metric introduced to measure the consistency of actions across the State Space of a Markov Decision Process and the time-correlation of the random processes used to gather initial training data for learning the model.

Model prediction error using low data budgets typically used in Model-Based Policy Search techniques is thus studied and the impact of each of the initial data gathering method is evaluated on a set of five environments with various consistency over actions. Moreover, a clear impact on a state-of-the-art step-based Model-Based Reinforcement Learning algorithm, PETS [26], is shown with varying initial returns obtained by the algorithm up to 10 times greater when the model is initialized with the suited random process. Similarly, an impact is observed on a state-of-the-art episode-based Model-Based Diversity Algorithm, Dynamics-Aware Quality-Diversity [98], with initial coverage metrics being slighlty affected by the random process used to pre-train the learned model.

Population Bootstrapping with Random Dynamics Models

Chapter 4 shifts the focus from the model to the initial population used in the Novelty Search algorithm and answers the following question:

How to bias the initial population without knowledge of the target task to increase initial population coverage and subsequent Novelty Search sample-efficiency ?

A preliminary study is conducted with such an objective. The proposed approach,

dubbed 0DAB for Zero-Shot Diverse Archive Bootstrapping, aims at leveraging randomly parameterized dynamics models to generate a set of diverse solutions to be used as the starting population of a Novelty Search routine on the real system. Ensembles of random dynamics models are used, and novelty metrics aiming at finding individuals that are as novel as possible on all environments are used to generate archives of generally diverse behaviors.

The approach successfully demonstrates a better initial coverage and increased sample-efficiency on a simple robot navigation task with smooth dynamics, reducing the number of evaluations required by Novelty Search to cover the environment by several hundred samples. The approach is then evaluated on a second and less smooth real dynamical system, an omnidirectional hexapod locomotion task. On this environment, the approach fails to reach our objective. The reasons for failure are analyzed and the random dynamics models representation being used is determined as a culprit as interesting solutions on the models collapse to small regions of the random dynamics models ensemble Outcome Space.

Model-Based Novelty Search

Finally, chapter 5 considers creating a novel Model-Based Policy Search algorithm completely, taking inspiration from Novelty Search. This new Model-Based Evolutionary Algorithm answers the question:

How to increase Novelty search sample-efficiency using learned dynamics models ?

A new Model-Based technique is thus proposed, dubbed Model-Based Novelty Search. This approach leverages the classic Novelty Search algorithm to generate a diverse set of individuals on a learned dynamics model. The most novel individuals generated using the model are then transferred onto the real system, and serve as the starting population for the next Novelty Search loop on the learned dynamics model.

This new policy search method performance in terms of coverage of the Behavioral Space is evaluated on three different robotic environments and compared against three baselines: Novelty Search, Vanilla Quality-Diversity and Dynamics-Aware Quality-Diversity. The new algorithm outperforms all three baselines in terms of sample-efficiency as it reduces the number of evaluations required to reach the final coverage of Novelty Search by 30% up to 75% depending on the considered

environment. Moreover, when compared against Dynamics-Aware Quality Diversity, Model-Based Novelty Search reduces the number of samples required to reach its final coverage between 30% and 50%.

To summarize, this Ph.D thesis work presents three contributions, all falling in the field of increasing sample-efficiency of Policy Search algorithms, and more specifically of the Novelty Search algorithm. The first contribution consists in helping scientists chose a better initialization technique for their Model-Based Policy Search techniques. The second contribution consists in a preliminary work on biasing the initial Novelty Search population to be more diverse and lead faster to more diverse individuals and finally the last contribution consists in a novel Model-Based Evolutionary Algorithm, Model-Based Novelty Search, which enhances greatly the sample-efficiency of the classic Novelty Search algorithm.

Chapter 2

Background And Related Work

2.1 Reinforcement Learning

As stated previously, Reinforcement Learning (RL) has emerged as a versatile approach for solving various problems, ranging from robotics [81] to healthcare [173] and economics [118]. The fundamental goal of RL is to learn a policy. The policy guides the agent's decision-making process by specifying the action to take given a state the system is in. The objective of RL is thus to find the policy that maximizes the expected cumulative reward over time, the agent receiving a positive or negative reward from the environment depending on the agent's current state and action taken toward solving the task.

Doing so and as shown on figure 2.1, the agent iteratively acts in its environment (according to its policy that takes into account the current state the agent is in), observe the change in state and the reward that is returned by his environment and start acting again, updating its policy periodically. This trial-and-error learning loop relies on a Markov Decision Process, a generic framework essential to RL and used to describe problems that respect the Markov property.

2.1.1 Markov Decision Process

The Markov property allows an important simplifcation of the dynamics of various systems, as it states that the transition from a state in the system to another state in the system solely depends on the current state and action taken, not on the past events. It is a strong hypothesis that nevertheless remains valid in most applications,

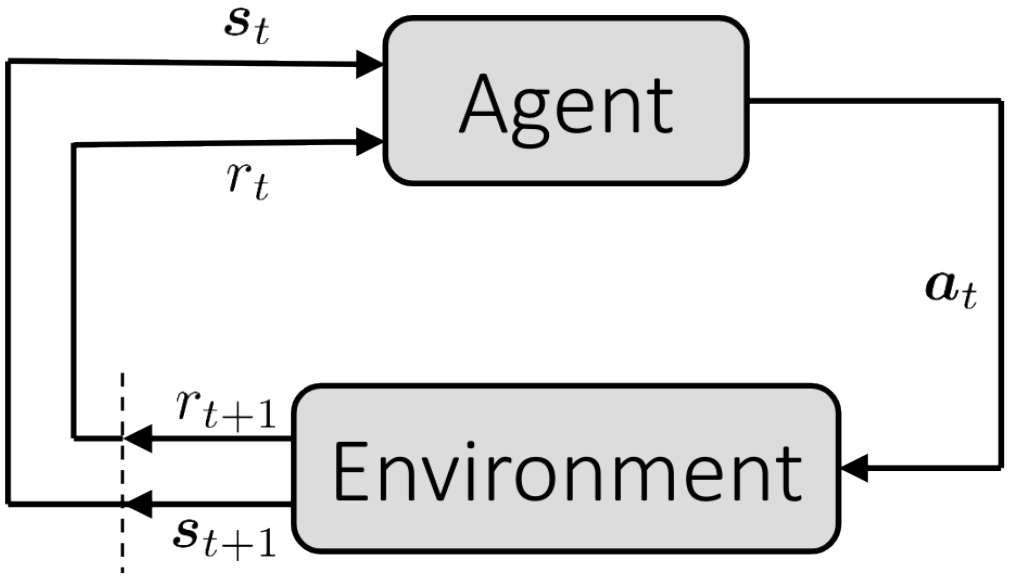


Figure 2.1: MDP with an agent interacting with a single environment

as long as the state does contain all the information needed to infer about its future.

Thus, a Markov Decision Process (MDP) constitutes a fundamental framework for modeling decision-making processes. Formally, a MDP is denoted as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ represents the set of states, $\mathcal{A}$ the set of actions, T denotes the transition function, and R represents the reward function.

In the context of MDPs, the elements of the MDP can be detailed as such:

- $\mathcal{S}$ is the **state-space** denoted as $\mathcal{S} \subseteq \mathbb{R}^{d_s}$, where d_s represents its dimensionality,
- $\mathcal{A}$ is the **action-space** denoted as $\mathcal{A} \subseteq \mathbb{R}^{d_a}$, where d_a is its dimensionality,
- T is the **transition function**, which associates to a current state s and action a the next state s' :

$$T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$$
and
$$T(s, a) = s'$$

with $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$. T thus describes how the agent's actions impact the system state and provide the complete system transition dynamics,

- R is the **reward function**, which associates a scalar value called reward to each state the system could be in, effectively guiding the agent toward solving its given task:

$$R : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$$
and
$$R(s, a) = r$$

  with r the scalar reward associated with state s.

- π is a **policy**, which is a function that returns an action $a \in \mathcal{A}$ to take in a state $s \in \mathcal{S}$:

$$\pi : \mathcal{S} \longrightarrow \mathcal{A}$$

- Θ is the **parameter space**, corresponding to the space of parameters used to define the policy π, such that if a policy is parameterized with a vector of parameters $\theta \in \Theta$, it will be denoted as π_θ

2.1.1.1 Estimating the expected reward

Solving MDPs involves determining an optimal policy $\pi^* : \mathcal{S} \longrightarrow \mathcal{A}$ that returns the best action to take in each state to maximize the expected cumulative reward. When dealing with parameterized policies, finding the optimal policy $\pi*$ thus mean finding the set of parameters $\theta^* \in \Theta$ that corresponds to the optimal policy π^*. For ease of reading, we usually won't explicit that policy are parameterized by set of parameters. In order to estimate the expected cumulative reward, two functions, the Value function and Q function, constitute the key-components of most RL algorithms, being used in various ways to optimize the agent policy. The Value function provides estimates of the expected rewards attainable in a specific state, while the Q function provide such an estimate for a specific state-action pair.

The Value function, denoted as $V^\pi(s)$, is the anticipated cumulative reward an agent can receive starting from state s, while following policy π. Formally, it is characterized as the expected summation of discounted future rewards, with the discount factor γ modulating the relative significance of immediate versus distant rewards. The Value function is mathematically expressed as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right] \tag{2.1}$$

Here, r_t signifies the reward obtained at time step t in state s_t obtained following policy π, and $\gamma \in [0, 1]$ is the discount factor, crucial for striking a balance between rewards closer or further into the future.

Secondly, the Q function, symbolized as $Q^\pi(s, a)$, is the projected cumulative reward an agent can receive from a state s, given that it initiates action a, and subsequently abides by policy π. The Q function accounts for the potential rewards of opting for action a in state s and subsequently following the policy π. The Q function is mathematically expressed as:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right] \tag{2.2}$$

In the above equation, a_0 denotes the initial action chosen by the agent, and r_t is defined as in equation 2.1.

The Value function and Q function thus serve as two cornerstones functions to estimate the reward of a given state or state-action pair in multiple Reinforcement Learning algorithms. Both of these functions estimates are learned and updated iteratively using experience gained from interactions between the agent and the environment.

2.1.1.2 Brief Overview

One method using the Value function is Value Iteration. Value Iteration stands as a foundational algorithm in reinforcement learning for solving Markov Decision Processes. This method, introduced by Bellman [10], computes optimal value functions iteratively by updating estimates based on the Bellman optimality equation. Modifications like Prioritized Sweeping [114] prioritize state updates for faster convergence thus using less data samples. As RL advances, Value Iteration remains relevant for its simplicity and theoretical significance in solving optimal control problems.

As another foundational algorithm in RL, Q-learning [165] consists in iteratively refining action-value estimates by balancing immediate rewards and long-term outcomes. Firstly proposed by Watkins and Dayan [165], recent enhancements like Deep Q-Networks [112] leverage deep neural networks to handle high-dimensional state spaces. Indeed, combining Q-learning with convolutional neural networks has demonstrated learned agents capable to outperform human experts in Atari games

using only inputs coming from raw image pixels. Thus showing a path to Deep Reinforcement Learning, Mnih *et al.* paved the way to handle environments with more complex states spaces and more complex dynamics. Approaches called Policy Gradients methods [168] only improve their policy using directly the reward signal. These approaches do not explicitly approximate the Value function or the Q function. Such methods were also improved vastly by using deep neural networks [142, 143].

Another family of methods is Actor Critic methods. Actor Critics methods [82] combine policy gradient and value function approximations to enhance furthermore the final agents performance. Deep Reinforcement Learning Actor Critic methods like A3C [111] introduces a better training efficiency and stability to actor-critic learning through asynchronous agents being trained in parallel. The agents policy and critic network parameters are then synchronized periodically, allowing training to remain stable. Guided by these techniques, Reinforcement Learning has achieved remarkable success in multiple areas [81, 118, 173], but applying these methods to complex robotics tasks in simulation or directly to real robotics systems engender various new problems [85, 12]. One of these problems for Reinforcement Learning is scenarios with sparse or deceptive rewards functions.

In scenarios with sparse rewards, such as robotic grasping [12] or game playing [147], where the environment provides feedback only rarely or sporadically, RL agents often struggle to learn meaningful policies. The lack of frequent rewards makes it difficult for the agent to discern which actions are beneficial and which are detrimental. This results in slow learning and high variance in the estimated value functions, as the agent has limited guidance to update its policy. Additionally, sparse rewards can lead to exploration difficulties, where the agent may not encounter rewarding states frequently enough to explore the entire state space effectively. Furthermore, deceptive reward scenarios present yet another significant challenge to RL agents. Contrary to sparse reward scenarios, in such cases the reward signal provided by the environment might mislead the agent, pushing it to pursue suboptimal strategies. For instance, in a maze with a dead-end very close to the actual objective, the agent could remain stuck inside this dead-end while actually never reaching the goal if the reward function is designed as a simple Euclidean distance. In such situations, the RL agent can get stuck in locally optimal policies that provide misleading feedback and hinder the discovery of globally optimal strategies.

Both sparse and deceptive reward scenarios often lead to the problem of exploration-exploitation trade-offs. In sparse reward settings, exploration is crucial for finding rewarding states, but RL agents might be hesitant to deviate from known strate-

gies due to the limited feedback. In deceptive reward scenarios, the agent might be discouraged from exploring alternative actions that could lead to better outcomes, as the deceptive reward misguides its learning process. To address these challenges, various techniques have been proposed within the RL community. In sparse reward scenarios, methods like curiosity-driven exploration [150, 18] and intrinsic motivation [25, 9] aim to incentivize exploration by introducing auxiliary tasks or objectives that encourage the agent to explore unfamiliar parts of the state space. These techniques can help alleviate the exploration problem and improve learning efficiency.

In deceptive reward scenarios, research is focused on designing reward functions that better align with the true objectives of the task. Techniques like reward shaping [106] attempt to provide more informative reward signals, thereby guiding the agent towards more desirable behavior. Reward shaping consists in providing additional information that would help the agent fulfill its task. The drawback of doing so is that it can require a great quantity of expert knowledge on the task to be able to design a reward function that will have the right shape for the RL algorithm to solve properly the given task.

To conclude, while Reinforcement Learning is a powerful paradigm for solving complex decision-making problems, it faces significant challenges in sparse reward and deceptive reward scenarios. These challenges can reduce learning efficiency and lead to suboptimal policies. Addressing these issues requires the development of novel algorithms and techniques that enhance exploration capabilities of RL algorithms or to adjust reward signals through cumbersome reward shaping.

2.2 Evolutionary Algorithms

Other optimization techniques exists to tackle the same problems as Reinforcement Learning. Evolutionary Algorithms (EAs) [110] are a family of optimization and search techniques inspired by the principles of natural evolution [34]. They work by maintaining a population of candidate solutions or individuals and applying successively selection and variation to evolve the population over successive generations. In Evolutionary Algorithms, a candidate solution is evaluated over a complete episode whose length is the task horizon. A value, called fitness (and usually corresponding to the accumulation of reward seen over the whole episode, or somehow adjusted to give the correct selective pressure) is thus used to direct the search in the parameter space using the evolution theory mechanisms. Evolutionary Algorithms thus generally follow the evolution cycle depicted on Figure 2.2. These methods have been

widely applied in the robotics field [42].

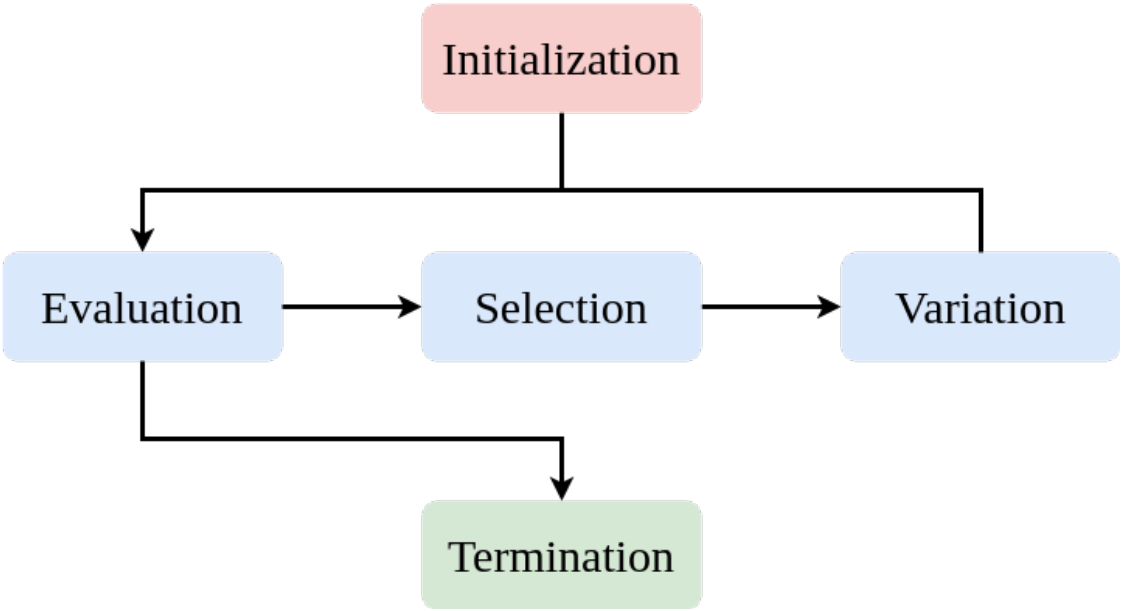


Figure 2.2: Evolutionary Algorithms general cycle

As described by Eiben *et al.* [46], firstly a population of individuals is initialized, each individual having its own set of parameters or **genotype** $\theta \in \Theta$. Secondly, each individual in the population is evaluated on a complete episode on the environment associated to the MDP of the task or any other task that returns a *fitness* for an individual evaluation. Depending on a **selection** criteria that will be developed in the next section, individuals might be selected among the population. Once an individual has been selected, its genotype will undergo **variation**. Several possibilities exists so as to variation, from mutations to breeding between two individuals. Possible variations will also be detailed in the next section. Once a new population of individuals has been created through selection and variation, it is once again evaluated and the process repeats itself until a termination criterion is reached, whether it be a task-oriented criterion or simply a budget-oriented criterion. Moreover, an individual performance is evaluated through what is called a *fitness function*. The fitness function has the same role as the reward function in RL, except that in Evolutionary Algorithms the fitness is evaluated over the whole episode rather than being a step-by-step reward.

2.2.1 Selection and Variation techniques

Selection is a component of Evolutionary Algorithms that plays a significant role in the effectiveness of the optimization process. The selection process determines which individuals from the current population will be chosen as parents for generating the next generation of individuals. The primary goals of selection in Evolutionary Algorithms are to preserve or ameliorate higher fitness individuals and to maintain diversity, thus striking balance between exploration and exploitation. Several techniques exist so as to how to select which individuals to variate or to carry over onto next generation. We list here the most commonly used ones:

- **Roulette Wheel Selection** is a method where individuals are selected proportional to their fitness. Higher fitness individuals have a higher chance of being selected. This method mimics the idea of "survival of the fittest" and can lead to faster convergence towards optimal solutions.
- **Tournament Selection** consists in firstly selecting a fixed number of individuals (the tournament size) randomly from the population, and then select the one from the tournament with the highest fitness. This process is repeated to select multiple individuals. This selection method allows for a good trade-off between exploration and exploitation.
- **Elitism** consists in selecting part of the best individuals from the current generation to be directly carried over to the next generation without any changes. This ensures that the best solutions are preserved across generations potentially decreasing convergence time.

On the other hand, variation plays a different but as important role in driving the evolutionary search. Variation encompasses the genetic operators of crossover and mutation, which introduce diversity and explore new solution regions in the population by directly modifying the individuals genotype:

- **Mutation**: Mutation involves making small random changes to an individual's genetic representation. This introduces exploration by creating new genetic material that might not have been present in the current population. Mutation can occur in various ways depending on the representation used (bit-flip mutation for binary strings and Gaussian mutation for continuous values for example).

- **Crossover**: Crossover involves combining genetic information from two or more parent individuals to create one or more offspring. In a binary representation, for example, a crossover point is chosen, and genetic material before and after the point is exchanged between parents to create offspring.

The variation process is fundamental in allowing EAs to not get stuck in local optima and converge towards high-quality solutions. Both mutation and crossover do not take into account the fitness of the individual and allow for gradient-free blind exploration of the individuals parameter space. Both processes are inherently stochastic as to what part of the genome should be modified or taken from the parents individual, and are completely agnostic to the problem considered. EAs can thus be applied to optimization problems formulated as a black-box problem [46], where the only need is to be able to provide a candidate individual to the problem and to evaluate its performance. Selection and variation thus creates the basis of any Evolutionary Algorithm.

Evolutionary Algorithms have exhibited remarkable efficacy in a diverse array of applications across various domains. In the realm of robotics, EAs have been leveraged for tasks such as robot path planning [62], evolutionary robot design [100], and swarm robotics optimization [17]. In the finance domain, EAs have been used in portfolio optimization [132] and trading strategy development [5]. Moreover, EAs have found utility in other fields, including data clustering [61] and medical image processing [87], highlighting their versatility and adaptability in addressing complex optimization challenges. But a sub-field of Evolutionary Algorithms, that we will call Diversity Algorithms throughout this thesis, is specifically interesting in many robotic tasks.

2.2.2 Diversity Algorithms

Firstly, as a sub-class of Evolutionary Algorithms, Diversity Algorithms do not aim to solely optimize for a specific fitness function or task but rather take into account or optimize for novelty in the obtained solutions with regards to the previously obtained solutions in what is called an **Outcome Space** or **Behavioral Space**. In addition to the previously defined Markov Decision Process tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, we thus now define:

- $\mathcal{B}$, the **Behavioral** or **Outcome space** in which the agent's behavior is described. Elements of this space are called behavior descriptors $b \in \mathcal{B}$. An

example of a behavior descriptor could be the final pose of a two-wheeled robot.

- $o_{\mathcal{B}}$, the **observer function** that associates to a trajectory of N consecutive states $\tau = \{s_0, s_1, \ldots, s_N\}$ a behavior descriptor $b \in \mathcal{B}$ that is usually of reduced dimensions compared to τ as it captures its essential parts:

$$o_{\mathcal{B}} : \mathcal{S}^N \longrightarrow \mathcal{B}$$

- $\mathcal{A}_{\Pi}$, the **Archive**, a set of all the individuals that are saved in the final diverse set of solutions found by the Diversity Algorithm, and that are used to compute the novelty of evaluated individuals.

Thus, the evaluation step of the Evolutionary Algorithms (Figure 2.2) now returns a behavior descriptor and a fitness as well. In the context of a MDP, at the end of an episode an individual is thus assigned a behavioral descriptor $b \in \mathcal{B}$ and a fitness $f \in \mathbb{R}$. Using the behavioral descriptor, Diversity Algorithms aim is to find a set of diverse individuals, which in our case are parameterized policies π_θ, that covers the Outcome space or Behavioral Space. Coverage is a metric that is measured by evaluating the ratio of the regions of the Behavioral Space that the Diversity Algorithm reached to the total reachable Behavioral Space. In this thesis, to measure coverage, the Behavioral Space is discretized along all its dimensions, and a bin is considered filled when a behavioral descriptor of an evaluated individual falls into this bin. Coverage is then measured by calculating the ratio of the number of filled bins to the total number of bins. Diversity Algorithms are particularly interesting in the case of sparse or deceptive reward scenarios, as they do not have as a first objective to maximize an extrinsic reward signal coming from the environment that can lead to suboptimal policies.

2.2.2.1 Novelty Search

The Novelty Search algorithm, introduced by Lehman and Stanley [91] as an alternative approach to classic evolutionary search, was the first Diversity Algorithm to be proposed. It indeed diverges from traditional fitness-based optimization methods by prioritizing the discovery of novel solutions rather than focusing on improving fitness at all. Novelty Search hypothesize that searching for novel solutions can unveil unexplored regions of the solution space, potentially leading to more diverse and innovative outcomes.

The Novelty Search algorithm introduces a new metric, usually called Novelty or Diversity, that is directly computed in the Behavioral space over the whole episode behavior of the individual. In mathematical terms, given a set of individuals $\{x_1, x_2, \ldots, x_n\}$, the novelty of the behavior descriptor b_{x_i} of a solution x_i is quantified by its Euclidean distance from its k-nearest neighbors in the population and in the archive. The novelty score $N(b_{x_i})$ can be calculated as:

$$N(b_{x_i}) = \frac{1}{k} \sum_{j=1}^{k} \text{distance}(b_{x_i}, b_{x_j}), \tag{2.3}$$

where $\text{distance}(b_{x_i}, b_{x_j})$ denotes the distance metric between the behavioral descriptors of solutions x_i and x_j.

Novelty Search thus encourages the exploration of the Outcome space, even if the novel solutions do not immediately exhibit higher fitness values. This creates the main difference between Evolutionary Algorithms and Diversity Algorithms as how good a solution is is not decided based on its fitness, but rather on the novelty of the discovered solution compared to the rest of the previously found solutions, measured in the Outcome Space. Doing so, it has been shown that the Novelty Search algorithm has covering properties over the user-defined Behavioral Space [43], making it a strong exploration algorithm for problems that can be considered as black-box as long as it is possible to define a Behavioral Space. Indeed, the Outcome space is usually designed so that it is aligned with the task to be solved by the agents, such that covering the Behavioral space covers the task space and generally allows to find at least one solution solving the given task in the task space. The aim of covering the Behavioral Space can also be to discover a variety of *skills* for an agent, that it could then use in a sequential way to solve more complex tasks.

Novelty Search iterative process involves generating a set of solutions and assessing their novelty to drive the population towards diversity. Individuals with higher novelty scores are retained as potential parents for the next generation, irrespective of their fitness levels. This decoupling of novelty and fitness ensures that the algorithm remains unbiased towards purely exploiting the best solutions, making Novelty Search a purely exploratory population-based policy search method. Moreover, at each generation a fixed number of individuals are added to the archive. Gomes *et al.* [53] shown on a set of robotic navigation tasks that choosing the individuals to add to the archive randomly rather than the most novel led to a higher exploration uniformity of the Behavioral Space than adding them based on their novelty, as well as reducing the sensitivity to the number of individuals added to the archive at each

generation. The archive is then involved in the novelty metric, such that the algorithm has a "memory" of the previously visited regions of the Outcome Space. The complete algorithm is outlined in Algorithm 1.

Algorithm 1 Novelty Search

s_p is the population size, s_o is the offspring size, k is the number of individuals to add to the archive at each generation, G is the generation budget
$\mathcal{A}_\Pi \leftarrow \emptyset$ ▷ *Initialize $\mathcal{A}_\Pi$ to an empty set*
$g \leftarrow 1$ ▷ *Set g at as the first generation*
$\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}} \leftarrow random\_parameters()$ ▷ *Set the population to randomly parameterized policies*
$b_{i \in [1,s_p]} \leftarrow \text{evaluate}(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Evaluate the randomly initialized policies*
$\eta_{i \in [1,s_p]} \leftarrow N(b_1), \dots, N(b_{s_p})$ ▷ *Compute the Novelty using the population behavioral descriptors*
$\mathcal{A}_\Pi \leftarrow sample(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Add k individuals from the population to the archive*
▷ *Performing Novelty Search*
for g *in* $[1, \dots, G]$ **do**
    $\pi_{\boldsymbol{\theta}^*_1}, \dots, \pi_{\boldsymbol{\theta}^*_{s_o}} \leftarrow variation(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Generate the offspring from the population using the variation operator*
    $b_{i \in [1,s_o]} \leftarrow \text{evaluate}(\pi_{\boldsymbol{\theta}^*_1}, \dots, \pi_{\boldsymbol{\theta}^*_{s_o}})$ ▷ *Evaluate the offspring*
    $\eta_{i \in [1,s_o]} \leftarrow N(1), \dots, N(s_o)$ ▷ *Compute the Novelty using the offspring behavioral descriptors*
    $\pi_{\boldsymbol{\theta}_1}, \dots, \pi_{\boldsymbol{\theta}_{s_p}} \leftarrow select(\pi_{\boldsymbol{\theta}^*_1} \dots \pi_{\boldsymbol{\theta}^*_{s_o}}, \eta_1 \dots \eta_{s_o})$ ▷ *Update population with the s_p most novel individuals in the offspring*
    $\mathcal{A}_\Pi \leftarrow sample(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Select k individuals from the population to be added to the archive*
    $g \leftarrow g + 1$
return $\mathcal{A}_\Pi$

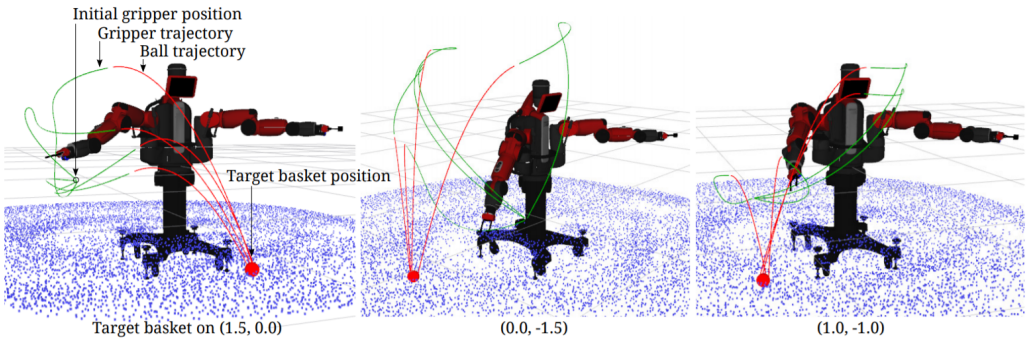


Figure 2.3: Example of diverse trajectories found by QD search for different goal coordinates [78]

Novelty Search has found various applications in robotics, the first one being in the original NS paper on a two-wheeled robot maze navigation task [91]. NS thus shown its ability to find policy representations and parametrizations that would successfully complete any of the given mazes, even the most deceptive ones. In a higher degrees-of-freedom robotic task, Kim *et al.* [78], as shown on 2.3, gave a robot the ability to throw a ball in a basket and adapt to failure thanks to a large and

diverse coverage of the defined behaviour space. Exploring directly the Outcome Space thus allows to tackle seamlessly robotic tasks of various degrees of freedom and dynamics, only by defining a proper Behavioral Space aligned with the goal task. Novelty Search has been the first Diversity Algorithm to be introduced, but paved the way for other algorithms, notably Quality Diversity Algorithms.

2.2.2.2 Quality Diversity

Quality-Diversity (QD) algorithms [133] aim to achieve a balance between optimizing the quality of solutions and promoting diversity among them. Different QD algorithms exists, but their objective is usually to maintain a diverse set of solutions that cover the Outcome Space as best as possible while keeping the highest performing solutions in each region of the Behavioral Space. Indeed, ignoring completely the reward in Novelty Search grants great exploratory capabilities but prevents the algorithm to exploit any reward signal found in the environment. Quality Diversity algorithms try to overcome this weakness of Novelty Search by creating competition between individuals whose Behavioral descriptors are close in the Outcome Space.

One main distinction between the different Quality Diversity algorithms is how the archive is organized. In the first Quality Diversity Algorithm, Novelty Search with Local Competition, Lehman And Stanley [92] propose to add a niching mechanism to the Novelty Search algorithm, such that once two solutions fall into the same niche, the highest performing one is the one that is kept in the Archive. The niching mechanism works by considering two policies close enough in the Outcome Space to be in competition for being saved in the niche they share. The authors then use a multi-objective optimization algorithm inspired from NSGA-II [36] to promote both local competition between individuals in neighboring niches and novelty, which allows them to cover as much the Outcome Space as Novelty Search while having individuals in each niche that outperforms the ones found with a simple Novelty Search on an Evolution of Robotic Organisms task.

Since, different niching mechanisms have been proposed. As much as NS-LC has outperformed NS on certain tasks, it remains computationally expensive, having to compute several metrics and optimize multiple objectives at the same time as the Outcome Space remains continuous. Addressing that issue, the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites, [121]) algorithm proposes to discretize the Behavioral Space, thus creating a fixed number of niches beforehand which have a look-up time lower than the one of a costly k-nearest-neighor computation in the

NS-LC algorithm. One disadvantage of MAP-Elites is that it does not scale to high-dimensionnal Outcome Space as the number of niches grow exponentially with its dimensions. Vassiliades *et al.* thus proposed to use a CVT [45] to create a fixed number of niches, whatever the dimension of the search space is. The complete MAP-Elites algorithm is detailed in Algorithm 2.

Algorithm 2 MAP-Elites

s_p is the population size, G is the generation budget
$\mathcal{A}_\Pi \leftarrow \emptyset$ ▷ *Initialize $\mathcal{A}_\Pi$ to an empty set*
$g \leftarrow 1$ ▷ *Set g at as the first generation*
$\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}} \leftarrow random\_parameters()$ ▷ *Set the population to randomly parameterized policies*
$b_{i \in [1,s_p]} \leftarrow \text{evaluate}(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Evaluate the randomly initialized policies*
▷ *Performing MAP-Elites*
for g *in* $[1, \dots, G]$ **do**
    $\pi_{\boldsymbol{\theta}^*_1}, \dots, \pi_{\boldsymbol{\theta}^*_{s_p}} \leftarrow variation(\mathcal{A}_\Pi)$ ▷ *Generate a new population from $\mathcal{A}_\Pi$ using the variation operator*
    $b_{i \in [1,s_p]} \leftarrow \text{evaluate}(\pi_{\boldsymbol{\theta}^*_1}, \dots, \pi_{\boldsymbol{\theta}^*_{s_p}})$ ▷ *Evaluate the population*
    for i *in* $[1, s_p]$ **do**
        if $\mathcal{A}_\Pi(b_i)$ is empty **then**
            $\mathcal{A}_\Pi(b_i) \leftarrow \pi_{\boldsymbol{\theta}_i}$
        else
            if $fitness(\pi_{\boldsymbol{\theta}_i}) > fitness(\mathcal{A}_\Pi(b_i))$ **then**
                $\mathcal{A}_\Pi(b_i) \leftarrow \pi_{\boldsymbol{\theta}_i}$
    $g \leftarrow g + 1$
return $\mathcal{A}_\Pi$

Quality Diversity methods were applied on locomotion tasks in order to give hexapod robots high adaptability even in case of mechanical malfunction, through maintenance of a highly diverse behaviour repertoire [31] [21]. These methods were also applied to provide rapid generalization capabilities to unforeseen scenarios to robotic systems when combined with higher-level planning capabilities [76]. Quality Diversity algorithms, exemplified by the MAP-Elites approach, thus provide a novel perspective on behavior optimization by also promoting the discovery of diverse solutions. These approaches are promising in problems where traditional purely fitness-based optimization methods may fail due to pitfalls in the fitness function. Nevertheless, Diversity Algorithms, as well as all the data-driven techniques that were detailed up till now, suffer from several drawbacks.

2.3 Closing the Reality Gap

While Reinforcement Learning allows for policy optimization and generalization for a specific task, it may struggle to converge when encountering problems with a

deceptive or sparse reward function. On the other hand, Diversity algorithms do not suffer from such problems, at the cost of even more data samples taken from the environment. When working with fast simulators, having an important data usage, or a low *sample efficiency*, might not matter as sampling transitions from the simulator is low cost in itself. But when considering real robotic applications, sampling transitions from the environment suddenly becomes very costly: it takes a significantly increased amount of time, it increases the real robot wear and tear, they might not be available for an extended period of time and require safety measures. Using such data-driven techniques directly on a real robot thus does not make much sense or requires an extraordinary amount of real-life resources to be applied properly like Levine *et al.* did [96] with 14 manipulator arms in parallel for data collection over the span of two months.

Nevertheless, the ultimate goal of robot learning approaches remains to have these policies perform on a real robot in the real world, not in simulation. To do so, one possible way is to leverage knowledge from a simulator to train a policy or learn an archive of policies that would then be *transferred* onto the real robot. But for the previously described techniques to transfer directly on real robots, it is a requirement that the policies are learned on an accurate enough simulator [80, 107], such that the discrepancies between the simulation and the real robotic setup are not too large. But when these differences are hard to overcome and prevent direct behavior transfer from one domain to another, an issue called the reality gap [66] or the simulation bias [6, 8] appears. Both terms designate the same problem, but we will refer to the aforementioned issue as the reality gap, this word being more commonly used in the literature [120].

The reality gap arises because the robot's behavior learned in simulation may not translate perfectly to the real-world robotic setup. This disparity can result in unexpected and sometimes catastrophic failures when deploying the robot controlled by the learned policy in real-world scenarios. Several factors can contribute to the reality gap:

- **Physics and Dynamics**: Simulations are simplifications of the real world, and as such, they often lack the fidelity required to accurately represent the physics and dynamics of real-world scenarios. In simulations, friction, air resistance, and other forces may be neglected or modeled at a coarse level, resulting in different robot behaviors compared to the real world.

- **Sensor and Actuator Models**: Sensors and actuators in simulations might

not perfectly replicate their real-world counterparts. Calibration errors, noise, and discrepancies between simulation and real-world sensor data can lead to different robot perception inducing a different control, affecting the robot's ability to interact as intended with its environment.

- **Environment Modeling**: Simulations rely on environment models that are designed to be computationally efficient and easy to simulate. However, these models often fail to capture the complexity and dynamics of real-world environments. Real-world scenarios include a myriad of unpredictable elements, such as varying terrain, lighting conditions, and dynamic obstacles, which are challenging to replicate accurately in simulations.

- **Adaptability**: Robots trained solely in simulation may lack the adaptability required to handle variations and uncertainties present in real-world conditions. They may overfit to the specific conditions of the simulation, rendering them ineffective or unsafe in unforeseen situations.

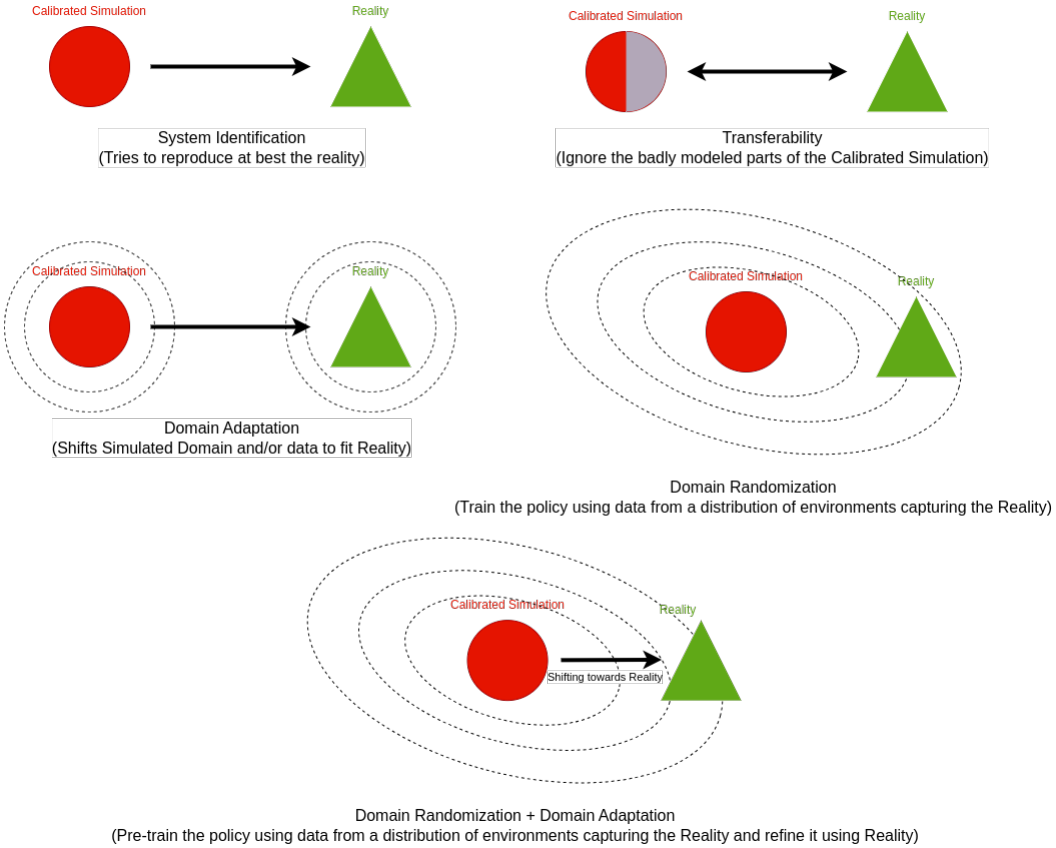


Figure 2.4: Different simulation-to-reality techniques [167]

Four of the main approaches proposed over the years to close the reality gap

are **System Identification**, **Domain Adaptation**, **Transferability** and **Domain Randomization** as shown of figure 2.4. These techniques are also called simulation-to-reality approaches. The amount of samples required for the various Policy Search techniques presented in this thesis are shown on figure 2.5. We take as a basis for harmonizing the amount of samples between the various techniques an environment with an episode length of 300 time-steps. System Identification objective is literally to build the best mathematical model for the real physical system that needs to be tackled. In the case of policy search in robotics, the mathematical model is thus the simulator, and System Identification consists in estimating at best the simulation parameters, either beforehand or using some interactions with the real world. Robotics Simulators [30, 157, 28] are the prime example of System Identification, as they are based on the various physics laws that determine the dynamics of robotics systems. Nevertheless, these simulators usually rely on fixed sets of parameters during the policy search, which can lead to the reality gap problem as mentioned previously as modelisation errors exist [47, 3]. Thus, some authors proposed approaches that automatically identify the mathematical models parameters using data from real-world interactions [1, 83].

Nevertheless, such approaches might struggle with the complexity and variability of real-world environments as they require accurate measurements and assumptions about the system's underlying physics, which can be difficult to provide, especially in dynamic and uncertain settings. Domain Adaptation techniques [69, 48, 101], on the other hand tackle this problem differently. Domain Adaptation techniques do not necessarily assume a fixed mathematical model, but rather learn a policy in simulation and refine it in the real-world setup or learn a mapping between the source and target domain that is used to bring closer the source domain data distribution (from which sampling is cheap) to the target domain data distribution. In the first case, the idea is thus to pre-train the policy by making the best usage of the existing similarities between the simulation and real-world scenario, so that the least number of interactions with the real system can be done afterwards to learn an efficient enough policy regarding the task to solve. The second case can be applied in several ways, like proposed by Chatzilygeroudis *et al.* [21] and generalized to multiple contexts by Kaushik *et al.* [76] in the Evolutionary techniques domain where authors propose to learn a corrected mapping between the policies in the archive and their outcome in the Behavioral Space. In the Reinforcement Learning context, authors proposed to reduce the gap between the simulated observations and the real observations, either learning to render more realistically the simulated observations [16] or by learning to represent any observations as a canonical view [67], effectively reducing the gap

in performance between simulation and reality as the policy is fed the same observations. Finally, Domain Adaption can also be done through adversarial training, where a discriminator is used to align the policy's output distribution in both simulation and real-world domains [159]. This adversarial setup encourages the agent to produce similar behaviors in both domains, effectively reducing the reality gap. Domain Adaptation techniques can use as low as 2000 real-world samples (steps on the target domain) but up to 100 000 depending on the complexity of the task and the method being used.

Domain Randomization techniques propose another path to attack the reality gap problem. In a way that reminds Domain Randomization, Jakobi *et al.* [66] proposed to fine-tune the simulation noise a bit more than the real-system noise such that the learned behaviors would be more robust to the real world dynamics that the simulation will anyhow fail to predict completely. Pursuing on the same track, authors proposed a Domain Randomization [156], which consists in learning the robot policies in the simulator while wrapping it in an envelope of noise, much like Jakobi *et al.* did in 1995. In Domain Randomization, the agent is trained in a simulated environment with a wide range of variations to create a more robust policy that can adapt to different real-world conditions, such that behaviors are more robust to transfer. It requires expert knowledge on both the source and target domains, as the programmer needs to know which parameters from the simulation are not well modeled or can vary slightly in the reality, like light conditions, friction or wear and tear in the robot mechanical system. This approach has been shown to improve the generalization of robotic control policies and perception systems, aiding in bridging the reality gap between simulation and reality[156, 129, 138]. Finally, Mehta *et al.* [109] propose an active way of doing Domain Randomization quite opposed to simply fine-tuning by hand the environments distribution parameters. They propose to refine the current simulated environment distribution parameters by selecting the most informative MDPs for the agent out of the current distribution. This allows them to enhance the zero-shot performance of Domain Randomization by 1.5-2 times on a task with a robotic arm pushing an object. Domain Randomization techniques are zero-shot transfer techniques, meaning that the policies are trained solely in the source domain.

Domain Randomization and Domain Adaptation techniques have also been combined together to allow real robots to accomplish tasks with higher accuracy and using a small number of real-world samples [174, 22, 67, 109, 138]. Yu *et al.* proposed to firstly search in randomized simulations for an ensemble of highly rewarding policies and then to select the policy among that ensemble that performs best in the

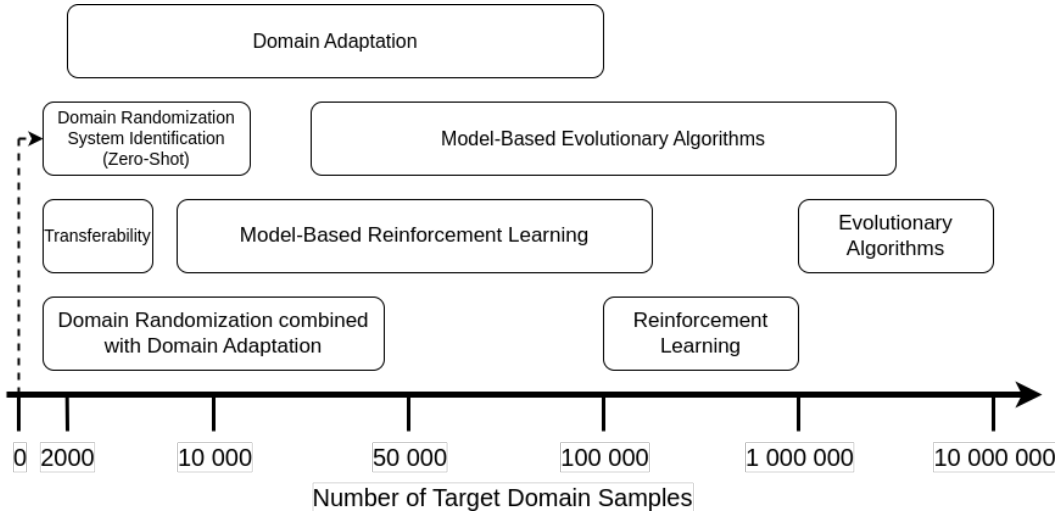


Figure 2.5: Graph of the number of samples required by each family of Policy Search algorithms and methods aiming at crossing the Reality Gap. Model-Based approaches are presented in the subsequent section.

target environment [174]. This reminds the works of Kim *et al.* [79] which used the Novelty Search algorithm (to generate an ensemble of policies) and local adaptation to cross the reality gap in a ball-throwing task on a real Baxter robot. Chebotar *et al.* propose to use some real-world samples to learn the parameters of the randomized environments distribution, so as to have a distribution of environments that captures as best as possible the real environment dynamics, which allows their real-world tasks (opening a drawer and swinging a ball in a hole) to be solved using only a few episodes [22]. The general idea of such hybrid techniques is that the distribution of data generated from the source environment is being brought closer to the target domain data distribution, while using means of domain randomization to capture as best as possible the real system data distribution as shown in Figure 2.4. Hybrid techniques combining Domain Randomization and Domain Adaptation allow the number of target domain samples required to be diminished compared to Domain Adaptation alone, and allows to tackle more complex tasks than Domain Randomization alone. Samples used typically ranges from 1000 to 50 000.

The Transferability approach is a little more radical compared to the other simulation-to-reality approaches. Indeed, the Transferability [84] approach chooses to completely ignore the behaviors found on the badly modeled parts of the simulator, by iteratively estimating the existing discrepancies between the simulator and the real-world. Even if this technique indeed overcomes the reality gap problem, it does so at the cost of potentially missing the most rewarding behaviors if the simulator cannot model these properly, as highlighted by some parts of the simulator

being greyed out as shown on figure 2.4. This once again highlights the fact that if the discrepancies existing between the simulation and the real world are too great, a critical problem arises. Indeed, if so, policies found using a simulator will most likely fail even using simulation-to-reality techniques as the simulator cannot contain enough meaningful information at all. Tasks with high system dynamics complexity, like manipulation of deformable objects [12], are particularly subject to this kind of problem as no existing simulator is fast and accurate enough to learn robust policies using an important amount of simulated samples. Ensuring that the learned policies generalize well to unseen real-world scenarios while avoiding negative transfer (where performance degrades due to mismatched domains) is thus an ongoing research focus, and the choice of simulation fidelity and how well it aligns with the real-world conditions significantly impact transfer success. The Transferability approach is a very cost-efficient technique even on locomotion tasks with high degrees-of-freedom dynamics. Indeed, Transferability was generally evaluated using a total of only 1000 to 4500 real-world samples, nevertheless at the cost of potentially missing some of the most rewarding behaviors.

2.4 Model-based Approaches

Few possibilities remain as to how alleviate the reality gap problem. One way to go around it, as no simulator is used, is model-based techniques [131]. In such approaches, a model of the system is directly learned from data gathered on the target domain. Model-based policy search approaches have been explored by the literature for decades [7], and have shown promising results in effectively giving real robotic systems the ability to perform simple [14, 37] to more complex tasks [26, 122] without needing an extreme amount of data. Even though model-based techniques suffer from their own drawbacks linked to the uncertainty within the model of the environment they are trying to learn, they have proven to be an interesting alternative or complement to other policy search methods relying exclusively on a simulator. A lot of model-based policy search techniques are step-based policy search methods, resulting from the reinforcement learning framework [151]. On the other hand, episode-based policy search methods are less represented in the robotics field and principally stem from Bayesian Optimization [148] and from Evolutionary methods [91].

Going more into detail, the transition function can be written as $s_{t+1} = T(s_t, a_t)$. When learning the dynamics model of the system, we are actually approximating the

true system's transition function T with a function $\hat{T}_\phi$ parameterized by a vector ϕ that can be represented using several different type of learnable models. Model-based policy search aim is thus to fit a model $\hat{T}_\phi$ using limited measurements of the true transition function T in the form of N data samples $\mathcal{D} = \{(s_n, a_n), s_{n+1}\}_{n=1}^{N}$. $\hat{T}$ is used recursively to predict a policy's behavior in imagination over a given horizon without trying it out on the real system, which helps to save expensive real system samples. Deisenroth *et al.* [39] showed that using tuples $(s_t, a_t) \in \mathbb{R}^{d_s+d_a}$ as training inputs and differences $\Delta_t = s_{t+1} - s_t \in \mathbb{R}^{d_s}$ as training targets facilitates vastly the training of a dynamics model. Indeed, doing so helps in several ways. Firstly, doing so makes the assumption of environment stationarity: In many real-world scenarios, the dynamics of the environment are relatively consistent and stationary over time. By learning the differences between consecutive states, the model can focus on capturing these changes, which tend to be smaller and more stable than the absolute values of states. Secondly, there is a regularization effect, as learning state residuals naturally regularizes the model. It encourages the model to focus on capturing the incremental changes in the environment, which often have more structure and regularity than absolute states which increases model stability throughout learning. Finally, it greatly helps toward model generalization as state deltas often exhibit more consistent and predictable patterns than absolute states. This can lead to better generalization when the model needs to handle unseen or novel states that were not explicitly encountered during training. Dynamics Models can be used in Various Model-Based planning schemes and Model-Based Policy Search algorithms using a Dynamics Model can be more or less divided into three categories: Model Predictive Control, Model-Based Reinforcement Learning and Model-Based Evolutionary Algorithms.

2.4.1 Model-Based Planning Schemes

2.4.1.1 Model Predictive Control

Model Predictive control (MPC) [19] is an advanced control technique for automation. The principle of this technique is to use a dynamic model of the process inside the controller in real time in order to anticipate the future behaviour of the process. The dynamics model can either be a mathematical model or a learned model. This technique is particularly interesting when systems have large delays or many disturbances. Model Predictive Control is widely employed in diverse applications such as robotics, process control [135], and autonomous vehicles [73]. It operates by solving

an optimization problem over a finite prediction horizon to determine a sequence of control actions that optimally achieve desired objectives while satisfying system constraints.

The fundamental principle of MPC involves formulating an optimization problem that considers a dynamic model of the system, an objective function, and constraints. Let $\mathbf{s}_t$ represent the system state at time step t, and $\mathbf{a}_t$ denote the actions taken. The optimization problem is defined as follows:

$$\begin{aligned} \min_{\mathbf{a}_{t:t+H}} & \sum_{k=0}^{H-1} C(\mathbf{s}_{t+k}, \mathbf{a}_{t+k}) + C_f(\mathbf{s}_{t+H}) \\ \text{subject to} \quad & \mathbf{s}_{t+k+1} = \hat{T}(\mathbf{s}_{t+k}, \mathbf{a}_{t+k}), \\ & \mathbf{s}_{t+k} \in \mathcal{S}, \\ & \mathbf{a}_{t+k} \in \mathcal{A}, \end{aligned} \tag{2.4}$$

where H is the prediction horizon, C represents the immediate cost associated to taking action a_t in state s_t at time step t and C_f is the cost associated to the final state. The costs functions C and C_f are usually defined as the opposites of the reward function R, as the reward should be maximized but the cost minimized.

The optimization problem is solved at each time step, producing a sequence of optimal control inputs $\mathbf{a}_{t:t+H}$ as shown on Figure 2.6. However, only the first control action $\mathbf{a}_t$ is applied to the system. As time progresses, the MPC optimization is re-executed with a receding horizon, adapting to changes in the environment and system dynamics. MPC's ability to handle constraints, uncertainties, and long-term planning has made it an essential tool for complex control problems.

Even though the MPC paradigm stems from the control community [19] where it has been applied even to very complex tasks like robot surgical interventions [161], it has also more recently been applied to different robotics setups with lot less hypothesis using learned dynamical models. Doing so, Nagabandi *et al.* [122] demonstrated that learning a dynamics model with a medium-sized neural network is sufficient to learn complex locomotion gaits when coupled with model predictive control. Taking this one step further, PETS [26] proposed a method in which they learned online a dynamics model on several reinforcement learning and robotics benchmark tasks using deep ensembles. They proposed using ensembles of probabilistic neural networks to capture both epistemic and aleatoric uncertainty [64], epistemic being the uncertainty that can be reduced through acquiring more data, thus useful to orient

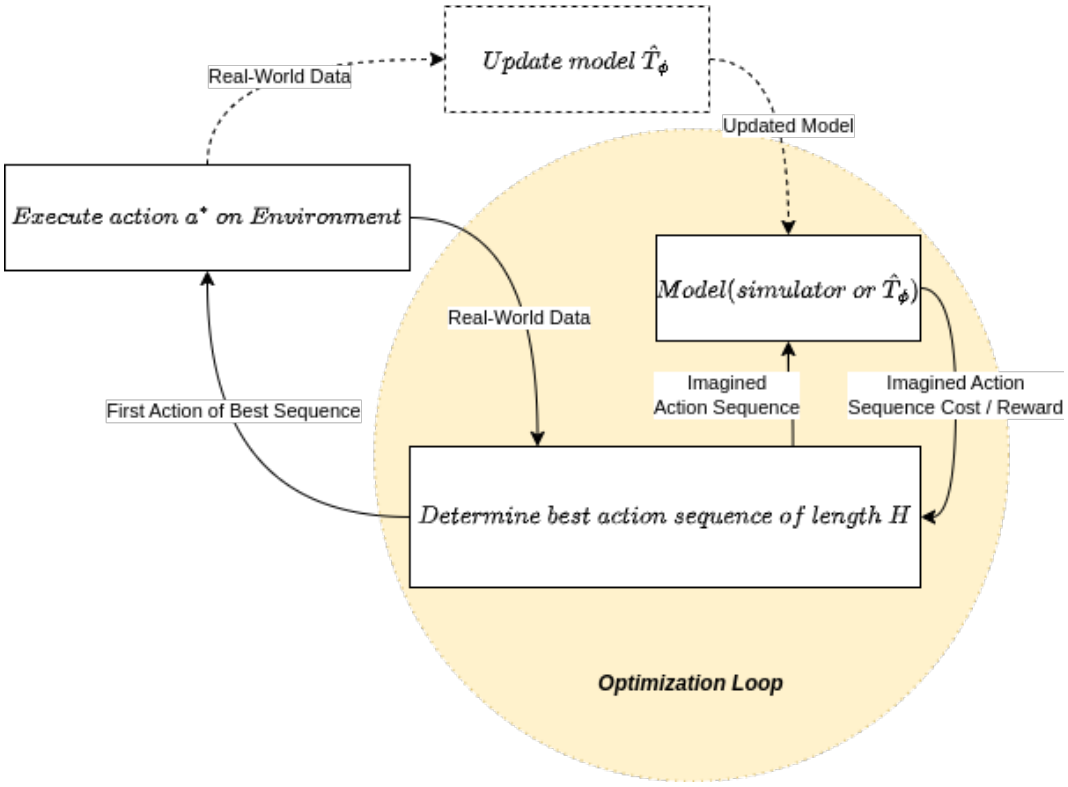


Figure 2.6: The Model Predictive Control Loop

exploration [155], and aleatoric being the one inherent to the underlying stochastic process. Moreover, other techniques like Dynamics Aware Discovery of Skills [145], Plan2Explore [144] and PDDM [123], all leverage learnt dynamics model to enhance sample-efficiency and make more informative decisions in the agent environment, using MPC either to select the best learned skill to use or to apply the best estimated action at each timestep.

2.4.1.2 Model-Based Reinforcement Learning

Combining Reinforcement Learning with a learned model, Model-Based Renforcement Learning uses the learned model to optimize the agent's policy π. Indeed, if a cost function is available as in MPC, being able to model the world also allows to update the policy like a regular Reinforcement Learning algorithm, except that the data used to update the policy does not come from the real-world but is generated using the learned model $\hat{T}_\Phi(s, a)$ by minimizing the cost function. This way, the policy is executed on the model, new data samples in the form of (s, a, s') tuples are obtained and the associated cost $C(s, a) + C_f(s')$ is computed and used to train the

policy π. The goal is thus to update the policy π so that it's closer to optimality on the model $\hat{T}_\Phi$, by backpropagating the cost using the learned dynamics model to train the policy over a finite horizon T. This process is detailed in Figure 2.7.

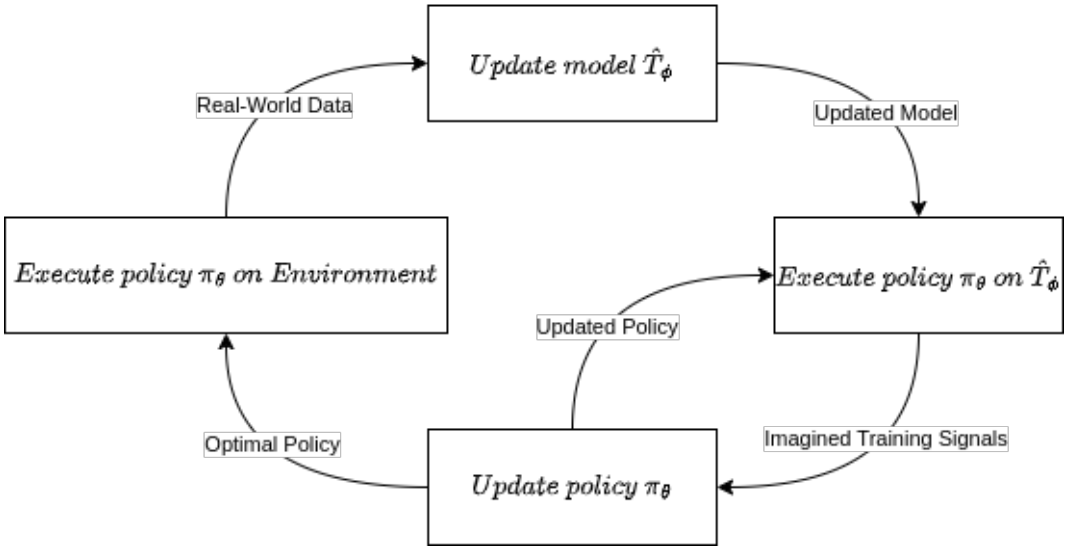


Figure 2.7: The Model-Based Reinforcement Learning Loop

Model-based Reinforcement Learning approaches have been explored by the literature for decades [7], and have shown promising results in effectively giving real robotic systems the ability to perform simple [37] to more complex tasks without needing an extreme amount of data. Doing so, [141, 37] control an actuated cart-pole to either maintain the pole up while moving the cart or to swing it up from a resting pose. Similarly, [14] demonstrate how a two-link robot arm with a single actuator can learn to swing up its non actuated endpoint above a target height. These works demonstrated interesting results on limited degrees-of-freedom robotic setups, opening the path to more complex control or policy search techniques using a model. Such techniques thus evolved towards more complex robotics systems, in terms of system dynamics and problem dimensionality. For example, several works have shown results on biped walking tasks. Like that, [116, 38, 95] all demonstrated the learning of successful walking gaits by bipedal robots, both on simulation [95] and on real robot systems [116, 38]. Kaiser *et al.* [75] also showcased SimPLe, an MBRL algorithm up to 10 times more sample-efficient than model-free algorithms while using only image inputs, much like [144] in the MPC framework. Finally, some methods demonstrates a robot learning to perform surgical tasks bootstrapped from demonstrations or in a pouring task for the PR2 robot [172], showcasing the ability of model-based techniques to learn even high dimensional behaviors in a few real-world interactions if a model can be learned quickly.

2.4.1.3 Model-Based Evolutionary Algorithms

A learned model can be integrated in different ways [23]. Indeed learned models have been coupled with evolutionary algorithms for quite a while [89]. Larragana *et al.* proposed to use the model to estimate the probability distribution of candidate solutions and use the model to generate a new population instead of using variation operators. The new population is then evaluated, and the model is trained iteratively with the newly collected data. Instead of estimating directly the distribution of candidate solutions, other methods try to directly learn the inverse model of the mapping between the objective space (the image of the decision space, in our case $\mathcal{B}$ the behavioral space) and the decision space (the space of the solutions, in our case the parameter space Θ) [52, 24, 29, 97, 153].

Finally, other methods directly try to learn a model of the objective function. This can be done in several ways depending on the nature of the problem to solve. The objective function can be estimated directly by mapping the decision space to the objective space [169, 71, 70, 152, 27, 49, 77], thus reducing the number of expensive evaluation of the objective function (in our case the transition function T over a complete episode). An other way applicable in the MDP case is to estimate directly the transition function, and use it to estimate the fitness and behavioral descriptors of the candidate solutions [98, 99].

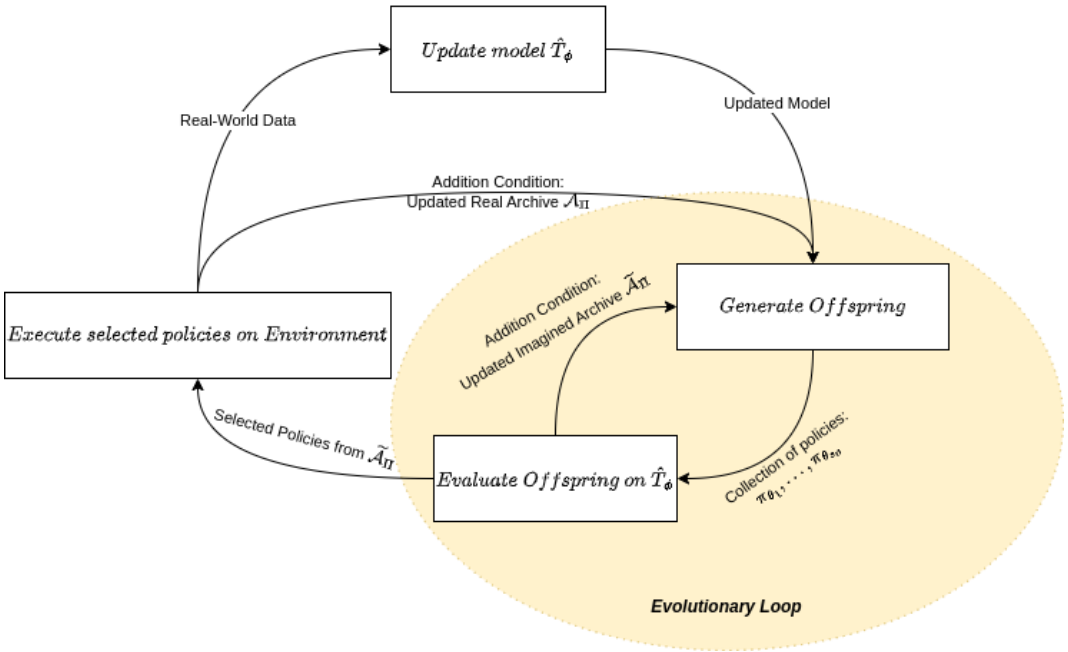


Figure 2.8: The Model-Based Evolutionary Algorithm Loop

Thus, Model-Based Evolutionary Algorithms, just like regular Evolutionary Algorithms, do not aim to optimize a single policy like Model Based Reinforcement Learning or a single action like Model Predictive Control but rather aim to generate a set of individuals that will solve the given optimization problem, whether it is to maximize a reward oriented fitness, to be as diverse as possible or multiple objectives at the same time. MBEAs work much like regular EAs, except that the optimization part is usually mostly done directly on the model $\hat{T}_\Phi$ as shown on Figure 2.8 for Novelty and Quality-Diversity based MBEAs (in this case the model $\hat{T}_\Phi$ can be a dynamics model [98, 99] or a direct model [49, 77])

In the robotics community, MBEAs have been applied only in a few recent occurences. Keller *et al.* thus uses a neural network to directly learn the mapping between actions and behaviours [77], which, depending on the complexity of the underlying dynamics, can be proven quite hard.

Prior to the works of this thesis, only Lim *et al.* [98, 99] coupled a learned Dynamics Model with an Evolutionary Algorithm. Authors showed on a locomotion task that the usage of a learned dynamics model coupled with the Quality Diversity algorithm MAP-Elites could reduce by 10 the number of samples required to reach a similar quality and coverage in the final archive of behaviors. This work is particularly interesting as it is the first to showcase that even in episode-based policy search techniques, where the planning horizon is usually pretty long, that a dynamics model can still hold meaningful information about the individual evaluated in imagination.

2.4.2 Model Representations

2.4.2.1 Gaussian Processes

Gaussian Processes (GPs) have emerged as a powerful tool for surrogate modeling. Gaussian Processes are the generalization of a Gaussian distribution over a finite vector space to a function space of infinite dimension [134]. These probabilistic models provide a flexible framework for approximating complex functions and learning transition dynamics. Given a set of observed state-action pairs and their corresponding next-state outcomes, a GP models the underlying transition dynamics and captures the uncertainty associated with the function approximation. The GP defines a distribution over functions and provides not only point estimates but also an uncertainty metric, which is interesting for making informed decisions when using an approximated model.

Mathematically, a GP is defined by a mean function $\mu(\mathbf{x})$ and a covariance function (kernel) $k(\mathbf{x}, \mathbf{x}')$, where $\mathbf{x}$ and $\mathbf{x}'$ represent the input points. Given a set of training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}$, where $\mathbf{y}_i$ is the observed next-state outcome corresponding to input $\mathbf{x}_i$, the predictive distribution at a test point $\mathbf{x}_*$ is a Gaussian distribution characterized by the mean and variance:

$$\mu_*(\mathbf{x}_*) = \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1} \mathbf{y}$$
$$\sigma_*^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1} \mathbf{k}_*$$

where k_* represents the vector of covariance between the test point and training points, K is the covariance matrix of the training points, and σ_n^2 is the noise variance parameter. The mean and variance provide information about the expected next-state outcome and the uncertainty associated with it.

The advantages of GPs for surrogate modeling lie in their ability to handle complex and nonlinear transition functions while providing a measure of uncertainty. This makes them well-suited for guiding exploration and decision-making in policy search problems, as they allow agents to make informed choices while accounting for the uncertainty inherent to the transition function approximation. Additionally, GPs can naturally handle unevenly spaced or noisy data, which is often the case in real-world applications. Gaussian Processes are thus a powerful tool, as it can leverage priors, exploit uncertainty in the model and generalize well with small data sets.

Another advantage of GPs is that they can leverage priors to actually learn quicker on the given task. Priors can come from previous tasks or from simulation. Several robotic applications made such use of GPs, such as ITE [31] which allowed an hexapod robot to adapt to different damage conditions in a few amounts of trials after facing this new situation. GPs were also used successfully in other robotic applications. For example, Martinez-Cantin *et al.* [105] used GPs to model a process in which a robotic system had to navigate in an uncertain environment. It was also used in other locomotion setups, for example to learn policies for a quadruped robot [102] or for a small vibrating soft tensegrity robot [136].

Nevertheless, Gaussian Processes suffer from several aspects. Firstly, they do require a prior assumption on the underlying data distribution, which can be detrimental when no knowledge of the task is available at all, as it requires to make

assumptions on the underlying function to approximate. Moreover, Gaussian Processes do not scale well to high dimensional data and big training data sets, as the computational power required for inference becomes quickly intractable [63]. This is particularly problematic in certain robotic tasks where the state and action space can grow pretty quickly

2.4.2.2 Deterministic Neural Networks

Just like GPs, Neural Networks (NNs) can be used to learn various models representing the task or its underlying dynamics. Standard NNs have been less used as models as they are deterministic and lack of the uncertainty information GPs have which allows better long-term predictions in terms of robustness as they tend to less be exposed to model bias. Nevertheless some researcher did use NNs for model learning in robotic applications. For example, Nagabandi *et al.* used NNs as dynamics model to initialize a model-free learner which was then used to learn walking gaits [122]. In the Evolutionary Algorithm framework, Keller *et al.* used a neural network to directly learn the mapping between actions and outcomes [77], which, depending on the complexity of the underlying dynamics, can be proven quite hard. Indeed, the mapping between actions and outcomes can be arbitrarly complex depending on the length of the task episode and the system dynamics complexity.

Nevertheless, Simple Neural Networks are still interesting to look at as they lay out the basis of the regression problem that needs to be solved to approximate the system dynamics. The goal is to minimize the discrepancy between the predicted next-state outcomes and the actual observed outcomes. The Mean Squared Error (MSE) is a commonly used loss function in this context:

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{y}_i - \hat{T}_\phi(\mathbf{s}_i, \mathbf{a}_i) \right\|^2,$$

where ϕ represents the parameters of the neural network, $\mathbf{s}_i$ and $\mathbf{a}_i$ are the state and action inputs of the i-th observation, $\mathbf{y}_i$ is the corresponding true next-state outcome, and N is the number of observations.

To address the challenge of generalization to unseen states and actions, regularization techniques like weight decay [86, 103] or dropout [149] can be employed. Weight decay adds a penalty term to the loss function based on the magnitudes of the network's weights, promoting smaller weight values and preventing overfitting.

Dropout randomly drops a fraction of the neurons during training, forcing the network to rely on a variety of features and thus enhancing its ability to generalize. This way, DEEP-PILCO [51] modified the previous PILCO algorithm by replacing the GP by a Neural Network, whose output uncertainty is captured using dropout [149], allowing the algorithm to handle bigger training datasets both in terms of dimensionality and training samples.

Despite their capabilities, NNs for learning transition functions have challenges, such as the curse of dimensionality and overfitting. Training on insufficient or biased data may lead to poor generalization and inaccurate transition models. Furthermore, the choice of architecture, regularization, and optimization strategies can significantly impact the performance of the learned transition function. Inspired by these drawbacks, other Neural Networks based approaches were proposed.

2.4.2.3 Probabilistic Neural Networks

Indeed, recent extensions of NNs to probabilistic models exists, notably through Probabilistic Neural Networks (PNNs) [50]. PNNs scale better than GPs to the number of samples, but don't have the advantage of the possibility to include priors (easily) and require more hyper-parameter optimization. Nevertheless, PNNs were used to achieve state of the art results, notably on the half-cheetah benchmark [26].

PNNs also have other advantages over GPs, as they are able to model scenarios with multiple modalities and heteroskedasticity [40], scenarios that GPs fail to model properly. PNNs can also be used, together with latent input variable to decompose the various types of uncertainties in the model, so that do determine better which points to query in the next trials [41].

Going more into detail, a probabilistic neural network is a type of neural network that outputs probability distributions, typically characterized by a mean and a variance, instead of point estimates. This characteristic allows PNNs to model uncertainty and capture the inherent variability in predictions, making them particularly useful for tasks where uncertainty estimation is essential.

The loss function for training a probabilistic neural network takes into account both the accuracy of the predicted mean values and the uncertainty captured by the predicted variances. The goal is to minimize the discrepancy between the predicted probability distributions and the true distributions of the target data. A commonly used loss function for PNNs is the negative log-likelihood (NLL) loss, which measures

the dissimilarity between the predicted distribution and the actual target distribution.

Mathematically, given an input sample $\mathbf{x}$ and its corresponding true output $\mathbf{y}$ characterized by a Gaussian distribution with mean $\mu_{\mathbf{y}}$ and variance $\sigma^2_{\mathbf{y}}$, the NLL loss can be defined as:

$$\mathcal{L}(\phi) = -\log\left(p(\mathbf{y}|\mathbf{x};\phi)\right),$$

where ϕ represents the network parameters, and $p(\mathbf{y}|\mathbf{x};\phi)$ is the predicted probability density function of the output distribution given the input $\mathbf{x}$. For a Gaussian distribution, this density function can be expressed as:

$$p(\mathbf{y}|\mathbf{x};\phi) = \frac{1}{\sqrt{2\pi\sigma^2_{\mathbf{y}}}} \exp\left(-\frac{(\mathbf{y}-\mu_{\mathbf{y}})^2}{2\sigma^2_{\mathbf{y}}}\right).$$

Minimizing the NLL loss encourages the network to generate predicted distributions that align closely with the true distributions, effectively optimizing both the mean and the variance predictions. This enables the network to learn not only accurate point estimates but also to capture the uncertainty associated with the predictions.

One advantage of using a probabilistic neural network with the NLL loss is its ability to provide calibrated uncertainty estimates. The predicted variances can also serve as a measure of the model's confidence in its predictions. High variance indicates higher uncertainty either in the model or in the transition dynamics when they are stochastic, whereas low variance suggests higher confidence in the predictions or lower randomness in the system dynamics. This can be particularly valuable in scenarios where reliable uncertainty estimation is crucial for decision-making or risk assessment. Nevertheless, it would be interesting to be able to dissociate the model uncertainty, which is called the epistemic uncertainty, from the system dynamics uncertainty, which is called the aleatoric uncertainty, as having both mixed can undermine the decision capabilities of a policy search algorithm using such a model [64].

2.4.2.4 Ensembling

Theoretically, the most natural way to capture epistemic uncertainty would be using Bayesian inference. This is done in what we call Bayesian Neural Networks (BNNs), which instead of estimating the Neural Network parameter vector ϕ try to estimate the posterior distribution $p(\phi|\mathcal{D})$ over the Neural Network parameters depending on the collected data. Doing so basically maintain a distribution over the network parameters and allows to naturally separate the epistemic and aleatoric uncertainty. BNNs also require the introduction of a prior distribution over the model parameters, which is a choice left to the designer and often set to a normal prior with a zero mean and diagonal covariance [72]. Nevertheless, time-efficient BNNs are still a challenge especially at training time. Even if great advances were made recently, state of the art training algorithms for BNNs bring the training time to 2-5 times more than that of the ADAM algorithm for non-Bayesian Neural Networks [124].

Hopefully, other techniques which are more tractable exist to approximate a Bayesian Framework, like Batch Ensembles [166], Deep Ensembles [125] and Variational Neural Networks [56]. We take particular interest in Deep Ensembles, as they proved being efficient to learn Dynamics Models in robotic tasks in various contexts [26, 68, 144, 123, 98]. Deep Ensembles are pretty simple to understand, as they simply are an ensemble of N PNNs with different initial set of parameters $\phi_i \in \Phi, i \in \{1, \ldots, N\}$, and trained with the same data samples in different batches. Ensemble methods offer various ways to make predictions using the combined knowledge of those N models. One common approach is to average the predictions of individual models to obtain a final prediction. This simple yet effective technique helps mitigate the potential errors of individual models and produce more stable and accurate predictions [88] and moreover epistemic and aleatoric uncertainty can be separated:

$$p(\mathbf{y}|\mathbf{x};\phi) \sim \mathcal{N}(\mu_*(\mathbf{x}), \sigma_*^2(\mathbf{x})) \tag{2.5}$$

$$\mu_*(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\mu_i(\mathbf{x}) \tag{2.6}$$

$$\sigma_*^2(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}(\sigma_i^2(\mathbf{x}) + \mu_i^2(\mathbf{x})) - \mu_*^2(\mathbf{x}) \tag{2.7}$$

One would thus use $\mu_*(\mathbf{x})$ directly as the next step estimate. Nevertheless, some methods do not use the average of the mean predictions over the models

$$\mu_*(\mathbf{x})$$

directly as the estimated transition function next step prediction [13, 68, 26], and make more sophisticated uses of the various models in the ensemble to modify the predictions properties. One way to use the predictions differently is by preserving the particles on a single model of the ensemble, such that the variance $\sigma_*^2(\mathbf{x})$ can be divided into aleatoric and epistemic uncertainty:

$$\begin{aligned}
\sigma_*^2(\mathbf{x}) &= \frac{1}{N}\sum_{i=1}^{N}(\sigma_i^2(\mathbf{x}) + \mu_i^2(\mathbf{x})) - \mu_*^2(\mathbf{x}) \\
&= \frac{1}{N}\sum_{i=1}^{N}\sigma_i^2(\mathbf{x}) + \frac{1}{N}\sum_{i=1}^{N}\mu_i^2(\mathbf{x}) - \mu_*^2(\mathbf{x}) \\
&= \mathbb{E}_i[\sigma_i^2(\mathbf{x})] + \mathbb{E}_i[\mu_i^2(\mathbf{x})] - \mathbb{E}_i[\mu_i(\mathbf{x})]^2 \\
&= \mathbb{E}_i[\sigma_i^2(\mathbf{x})] + \mathbf{Var}_i[\mu_i(\mathbf{x})]
\end{aligned} \tag{2.8}$$

According to Equation 2.8, the right term, the variance of the means, corresponds to epistemic uncertainty, the uncertainty that stems from the lack of data and is measured through disagreement between the models of the ensemble. On the otherside, the left term is the mean of the variances and represents aleatoric uncertainty, the inherent stochasticity in the underlying transition function to be learned [160].

This thesis is focused on using such learned models, more specifically those based on Neural Networks architectures to increase Novelty Search sample-efficiency. Those learned models can be used in various ways as Neural Networks based architectures hold a great representational capacity [11, 20] making them perfect for modeling dynamics of robotic systems of various State-Action Space dimensions as well as modeling non-linear and even discontinuous transition functions [65]. Combining such modeling methods with Evolutionary Algorithms is thus at the core of this thesis, using the combination of both in various ways towards increasing Novelty Search sample-efficiency. But such models might yield very different prediction errors depending on the training data provided, especially in the first few interactions with the environment when the amount of training data available is relatively small. Next chapter addresses this issue by focusing on various random processes controlling the robot behavior to gather initial training data such that the learned models prediction error would be reduced in the context of Model-Based Policy Search.

Chapter 3

Bootstrapping Model-Based Policy Search: A Study

As presented in the precedent chapter, robotics learning suffers from several problems, most notably the reality gap problem and the sample efficiency problem. The reality gap can be addressed using a variety of techniques, but a family of method tries to overcome both the reality gap and the sample efficiency problem at once: model-based techniques. In this chapter, we take interest in such methods and more particularly into the way the data to initialize the models is gathered. Indeed, when dealing with a black-box environment, the only possible way to gather data is to give the agent actions to perform. In the case of model-based policy search algorithm, this initial data is used to train the model of the environment dynamics which is subsequently used to guide the policy search. The literature has addressed this initial data gathering problem mainly in two ways: use uniformly sampled random actions for a number of steps [122, 57, 144] and use randomly parameterized policies [98, 99]

Nevertheless, this poses a problem. Indeed, no consideration is given to the method used for initial data gathering and the impact it has on the model bootstrap depending on the environment at hand, meaning that we could use the same initialization method for any environment, even though it could not be suited at all to the dynamics of the given Markov Decision Process. Depending on the Model-Based Policy Search approach used, having a poorly initialized model could lead to sub-optimal behaviors for several episodes, which is detrimental to sample efficiency in a context where the goal is to maximize the information gain of each sample.

In this chapter, we thus propose to study the impact of the initial data gathering method on two Model-Based Policy Search algorithms: Dynamics-Aware Quality-

Diversity (DAQD) [98], an episode-based Model-Based Quality Diversity algorithm, and Probabilistic Ensembles with Trajectory Sampling (PETS) [26], a step-based Model-Based Reinforcement Learning algorithm. In order to provide more depth to our study, we will be studying the impact of five different initial data gathering techniques: Random Actions, Random Policies, and three variations of Colored Noise Random Walks. The last initial data gathering technique proposed is particularly interesting as it allows, independently from the environment at hand, to generate action sequences of varying time-correlation.

Those different initial data gathering techniques thus have varying degrees of auto-correlation, which in turn is linked to the ability to generate different data distributions in the action-space $\mathcal{A}$ and in the state-space $\mathcal{S}$ of a given environment. To determine beforehand what type of initial data gathering technique could work for a given environment, we propose a **consistency** metric, evaluating the propensity to which actions taken in the environment yield the same transition whichever state the system is in. We then link that consistency to the evaluated model prediction error to show that there is a relation between auto-correlation in the generated action sequences, model prediction error and environment consistency such that one could to some extent predict which type of method to use to bootstrap properly its Model-Based Policy Search algorithm model. Finally, we evaluate the actual impact this has on the two aforementioned Model-Based Policy Search algorithms

3.1 Methods

3.1.1 Initial Data Gathering Methods

One of the core principles of model-based techniques is that they iteratively alternate between exploration on the real system and exploration on the learned model. As stated before, in this study we focus on the data gathering method that will help pre-train the learned model. The objective is to have a model that is as much as possible properly bootstrapped given a limited budget that should be significantly inferior to the budget given to the subsequent model-based policy search method used. In this context, we consider several initialization methods that are detailed furthermore in the next paragraph.

Considering action $a_n \in \mathcal{A}$, what we call an initialization method, or initial data gathering method can be described as an action sequence, ideally independent of

the agent's environment E. We describe such an action a_n as a sequence of random variables A_n such that:

- $A_0 = a_0$
- $A_n = \sum_{i=0}^{n} \alpha_i Z_i$, Z_i being the random variable at step i

In this chapter, we'll consider the following different initialization methods:

- Random Actions denoted RA
- Colored Noise with $\beta = 0$ (Brownian Motion [117]) denoted $CNRW_0$
- Colored Noise with $\beta = 1$ denoted $CNRW_1$
- Colored Noise with $\beta = 2$ denoted $CNRW_2$
- Random Policies denoted RP

All of these different initialization methods can be formalized in the same manner, except for random policies as they depend on the policy representation. we start by describing the other initialization methods that do fit the presented formalism and will finish by detailing random policies.

Random Actions (RA):

Random actions: B uniformly drawn actions $a \in \mathcal{A}$, each applied for R step sequentially on the task horizon H such that $B = \frac{H}{R}$. An episode thus consists of rolling out B different actions, for R step each, on a newly reinitialized environment. In the previously presented formalism gives:

$$Z_i^{RA} \sim U(\mathcal{A})$$

Such that:

$$A_n^{RA} = Z_{n-nmodB}^{RA}$$

Colored Noise Random Walk (CNRW): [130] proposed correlated action sequences through modifying the power spectral density (PSD) of that time-series. Indeed, the correlation structure of a sequence, if looked at as a time series is directly connected

to the power spectral density (PSD) [130]. As such, it is possible to define colored-noise sequences as a function of a β factor which creates correlated action sequences such that $PSD(f) \propto \frac{1}{f^{\beta}}$. It is important to note that with a β factor equal to zero, the Colored Noise distribution is equivalent to a Gaussian distribution $\mathcal{N}(\mu, \sigma)$, such that the generated noise sequence is equivalent to white-noise and the associated action sequence is thus equivalent to Brownian Motion [117]. We emphasize the fact that Random Actions and $CNRW_0$ (Brownian Motion) are distinct methods, as the first produces uncorrelated action sequences. Colored Noise Random Walk can thus be described as follow, with β modifying the PSD of the random variable Z_i as described before:

$$Z_i^{CNRW_\beta} \sim \mathcal{N}_{CN}^i(\mu, \sigma, \beta)$$

Such that:

$$A_n^{CNRW_\beta} = \sum_{i=0}^{n} Z_i^{CNRW_\beta}$$

Where $\mathcal{N}_{CN}^i(\mu, \sigma, \beta)$ represents the Colored Noise distribution conditioned on the β factor and the $i-1$ previous samples regarding the i_{th} sample, as the sampled noise sequence is highly time correlated.

Random Policies (RP): In the scope of this chapter, Random Policies will refer to two-layers deep neural networks, with ten neurons per layers. The input layer of the network is the current state s_n the agent is in and the output layer is the current action a_n the RP controller outputs for state s_n depending on the controller parameterization by vector θ, the parameters of the vector being the weights and biases of the underlying neural network. Simply put:

$$a_n = RP(s_n, \theta)$$

RP controllers are thus directly dependant on the environment, while all the previously mentioned controllers are completely environment independent. Previous controllers, except for Random Actions, are all Random Walks in the action space $\mathcal{A}$ of the agent's environment E. The interest that we take in this study is thus to evaluate the importance of the time-correlation in the action sequences used to generate initial training data as the described initialization methods cover the range of time-correlation values.

3.1.2 Environment consistency metric

To measure the dynamics consistency of an environment, we propose to use the coefficient of variation [2] of the $\Delta s = s_{t+1} - s_t$ data distributions obtained by sampling uniformly a single action $a \in \mathcal{A}$ in M multiple uniformly sampled states $s \in \mathcal{S}$ and acting in the environment using the true transition function. Multiple coefficient of variation measures are then obtained for all the N sampled actions, and we look at the mean and standard deviation of these measures across the system state as outlined in Algorithm 3. The coefficient of variation is defined as follow, being the ratio of the aforementioned distribution standard deviation to the mean:

$$c_v = \frac{\sigma}{\mu} \tag{3.1}$$

The consistency metric is thus defined as $u = 1 - c_v$, such that when observed Δs variation for sampled actions is high, environment consistency is low and *vice versa*. Observed coefficient of variations are then averaged over sampled actions.

Algorithm 3 Consistency measure

Sample uniformely N actions $a \in \mathcal{A}$
for each action a **do**
    Sample uniformely M states $s \in \mathcal{S}$
    for each state s **do**
        $(S, A, NS) \leftarrow t(s, a)$ ▷ Collect real transition data
        $\Delta S \leftarrow NS - S$ ▷ Append transition residual to ΔS
    $u_A \leftarrow 1 - \frac{\sigma_{\Delta S}}{\mu_{\Delta S}}$ ▷ Compute coefficient of variation of distribution of transitions for action a and corresponding consistency

return $mean(u_A), std(u_A)$

3.1.3 PETS and DAQD

The Probabilistic Ensembles with Trajectory Sampling (PETS) algorithm is a model-based reinforcement learning (MBRL) approach designed for sample-efficient learning with limited trials. PETS utilizes probabilistic dynamics models, as presented in Section 2.4.2.3 to learn the system dynamics, and train them as an ensemble to capture various sources of uncertainty as described in Section 2.4.2.4. PETS proposes to employ trajectory sampling, generating multiple trajectories by sampling from the learned dynamics models, and utilizes the cross-entropy method [35] to optimize for the best next action to perform under a specified planification horizon. PETS alternates between optimizing for the best return under its current modelization

of the environment dynamics the next action to perform and acting in the environment. By combining probabilistic modeling, ensemble learning, and trajectory sampling, PETS is a sample-efficient Model-Based Policy Search algorithm that can solve robotic tasks with only a small number of interactions, making it particularly suitable for scenarios where trials are limited. The algorithm is further detailed in 4, with TS referring to both trajectory sampling methods T_1 and T_{inf} described below.

Algorithm 4 PETS

Initialize data $\mathcal{D}$ with an initial data gathering method for 10 episodes
▷ *Performing PETS*
for k *in* $[1, \ldots, K]$ **do**
    Train a PE dynamics model $\hat{T}$ given $\mathcal{D}$
    for t *in* $[0, \ldots, H]$ **do**
        for Action sampled $\mathbf{a}_{t:t+T} \sim CEM(\cdot)$, 1 to N Samples **do**
            Propagate state particles $\mathbf{s}_\tau^p$ using TS and $\hat{T}$
            Evaluate actions as $\sum_{\tau=t}^{t+T} \sum_{p=1}^{P} r(\mathbf{s}_\tau^p, \mathbf{a}_\tau)$
            Update $CEM(\cdot)$ distribution
        Execute first action $\mathbf{a}_t^*(only)$ from optimal actions $\mathbf{a}_{t:t+T}^*$
        Record outcome: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathbf{s}_t, \mathbf{a}_t^*, \mathbf{s}_{t+1}$

Chua *et al.* propose two methods of Trajectory Sampling:

- TS_1 corresponds to uniformly selecting a model from the ensemble at each time step to propagate the current prediction or *particle*. This corresponds to considering the ensemble as a Bayesian model as each particle would be resampling from the approximated marginal posterior of the system dynamics.

- TS_{inf} corresponds to never changing the model from the ensemble being used to infer a particle trajectory. A model is uniformly sampled at the first step and thus used for the planning horizon. The idea behind TS_{inf} is that the ensemble is a collection of plausible models. Not changing the model used during inference thus captures the time-invariance of the real system dynamics function T, considering each model of the ensemble as a plausible approximation of the true dynamics function. Moreover, using TS_{inf} allows for separation of epistemic uncertainty and aleatoric uncertainty [41] as shown in Equation 2.8, where aleatoric uncertainty is captured by the average variance of the particles inferred on the model ensemble and where epistemic uncertainty is the variance of the average of particles for each time-step.

In this chapter, we will be using the TS_1 propagation method, as it yielded the best results in the original PETS paper.

The other method we will be using is the Dynamics-Aware Quality-Diversty (DAQD) algorithm, which is a Model-Based Quality-Diversity algorithm (and more broadly a Model-Based Evolutionary algorithm). Much like PETS, DAQD utilizes probabilistic models as an ensemble to provide more accurate predictions of the roll-out of a candidate solution. As opposed to PETS, DAQD uses TS_{inf} for the prediction of the individual trajectory, from which is derived the behavioral descriptor using the observer function $o_{\mathcal{B}}$ and the individual fitness using the fitness function. DAQD thus transfer the QD search onto the model using a random selection process and directional variation [162]. Once enough individuals are found on the model to potentially be added to the archive (100 is the number used in our and their experiments), all individuals found are transferred onto the real system and the archive selection process on the real system takes place, finally adding to the archive only the individuals that are actually either filling a new bin or that replace a previous individual. The complete DAQD algorithm is detailed in 5.

In this chapter, we will use two different version of the DAQD algorithm. Firstly, the Vanilla version, which bootstraps the archive like a regular QD algorithm using a fixed number of randomly parameterized policies before performing any search using the model. The data gathered doing so is also used to train the dynamics model. Secondly, we will use a custom version of the DAQD algorithm which consist of two phases: the initialization phase, where a specified budget of an initial data gathering method is spent to train the model and a second phase of QD search on the model, similar to Vanilla DAQD after its bootstrap phase. The budget for initialization in the custom DAQD version will be 10 times lower than the one of the Vanilla DAQD algorithm.

3.2 Experiments

3.2.1 Experimental Setups and corresponding Consistency

In our experiments, we consider five setups, with various consistency metrics and some with a distinct characteristic not present in any other environment and representative of real world robotic setup commonly found dynamics or specificities. Four characteristics are thus studied: sparse interactions (in the Ball In Cup scenario), which are the fact that some transitions appear very scarcely in the transition function T, discontinuities in the transition function dynamics (in the Two-wheeled Robot Maze scenario), and finally physical traps (in the Two-wheeled Robot Maze Traps

Algorithm 5 Dynamics-Aware Quality-Diversity (DAQD)

Initialize repertoire $\mathcal{A}_\Pi$ (to $\emptyset$), imagined repertoire $\widetilde{\mathcal{A}_\Pi}$ (to $\emptyset$), dynamics model p_{dyn}, and replay buffer $\mathcal{B}$ (to $\emptyset$)

while maximum number of evaluations not reached **do**

  if first iteration **then** ▷ *Generate random policies at first iteration.*

    $\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*} \leftarrow$ random_parameters()

  else

    $\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_b} \leftarrow$ select($\mathcal{A}_\Pi$) ▷ *Selecting b policies from repertoire $\mathcal{A}_\Pi$*

    $\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*} \leftarrow$ variation_operators($\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_b}$) ▷ *Apply variations to parameters $\boldsymbol{\theta}_i$.*

  ▷ *Performing QD Exploration in Imagination*

  while model stopping criterion is False **do**

    $(\widetilde{\boldsymbol{sd}_i}, \widetilde{R_i})_{i\in[1,b]} \leftarrow$ imagined_rollouts($\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*}$) using p_{dyn} ▷ *Evaluate using dynamics model.*

    $\widetilde{\mathcal{A}_\Pi} \leftarrow$ model_condition($\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*}, \widetilde{\mathcal{A}_\Pi}$) ▷ *Add $\pi_{\boldsymbol{\theta}_i^*}$ to $\widetilde{\mathcal{A}_\Pi}$ depending on $\widetilde{R_i}$ and novelty of $\widetilde{\boldsymbol{sd}_i}$.*

    if model stopping criterion is False **then** ▷ *Continue performing QD exploration in imagination.*

      $\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_b} \leftarrow$ select($\widetilde{\mathcal{A}_\Pi}$) ▷ *Selecting b policies from imaginary repertoire $\widetilde{\mathcal{A}_\Pi}$.*

      $\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*} \leftarrow$ variation_operators($\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_b}$)

  ▷ *Acting in the Environment*

  $\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_N} \leftarrow$ select($\widetilde{\mathcal{A}_\Pi}$) ▷ *Selecting N policies from imaginary repertoire $\widetilde{\mathcal{A}_\Pi}$ to be evaluated.*

  $(\boldsymbol{sd}_i, R_i)_{i\in[1,N]}$, transitions $\leftarrow$ evaluation($\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_N}$) ▷ *Evaluate in environment; get transitions.*

  $\mathcal{B} \leftarrow$ add_to_replay_buffer(transitions, $\mathcal{B}$)

  $\mathcal{A}_\Pi \leftarrow$ repertoire_condition($\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_N}, \mathcal{A}_\Pi$) ▷ *Add $\pi_{\boldsymbol{\theta}_i}$ to $\mathcal{A}_\Pi$ depending on R_i and novelty of $\boldsymbol{sd}_i$.*

  $\widetilde{\mathcal{A}_\Pi} \leftarrow$ synchronise_repertoires($\mathcal{A}_\Pi$, $\widetilde{\mathcal{A}_\Pi}$) ▷ *Erase content of $\widetilde{\mathcal{A}_\Pi}$ and replace it with the content from $\mathcal{A}_\Pi$.*

  ▷ *Learning Dynamics Models*

  Update p_{dyn} using $\mathcal{B}$ ▷ *Train dynamics model with transitions collected in replay buffer.*

return $\mathcal{A}_\Pi$

scenario), much like a robot falling into a pit not being able to escape it or dropping an object crucial for the end of its task out of its reach, such that task completion isn't possible anymore. We also consider two other environments for their different consistency and their application with the PETS algorithm: the Cartpole scenario and the Pusher scenario directly taken from the PETS paper [26].

The two Two-Wheeled Robot Mazes scenarios will be used exclusively with DAQD and are based on the Hard Map from the original Novelty Search paper

[91]. Those environments are not used with PETS as designing a simple dense reward function would be deceptive and unhelpful. The Cartpole and Pusher will be used exclusively with PETS. Regarding the Ball In Cup task, it is an extension of the two-dimensional Ball In Cup environment from the DeepMind control suite [154]. It will be used with both methods as a specific dense reward has been designed, such that the agent is rewarded when swinging up the ball above the cup, then for reducing the distance between the ball and the cup and finally by successfully putting the ball inside the cup, allowing for usage of the PETS algorithm on this task.

3.2.1.1 Two-Wheeled Robot Maze

The Two-Wheeled Robot Maze environment consists in a two-wheeled robot navigating in a maze that is hard to explore, taken from [91]. The robot wheels are independently speed controlled, the action-space of the robot thus is two dimensional and consists of the motor command to send to each of the robot wheels. The state-space consists of the position, velocity and orientation of the two-wheeled robot. The orientation ϕ, that is expressed in degrees, is divided in two fields corresponding to $sin(\phi)$ and $cos(\phi)$, to maintain continuity in the state values whatever the value of ϕ is. The state-space is thus of dimension 6. The task consists in reaching as many robot positions as possible, the behavior descriptor thus being the end position of the two-wheeled robot trajectory over an episode. Two versions of the task are considered: one in which the walls act as discontinuities in the transition function, the robot simply colliding with them when touching them, and an other one in which the walls act as physical traps, effectively ending the episode. The episode length is 1000 steps on these environments. An image of the setup is shown on Figure 3.1d. The outcome space we consider for this environment is the final position x-axis and y-axis position of the robotic mobile base. The two environments are based on the Fastsim simulator [119].

3.2.1.2 Ball In Cup

The Ball-In-Cup environment consists in a ball that is hanging below a cup thanks to a string. The cup is controlled in position in the three-dimensional space. The action-space thus consists in the three dimensional position of the cup in the 3D space. The state-space consists of the relative position and velocity of the ball to the cup. The state-space is thus of dimension 6. The task thus consists in putting the ball inside the cup. The task can either have a sparse reward function or a dense

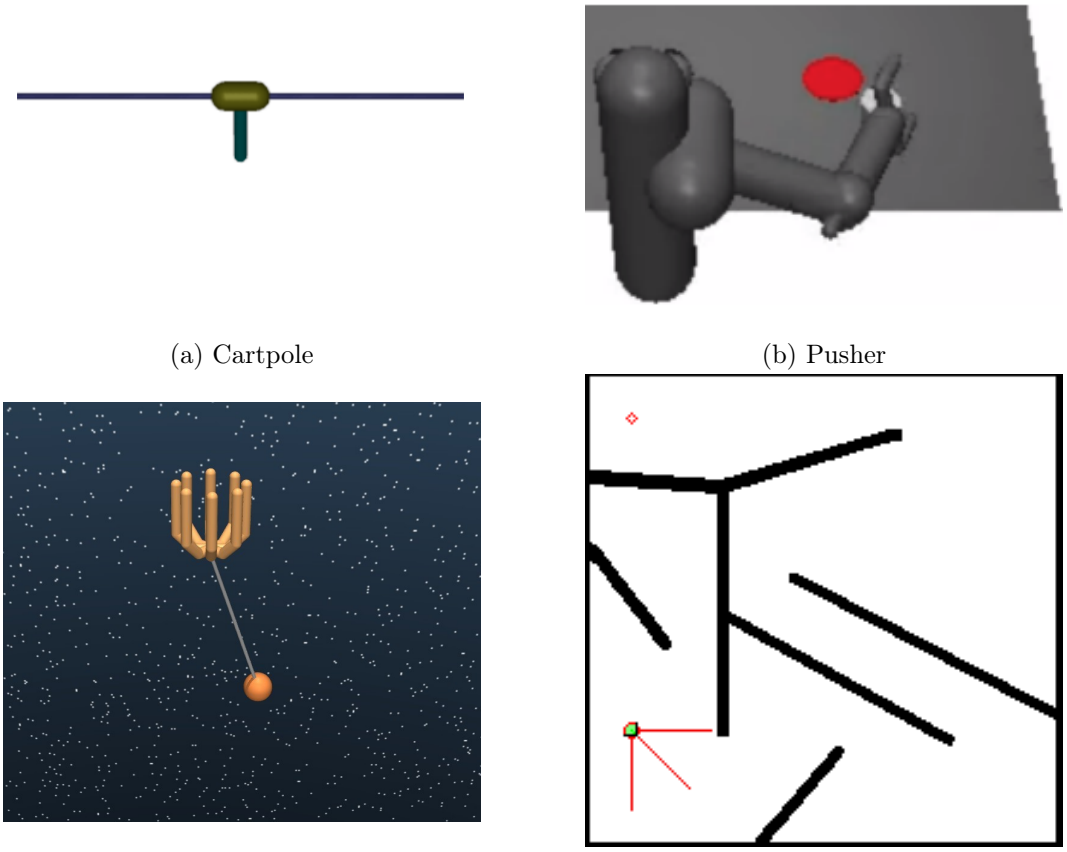
reward function. In the sparse reward scenario, reward is only given when the ball gets into the cup. In the dense reward scenario, a specific dense reward function has been designed, such that the agent is rewarded when swinging up the ball above the cup, then for reducing the distance between the ball and the cup and finally by successfully putting the ball inside the cup. In both scenarios, this tasks presents an important challenge as the problem involves sparse interactions. Indeed, to give the ball an upward velocity requires to swing the ball, which can only be done through a portion of executable action sequences. An image of the setup is shown on Figure 3.1c. The outcome space we consider for this environment is the relative position of the ball to the cup. The episode length is 300 steps on this environment.

3.2.1.3 Cartpole

The cartpole environment is a classic robotic control environment consisting of a pole linked to a cart through an unactuated revolute joint as shown on Figure 3.1a. The cart itself is then controlled in a single dimension along a prismatic joint. The action-space thus consists in a single action representing the target velocity (positive or negative) of the cart along the prismatic joint. On the other hand, the state-space consists of the current cart position and velocity along the prismatic joint as well as the angular position and angular velocity of the pole linked to the cart. The state-space is thus of dimension 6. Here, the given task is to swing-up the pole and keep it upright. The closer the pole is to its upright position, the higher the reward is. This task, as well as the Pusher task, have a non-deceptive dense reward function and is ideal for step-based model-based techniques that require a dense reward, like PETS [26]. Both the Cartpole and the Pusher environments are directly taken from the PETS experiments. The episode length is 150 steps on this environment.

3.2.1.4 Pusher

The final environment we consider is the pusher environment which consists in a 7 degrees of freedom robotic arm of a PR2 robot, of a puck on a table and of a target position for the puck to reach, as shown on Figure 3.1b. The robotic arm is torque controlled in each of its joints. The action-space thus consists in a vector of dimension seven, each of its dimensions being the torque to apply to each of the seven joint. The state-space consists of the current angular position and velocity of the robotic arm joints, of the end-effector Cartesian position, of the puck Cartesian position and finally of the goal Cartesian position. The state vector is thus a 23



(a) Cartpole

(b) Pusher

(c) Ball In Cup

(d) 2-wheeled Robot Maze

Figure 3.1: Benchmark environments

dimensional vector. The task objective is to push the puck towards the goal position on the table using the robotic arm. The reward function is a weighted mix of the distance between the end-effector and the puck and of the distance between the puck and its goal position. The episode length is 150 steps on this environment.

The consistency metric is being computed on a budget of $N = 1000$ actions sampled uniformly in the action-space $\mathcal{A}$, each action being applied in $M = 1000$ states sampled uniformly in the state-space $\mathcal{S}$, for a total of 1 million observed transitions. Observed Δs are obtained using the real transition function t and are min-max normalized *w.r.t.* the observed minimum and maximum values of Δs for each of the state dimensions. Figure 3.2 shows the consistency measure for each of the considered benchmark environments. The cartpole environment has the highest consistency metric around 0.9, while the Pusher environment is slightly above 0.7 followed closely by the two Two-Wheeled Robot Maze environments (fastsim_maze and fastsim_maze_traps) at 0.67 and finally by the Ball In Cup environment right below 0.65. We observe that the cartpole has a greater standard deviation than the

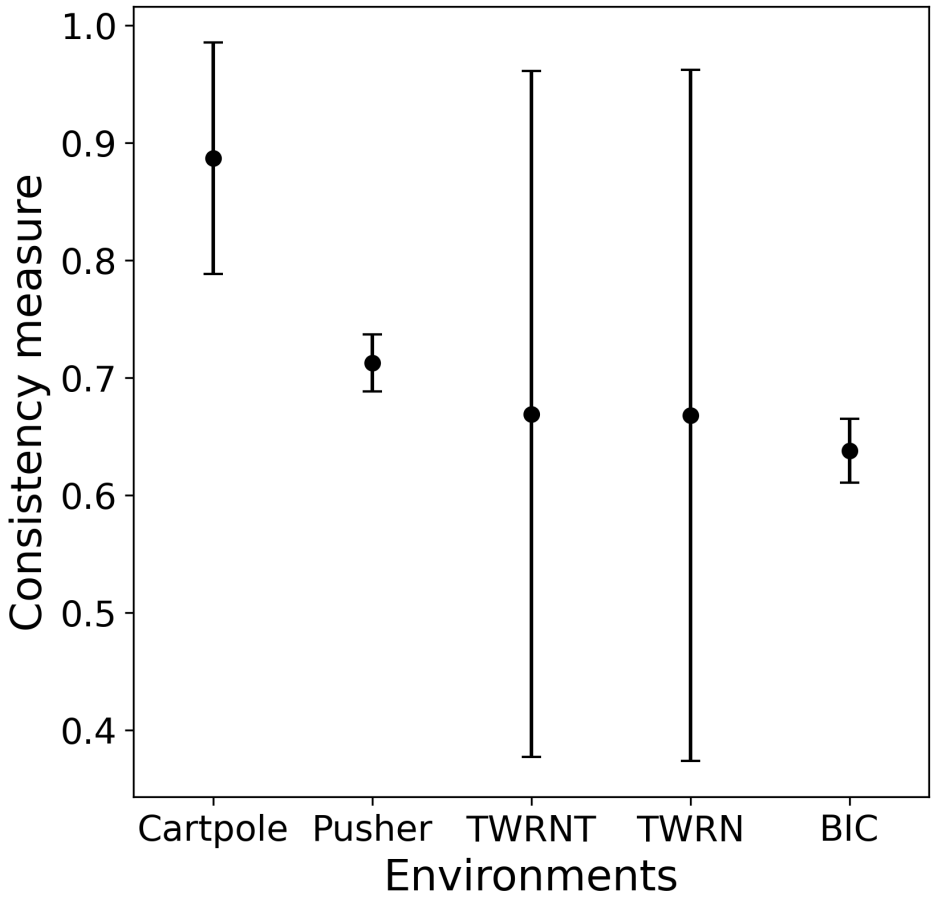


Figure 3.2: Consistency metric for benchmark environments

Pusher and Ball In Cup environment which is most likely due to the cartpole state dimensions, comprised of the slider position and pole angle, following very different dynamics. Finally, we observe that the two Two-Wheeled Robot Maze have a very great standard deviation in their consistency. In this case, it seems to indicate that the environments are both very consistent (in the regions without any obstacle) and very inconsistent (in the regions with obstacles). Results on the two Two-Wheeled Robot Maze might be harder to interpret given the inconsistent consistency metric of these environments and their long prediction horizons.

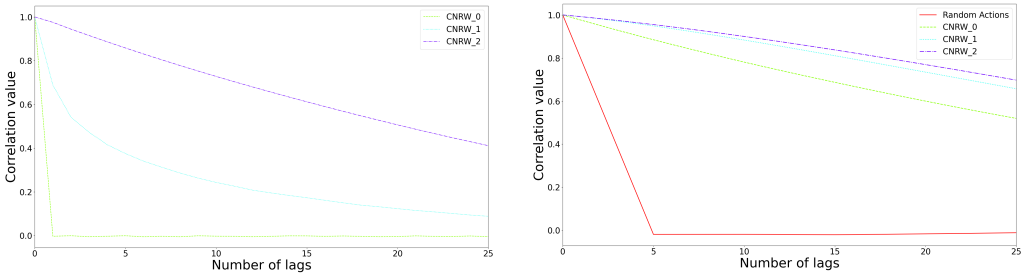
3.2.2 Data gathering policies

3.2.2.1 Auto-correlation

We first want to ensure that the initialization methods we consider are set up properly at different values in the time correlation spectrum. In that case, Random Actions

represent the least time-correlated end of that spectrum, the actions being uniformly re-sampled every R time-step. We now want to have the other proposed methods laid at the opposite of that spectrum, ideally forming a gradient of time-correlation. We thus propose to look at the auto-correlation of the generated action sequence to ensure that we have the intended action sequences behavior.

In order to do so, we first observe the auto-correlation of the generated noise sequences, to ensure that an increasing β factor in Colored Noise results in higher time-correlation in the noise sequences. We consider a time lag of up to 25 elements as we find this a suitable value to observe the differences in time-correlation between the considered noise generators. As expected and as observed on Figure 3.3a, Brownian Motion ($CNRW_0$) underlying noise generator produce a noise sequence with near zero auto-correlation values. On the contrary, $CNRW_1$ and $CNRW_2$ underlying noise generators produce higher time-correlated noise sequences. Indeed, we can see on Figure 3.3a that Colored Noise with a β factor equal to one has auto-correlation values that decreases exponentially while with a β factor equal to two the auto-correlation values diminishes almost linearly. Being sure that the underlying noise generators are working as intended, we now observe the same graphs directly in the action-space $\mathcal{A}$ to ensure that the observed properties carries on from $\mathcal{R}$.



(a) Noise sequence auto-correlogram

(b) Action sequence (random walk in action space) auto-correlogram

Figure 3.3: Auto-correlograms of noise and action sequences on a time lag of 25 sequence elements

Firstly, it is important to note that, by nature, Random Walks induce time-correlation in the generated sequences. Shown results are obtained on 1000 randomly sampled sequences for each method. As observed on Figure 3.3b, Brownian Motion ($CNRW_0$) produce a pretty low time-correlation on the considered time lags. We now want to confirm that $CNRW_1$ and $CNRW_2$ have higher auto-correlation over those input lags, and that $CNRW_2$ is more time-correlated than $CNRW_1$. We can observe on 3.3b that $CNRW_2$ is indeed more time-correlated than $CNRW_1$, as $CNRW_2$ correlation value for a 25 time-lag, $CNRW_2$ is settling around 0.7, while

$CNRW_1$ settles around 0.65. The decrease in time-correlation across time is thus steeper for $CNRW_1$ with regards to $CNRW_2$, which is the intended behavior. Now that we ensured that we have various time-correlated initial data gathering methods, we continue this study by analyzing the initialized models prediction errors and by putting these into perspective with the action sequences time-correlations and the consistency metrics of each of the environments.

3.2.2.2 Data distribution study

In this section, we focus on studying the data distributions produced by each of the initialization methods. We both observe the data distribution obtained on actions, which in most cases is the same across all dimensions of the action space $\mathcal{A}$ for a single initialization method, and observe the resultant state data distribution obtained taking the actions produced by the initialization method. All results are gathered with a 10 episodes budget on all environments for each method with 10 repetitions. Shown state-space dimensions are usually representative of the other dimensions which are not shown.

Here, the goal is to determine which initialization method state-action pairs are the most spread in the state-action space. Indeed, this should increase model prediction capabilities, as well as allow for more generalization to unseen states as more various data is fed to the model for training. What is to be noted is that inducing more time correlation in the initial data gathering method will in most environments give the model a broader data distribution in the state space $\mathcal{S}$ to learn from. Nevertheless, it needs to be taken into consideration that as much as visiting more diverse states can bring better generalization capabilities overall, having a denser training data support can help reduce variance in predictions among the models of the ensemble. Moreover, as mentioned previously, four environments will be considered, and each of the selected dimensions to visualize are represented as histograms on figures 3.4, 3.5, 3.6, 3.8 and 3.7.

Ball In Cup

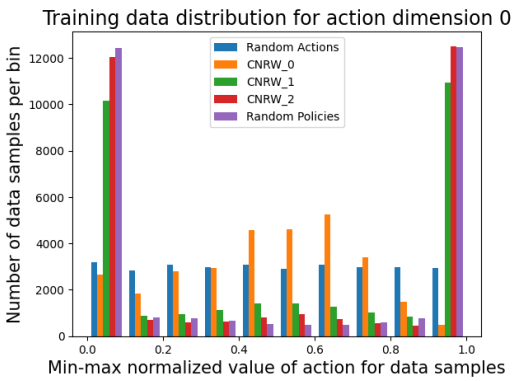
Looking at the distribution of the generated actions on the Ball In Cup environment on Figure as shown on figure 3.4a, we observe that Random Actions effectively provide uniformely distributed data in the action space. The other methods provide a sort of gradient of the data distribution between Random Actions and Random Policies in the action space, with $CNRW_0$ being the closest to uniformely distributed data in the action space, while more time-correlated initialization meth-

ods like $CNRW_1$ and $CNRW_2$ are closer to Random Policies, having most of their action values at the limit of the action space. We remark that as intended, $CNRW_1$ appears closer to a uniform data distribution in the action space than $CNRW_2$, even if both are far from being effectively uniform in the action space. Most of the remarks made on the action-space data distribution here are also valid for other environments and will not be repeated for each environment.

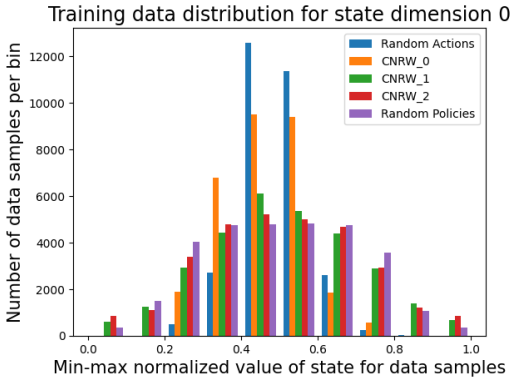
On the state-space side, we observe that on this environment higher time-correlated initial data gathering methods visits more different states, reaching regions of the state space that lesser time-correlated initialization methods, like Random Actions and Brownian Motion fail to reach as shown on figures 3.4b, 3.4c and 3.4d. On this environment, this is mostly due to the sparse interaction in the dynamics of the system of giving the ball an upward speed (as shown on figure 3.4d) which is only reachable if the same motion of the cup is continued for enough time steps. This translates in more states visited along the z-axis (as shown on figure 3.4c) for higher time-correlated methods. On the x-axis and y-axis, all methods more or less visit the sames states as shown on figure 3.4b but higher time-correlated techniques still visits more uniformly states along those axes. It is thus expected that higher time-correlated initialization methods will yield better prediction results, as they are able to visit more diverse states. $CNRW_1$ should perform among the bests in terms of prediction error, as it seems to be the closest to well spread training data in the state-action space.

Two-Wheeled Robot Maze

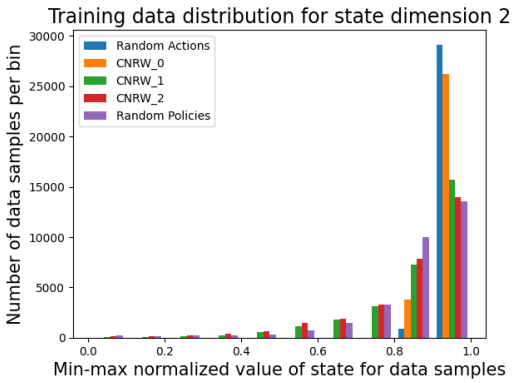
Looking at the state-space data distributions on the Two-Wheeled Robot Maze environment on figure 3.5, we observe that more time-correlated initialization methods have wider data distributions that reach states that lower time-correlated initial data gathering methods like Random Action do not reach. In more detail, we observe on first and second dimensions of the state vector, which corresponds to the x-axis and y-axis robot position in the maze that methods like Random Actions or $CNRW_0$ remain in the same area around the initial state. On the other hand, even though methods with higher time correlation should hit the walls of the maze quite frequently, $CNRW_1$ seems to be the initialization method that explores the most the environment. We would thus expect this method to yield best results in terms of minimization of the model prediction error. Nevertheless, this environment dynamics are quite simple when setting the discontinuities (the walls) aside, so we also could expect small prediction error from such methods if the walls do not change the robot trajectory too drastically.



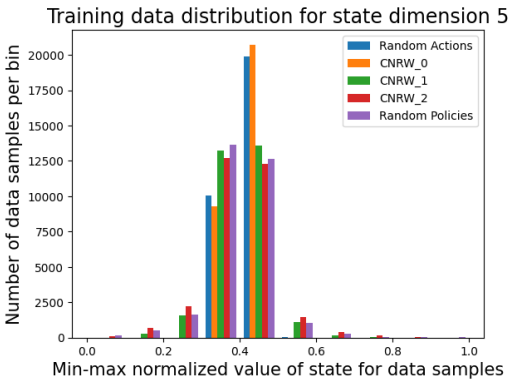
(a) 1^{st} dimension of action vector



(b) 1^{st} dimension of state vector



(c) 3^{rd} dimension of state vector



(d) 6^{th} dimension of state vector

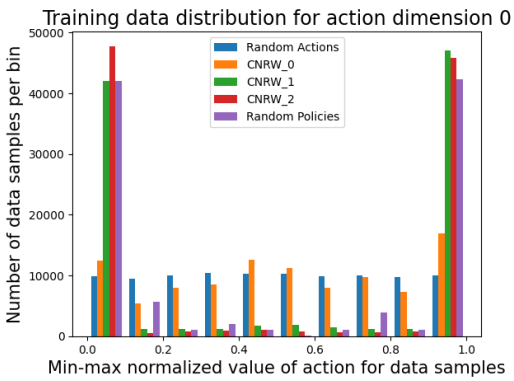
Figure 3.4: Training data distributions on Ball In Cup environment

Two-Wheeled Robot Maze Traps

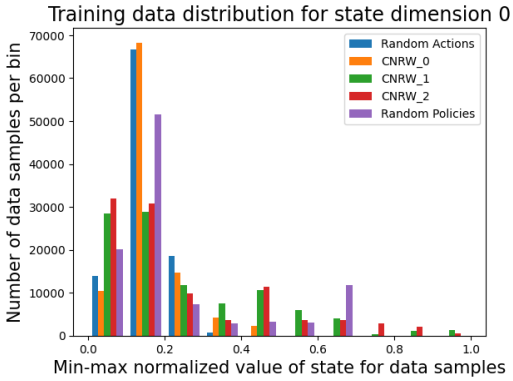
We now take a look at the same maze environment but change the walls impact on the dynamics. Instead of simply being an obstacle for the robot, walls now end the data gathering episode when hit by the robot. We expect that this would create a disadvantage for highly correlated trajectories that might hit walls while going further in a single direction, effectively going far from the safe initial state of the system. Indeed, as we can see on the figure 3.6, initialization methods that have high time correlation in their action sequences have quantitatively less training samples than other methods with less correlation in their action sequences, like Random Actions or Brownian Motion. We thus expect to see a drop in prediction quality for the model trained with higher time-correlated initialization methods, as they have lower amounts of data to train from on the Two-wheeled maze trap field environment.

Cartpole

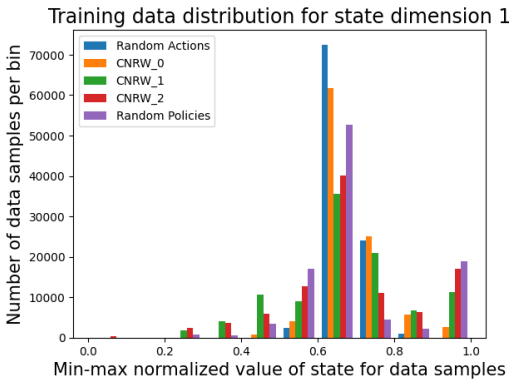
We now look at the data distribution on the Cartpole environment on Figure



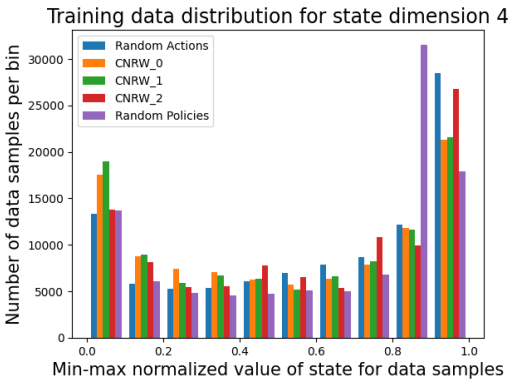
(a) 1^{st} dimension of action vector



(b) 1^{st} dimension of state vector



(c) 2^{nd} dimension of state vector



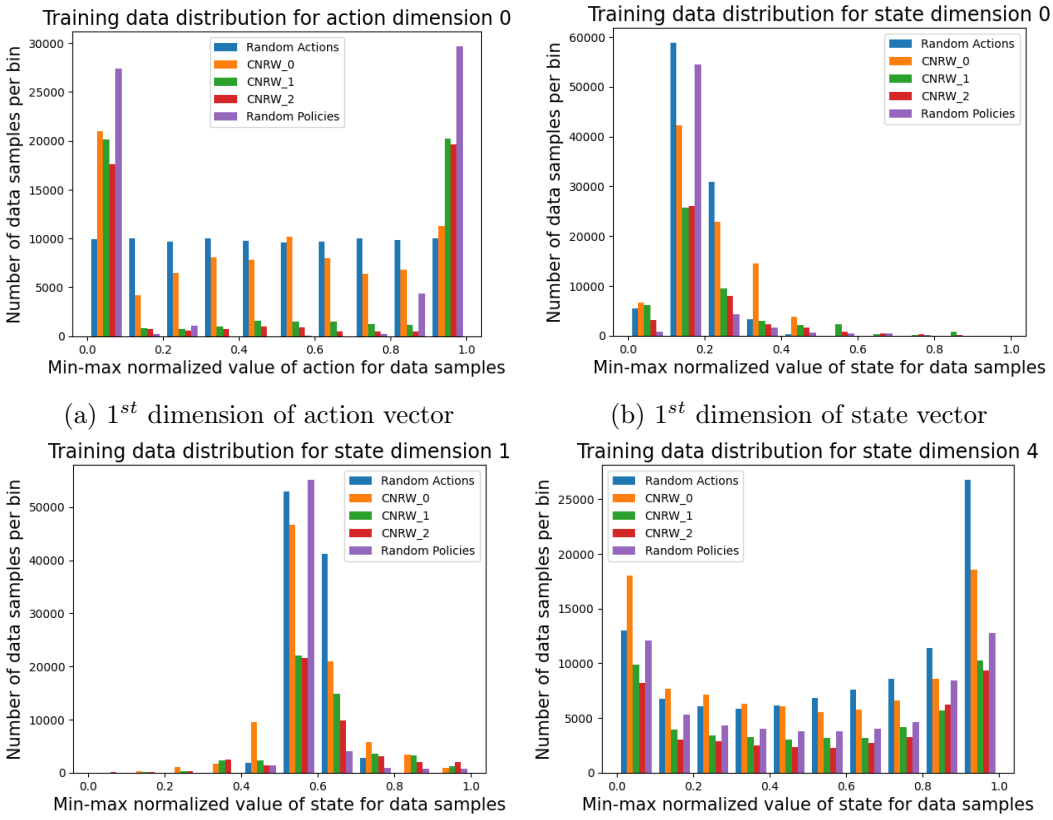
(d) 5^{th} dimension of state vector

Figure 3.5: Training data distributions on Two-wheeled robot maze environment

3.7. We only look at the state-space data distribution, as the action space data distribution is similar to the priorly exhibited ones. On the Cartpole environment, we look at 4 dimensions of the state-space: the cart position and velocity, and the pole angular position (decomposed into its cosine and sine parts for continuity). We observe that Random Actions allow for visitation of more diverse states for every studied dimension of the state-space. Moreover, looking at Figure 3.7a, we can clearly see that lesser time-correlated methods, like random Actions and $CNRW_0$ are the most well-spread in all the observed state-space dimensions, while higher time-correlated methods tend to maintain the system in the same states. According to this distribution data analysis, Random Actions should give the least prediction error on the Cartpole Environment among the five considered initialization methods, which corroborates the idea that an environment with a high consistency metric should work best with lower time-correlated initialization methods.

Pusher

Going onto the Pusher environment generated distributions on Figure 3.8, we



(a) 1^{st} dimension of action vector

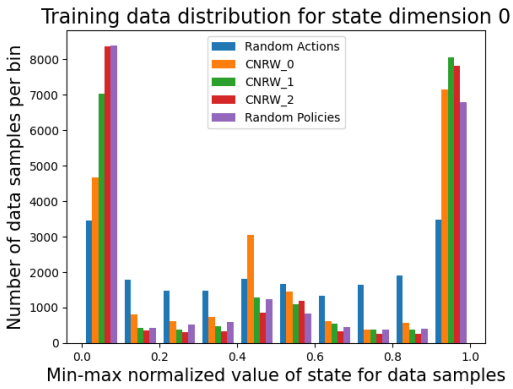
(b) 1^{st} dimension of state vector

(c) 2^{nd} dimension of state vector

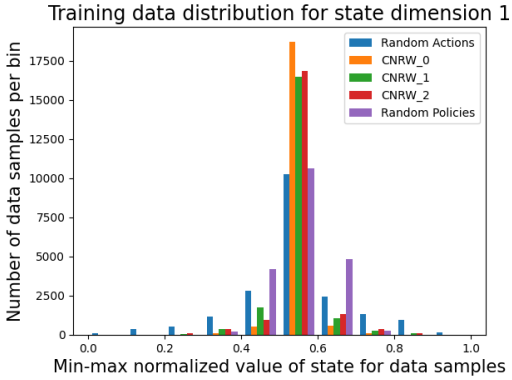
(d) 5^{th} dimension of state vector

Figure 3.6: Training data distributions on Two-wheeled robot maze trap field environment

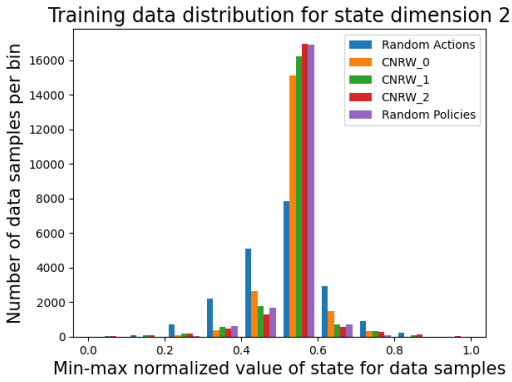
once again only look at the state-space data distribution, as the action space data distribution for similar reasons as previously. On the Pusher environment, we look at 4 dimensions of the state-space: the x and y position of the end-effector of the robotic arm, and the x and y position of the palet the robot can interact with. As expected, we observe on Figure 3.8a and Figure 3.8b that higher time-correlated initialization methods tend to reach more diverse end-effector positions more often than lower time-correlated initialization methods. Nevertheless, when looking at the palet x and y positions attained on Figure 3.8c and Figure 3.8d, all methods more or less keep the palet in its original position, with some rare occurences making the palet move. In this environment, we would thus expect higher time-correlated methods to have a slightly lower prediction error as they visit more diverse states, but not an important difference as all methods fail to interact with the palet. Indeed, being able to move the palet is central to solving the task, and a model not being able to predict what actions will move the palet won't be better at solving the problem given at bootstrap, no matter how good it is to predict the rest of the system dynamics.



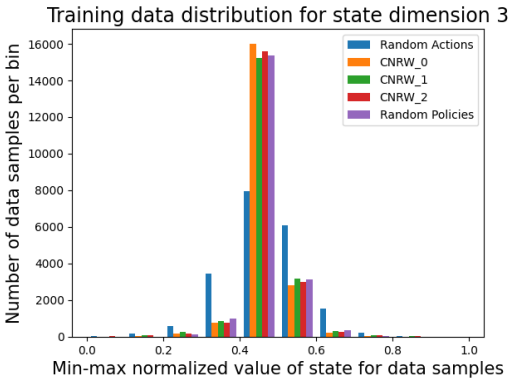
(a) 1^{st} dimension of state vector



(b) 2^{nd} dimension of state vector



(c) 3^{rd} dimension of state vector



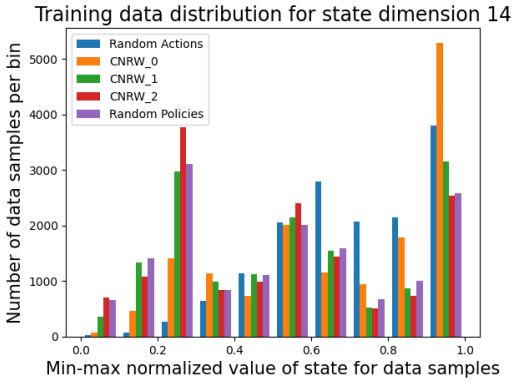
(d) 4^{th} dimension of state vector

Figure 3.7: Training data distributions on Cartpole environment

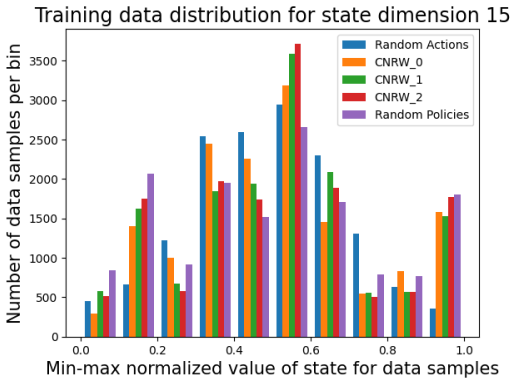
3.2.2.3 Prediction error analysis

Prediction errors are obtained by taking the average mean prediction error over 10 repetitions of models trained using the initialization methods detailed previously. The prediction error is computed against a diverse set of 10 trajectories resulting from Novelty Search [91] runs, so as to capture as much as possible of the system dynamics. Trajectories are specifically chosen as to be as diverse as possible behavioraly speaking, such that their trajectories are representative of the real system dynamics. Predictions are made on two different horizons, to highlight the model performance under different planning schemes. We thus show model prediction error on 1-step predictions and on 25-step predictions (which is the PETS planification horizon for all of the three considered tasks). Predictions are obtained using TS_1 particle propagation and averaging predictions over all particles for the Cartpole, Ball In Cup and Pusher environment.

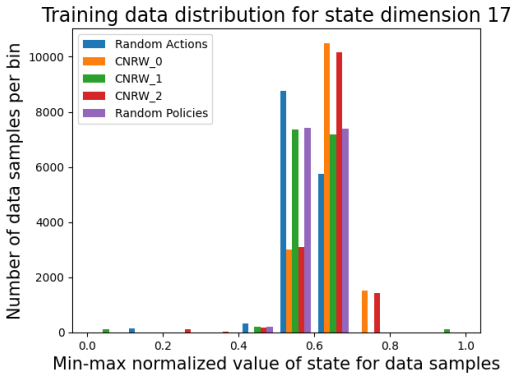
For environments that are used with DAQD , the predictions are also obtained on



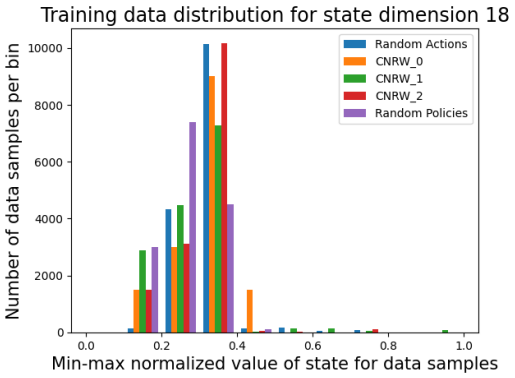
(a) 15^{th} dimension of state vector



(b) 16^{th} dimension of state vector



(c) 18^{th} dimension of state vector



(d) 19^{th} dimension of state vector

Figure 3.8: Training data distributions on Pusher environment

the complete episode length meaning that model prediction error on environments with sparse reward functions are shown on 1-step predictions, on 25-step predictions and on H-steps, H being the task horizon. They are obtained using TS_{inf} on the two Two-Wheeled Robot Maze environments. We consider a single initialization budget of 10 episodes as it represents a reasonable number of environment interaction *w.r.t.* the required number of episodes for task completion for all initial data gathering methods.

Raw prediction error (mean and standard deviation) results for the Cartpole,

Cartpole					
Prediction horizon	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
1	**0.18 ± 0.44**	0.33 ± 0.58	0.39 ± 0.63	0.39 ± 0.63	0.26 ± 0.5
25	**1.15 ± 1.53**	2.39 ± 5.39	2.5 ± 2.54	3.35 ± 2.51	1.55 ± 1.86

Table 3.1: Mean prediction error on Cartpole

Cartpole					
	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
RA	1.000e+00	8.357e-04	9.786e-03	7.113e-03	9.786e-03
$CNRW_0$	8.357e-04	1.000e+00	4.903e-01	2.853e-01	1.047e-06
$CNRW_1$	9.786e-03	4.903e-01	1.000e+00	9.246e-01	1.047e-06
$CNRW_2$	7.113e-03	2.853e-01	9.246e-01	1.000e+00	1.576e-06
RP	9.786e-03	1.047e-06	1.047e-06	1.576e-06	1.000e+00

Table 3.2: Mann-Whitney U Test p-values over 25-step prediction errors on Cartpole

Pusher, Ball In Cup, Two-Wheeled Robot Maze and Two-Wheeled Robot Maze Traps environment are respectively shown in Tables 3.1, 3.6, 3.3, 3.8 and 3.10. Moreover, Mann-Withney U Test [104] p-values tables are also given for each environment appropriate prediction horizon in Tables 3.2, 3.4, 3.5, 3.9, 3.11 and 3.7. The Mann-Whitney p-values are obtained by computing the U Test for each of the given initialization method prediction error against each of the other initialization method prediction error. We will only refer to both tables to interpret more in depth results from Figures 3.9a and 3.9b, which show the model prediction error for each of the initialization methods considered depending on the environments, ranked by their consistency measure from high consistency to low consistency. Model prediction errors are normalized between the minimum and maximum prediction error obtained with the bootstrapped models. Ideally, if our hypothesis is correct, we should see a downtrend from left to right for methods with high time-correlation and an uptrend from left to right for methods with low time-correlation in their generated action sequences.

Figure 3.9a shows the 1-step mean prediction error measured for the models trained using the data gathered by the initialization methods. We observe on the Cartpole environment that the least time-correlated method, Random Actions, is

Ball In Cup					
Prediction horizon	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
1	0.058 ± 0.107	0.037 ± 0.066	0.022 ± 0.045	**0.021 ± 0.044**	0.024 ± 0.048
25	0.796 ± 1.322	0.681 ± 1.116	0.346 ± 0.561	**0.327 ± 0.515**	0.416 ± 0.682
300	15.00 ± 45.40	14.90 ± 22.20	3.400 ± 2.800	3.600 ± 1.900	**3.100 ± 1.300**

Table 3.3: Mean prediction error on Ball In Cup

Ball In Cup					
	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
RA	1.000e+00	2.139e-03	6.796e-08	6.776e-08	1.235e-07
$CNRW_0$	2.139e-03	1.000e+00	6.796e-08	6.776e-08	2.218e-07
$CNRW_1$	6.796e-08	6.796e-08	1.000e+00	5.978e-01	1.895e-01
$CNRW_2$	6.776e-08	6.776e-08	5.978e-01	1.000e+00	3.850e-02
RP	1.235e-07	2.218e-07	1.895e-01	3.850e-02	1.000e+00

Table 3.4: Mann-Whitney U Test p-values over 25-step prediction errors on Ball In Cup

Ball In Cup					
	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
RA	1.000e+00	1.988e-01	4.679e-02	2.137e-03	1.636e-01
$CNRW_0$	1.988e-01	1.000e+00	4.166e-05	3.063e-06	2.341e-03
$CNRW_1$	4.679e-02	4.166e-05	1.000e+00	2.563e-02	6.949e-01
$CNRW_2$	2.137e-03	3.063e-06	2.563e-02	1.000e+00	2.943e-02
RP	1.636e-01	2.341e-03	6.949e-01	2.943e-02	1.000e+00

Table 3.5: Mann-Whitney U Test p-values over H-step prediction errors on Ball In Cup

the most precise. The Cartpole environment being the most consistent one, we did expect to observe a gradient of prediction errors from the least time-correlated to the highest time-correlated initialization methods. We also remark that $CNRW_1$ and $CNRW_2$ are very close in terms of prediction errors, both having the same mean and std prediction error. Random actions prediction error is more than twice lower, thus we expect to see a better performance of PETS on the cartpole environment with a model bootstrapped with Random Actions. Moreover, Table 3.2 shows that Random Actions are significantly different from all others initialization methods. Random Policies also perform pretty well, sitting in-between the least to the highest initial data gathering methods.

At first glance, the Pusher environment has a strange order not corresponding to our hypothesis. But when looking more closely at Table 3.6 we observe that all prediction errors are actually really close from each other, which is confirmed by looking at Table 3.7, where no statistically significant difference is found. We thus would expect all methods to impact equally the PETS algorithm. As expected, the Two-Wheeled Robot Maze Traps environment is harder to interpret. Indeed, both

Pusher					
Prediction horizon	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
1	**0.2 ± 0.63**	0.22 ± 0.64	0.21 ± 0.64	0.22 ± 0.64	0.21 ± 0.64
25	1.69 ± 2.13	1.41 ± 1.99	**1.28 ± 1.79**	1.28 ± 1.81	1.28 ± 1.88

Table 3.6: Mean prediction error on Pusher

Random Actions, $CNRW_1$ and $CNRW_2$ perform almost equally if we only look at the mean prediction error. Nevertheless, Random Actions initialized models actually have a lower standard deviation on this prediction horizon as shown on Table 3.10, which could indicate that Random Actions would be the better method to use for this environment. Moreover, Table 3.11 shows on the 1000 steps prediction horizon that Random Actions are significantly higher from all others initialization methods.

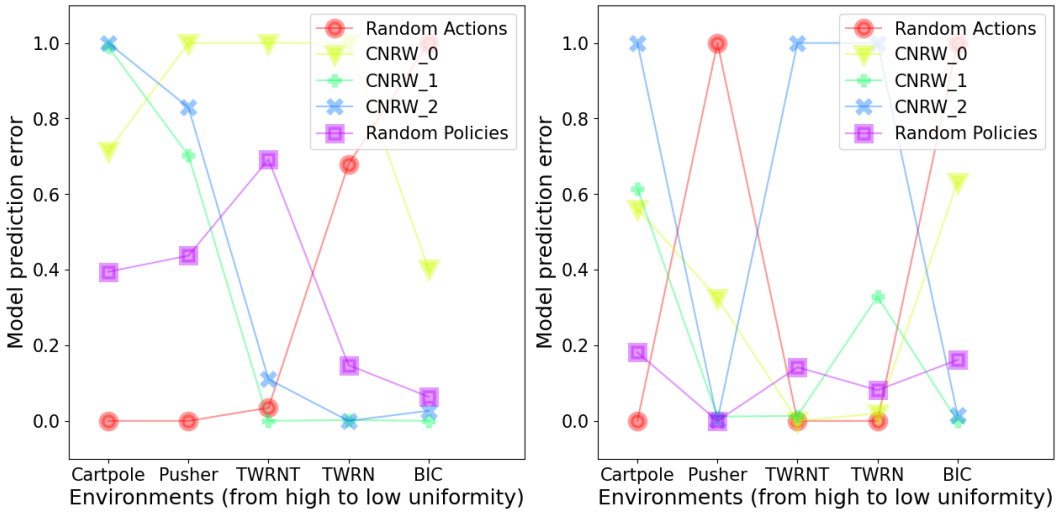
The Two-Wheeled Robot Maze environment is also hard to interpret, as Random Actions is the second worst in terms of mean prediction error. Nevertheless, once again the standard error is 4 times lower for Random Actions than for $CNRW_1$, $CNRW_2$ and Random Policies, even though these methods give the lowest mean prediction error. Since errors are compounded, a high variance in the predictions can lead to highly diverging trajectories compared to a higher mean prediction error with a lower variance. Looking at this data, it is still unclear which method would be best to initialize the model for that task, as no clear statistical significance is found between the initialization methods prediction errors on the 1000 step prediction horizon as shown in Table 3.9.

Finally, on the Ball In Cup environment, higher time-correlated methods show the smallest prediction error. Indeed, $CNRW_1$ and $CNRW_2$ have a mean and standard deviation of prediction error that is almost three times smaller than that of Ran-

Pusher					
	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
RA	1.000e+00	1.636e-01	2.139e-03	3.851e-02	3.507e-01
$CNRW_0$	1.636e-01	1.000e+00	7.712e-03	9.786e-03	6.389e-02
$CNRW_1$	2.139e-03	7.712e-03	1.000e+00	9.461e-01	1.478e-01
$CNRW_2$	3.851e-02	9.786e-03	9.461e-01	1.000e+00	2.853e-01
RP	3.507e-01	6.389e-02	1.478e-01	2.853e-01	1.000e+00

Table 3.7: Mann-Whitney U Test p-values over 25-step prediction errors on Pusher

dom Actions. Random Policies also perform pretty well, once again having a mean prediction error quite close to the best prediction error observed. Table 3.4 shows the p-values of the Mann-Whitney U test for prediction errors at 25 steps prediction horizon. On that horizon, higher time correlated methods ($CNRW_1$ and $CNRW_2$) are significantly different from lower time correlated methods (RA and $CNRW_0$). Nevertheless, Table 3.5 shows that on a higher prediction horizon, the significance of the prediction errors differences diminishes by several orders of magnitude.



(a) 1-step

(b) 25-step

Figure 3.9: Min-max normalized model prediction error depending on the environment consistency measure

We will now look at the 25-step prediction error as it is the one that interests us the most, since its aligned on PETS optimizer planning horizon. On the same graph we also plot the H-step prediction error for the two Two-Wheeled Robot Maze environments, as this is the planning horizon of interest in the case of DAQD. The Ball In Cup environment prediction error is shown on the 25-step horizon. This horizon was chosen as it should compound in a similar way whichever the horizon is, given the fact that the variance in the prediction error is not too important as shown in Table 3.3. This graph, on Figure 3.9b could thus directly predict the initial PETS and DAQD algorithm performance depending on the initialization method used to gather training data.

Overall, we see that the model prediction error order remain the same as for 1-step prediction error on the Cartpole and Ball In Cup environments, except for the pusher environment, where more time-correlated methods have the least mean prediction error now. Nevertheless, looking more into detail on Table 3.6, we observe once again

that most methods are pretty close one to another in terms of prediction error, even though Random Actions have the worse prediction error. On the two Two-Wheeled Robot Maze environments, Random Actions now have the lowest prediction error as expected from the 1-step prediction errors observed in detail on the Table 3.8 and 3.10. Indeed, Random Policies, $CNRW_1$ and $CNRW_2$ mean prediction errors explode on the long prediction horizon of 1000 steps and completely deviate from the true test trajectories. We can thus conclude that Random Actions should initialize the model at best and give the best results for the DAQD algorithm.

The Ball In Cup and Cartpole environments display a prediction error two to three times lower with reduced standard deviation respectively for higher time-correlated methods and lower time-correlated methods as expected. Overall, we observe that our hypothesis that the least time-correlated methods would perform best on more consistent environments seems to be validated when considering the environments we experimented on. Results are more difficult to interpret when considering the three intermediary environments, Pusher, Two-Wheeled Robot Maze and Two-Wheeled Robot Maze Traps, as it seems for the Pusher that all methods shall perform equally, while on the Maze environments it seems that Random Actions should give the best initialized model in terms of mean prediction error for the planning horizon of the algorithm. It will now be interesting to observe the actual impact of these different model prediction errors on the PETS and DAQD algorithms, as we would expect the performance of the algorithm to follow that of the models.

3.2.3 Model-Based Policy Search on a Learned Model

In this section, we study the impact of the initial data gathering technique chosen on two different Model-Based Policy Search algorithms: an episode-based one, DAQD [98] and a step-based one, PETS [26]. Consequently to what was shown in the previous sections, we expect the Model-Based Policy Search algorithms performance to be increased on environments with high consistency when initializing the model

Two-wheeled robot maze					
Prediction horizon	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
1	**0.152 ± 0.249**	0.207 ± 0.480	0.036 ± 1.040	0.035 ± 1.045	0.061 ± 0.986
25	**1.698 ± 3.277**	3.433 ± 6.600	0.921 ± 13.15	0.930 ± 14.246	1.411 ± 10.85
1000	**291.8 ± 165.4**	460.6 ± 501.5	3454 ± 6112	8994 ± 5357	1026 ± 1313

Table 3.8: Mean prediction error on Two-wheeled robot maze

Two-Wheeled Robot Maze					
	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
RA	1.000e+00	3.942e-01	2.748e-02	9.780e-03	1.058e-02
$CNRW_0$	3.942e-01	1.000e+00	7.643e-02	5.649e-02	8.103e-02
$CNRW_1$	2.748e-02	7.643e-02	1.000e+00	3.234e-01	6.168e-01
$CNRW_2$	9.780e-03	5.649e-02	3.234e-01	1.000e+00	2.503e-01
RP	1.058e-02	8.103e-02	6.168e-01	2.503e-01	1.000e+00

Table 3.9: Mann-Whitney U Test p-values over H-step prediction errors on Two-Wheeled Robot Maze

Two-wheeled robot maze trap field					
Prediction horizon	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
1	**0.133 ± 0.677**	0.307 ± 1.585	0.133 ± 4.032	0.146 ± 4.043	0.227 ± 3.901
25	**1.452 ± 2.784**	4.207 ± 7.751	2.514 ± 51.47	2.820 ± 56.82	4.060 ± 48.84
1000	**118.6 ± 55.7**	281.4 ± 135.4	7977 ± 5459	9183 ± 9597	8658 ± 9418

Table 3.10: Mean prediction error on Two-wheeled robot maze trap field

Two-Wheeled Robot Maze Traps					
	RA	$CNRW_0$	$CNRW_1$	$CNRW_2$	RP
RA	1.000e+00	2.596e-05	1.037e-04	2.684e-06	2.563e-07
$CNRW_0$	2.596e-05	1.000e+00	1.075e-01	1.547e-02	6.040e-03
$CNRW_1$	1.037e-04	1.075e-01	1.000e+00	1.404e-01	2.977e-01
$CNRW_2$	2.684e-06	1.547e-02	1.404e-01	1.000e+00	7.557e-01
RP	2.563e-07	6.040e-03	2.977e-01	7.557e-01	1.000e+00

Table 3.11: Mann-Whitney U Test p-values over H-step prediction errors on Two-Wheeled Robot Maze Traps

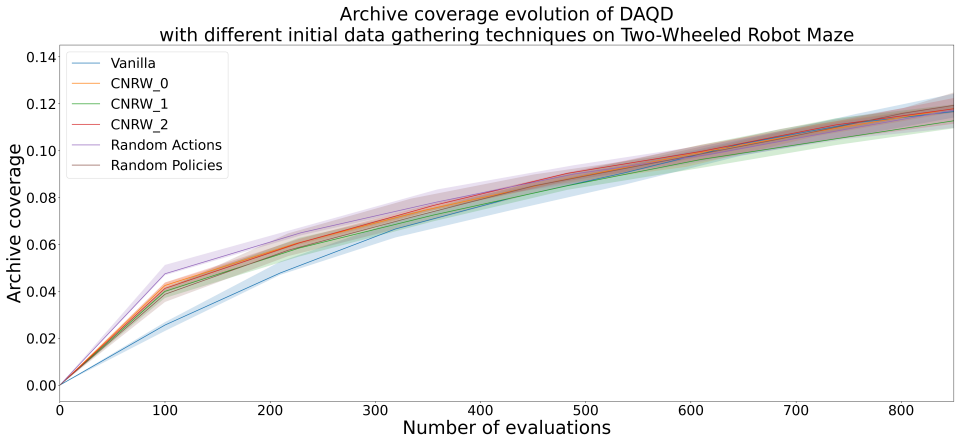
with lesser time-correlated initial data gathering techniques. The same reasoning goes for environments with low consistency, and intermediary environments should not advantage much any initialization technique.

3.2.3.1 Impact on an episode-based Model-Based Policy Search algorithm: DAQD

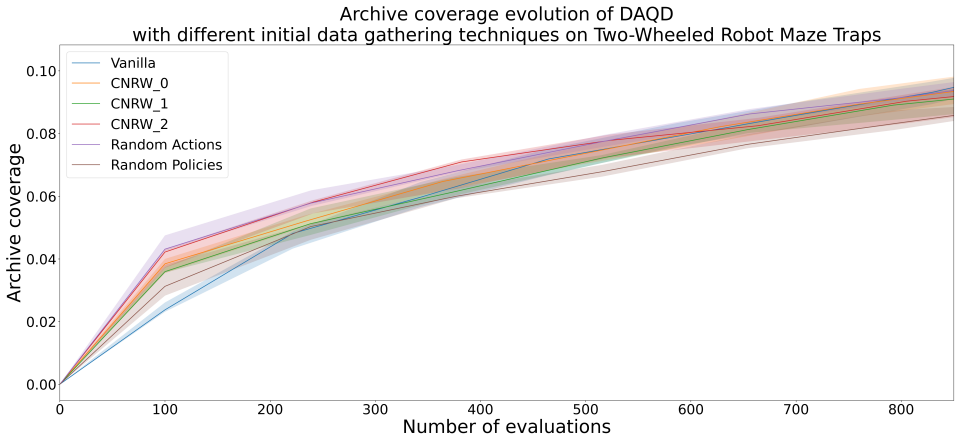
Firstly, we study the impact of the different initial data gathering methods on DAQD [98]. Two different version of the algorihtm are compared: the *Vanilla* version, which bootstraps the model and the archive with 100 randomly parameterized policies, and a second version directly using the model at the first iteration after bootstrapping it with one of the initial data gathering methods compared in this chapter. We expect DAQD to have a lower initial coverage when model prediction error is higher, as a model with a higher prediction error shouldn't be able to predict as accurately as a model with lower prediction error the behavior descriptor of a candidate solution.

Figures 3.10a, 3.10b and 3.10c show the evolution of the coverage using either Vanilla DAQD or the bootstrapped version of DAQD. Looking first at Figure 3.10a, we observe that Random Actions perform slightly better than the other initialization methods on the first generation, with a very slight 20% increase in coverage. This seems to corroborate what we concluded in the previous section, with Random Actions having a lower model prediction error. All other initial data gathering methods bring a comparable coverage. This results is consistent with the model prediction errors we evaluated, as only the models initialized with Random Actions kept their mean and standard deviation on prediction error in a reasonable margin on the task horizon. The Ball In Cup environment results on DAQD do not show any difference whichever method is used, even though a pretty important gap was found in terms of model prediction error. This can be due to the fact that solutions are evaluated episodically, which bias them toward important time-correlation given the policy representation, and that whichever the model prediction error is.

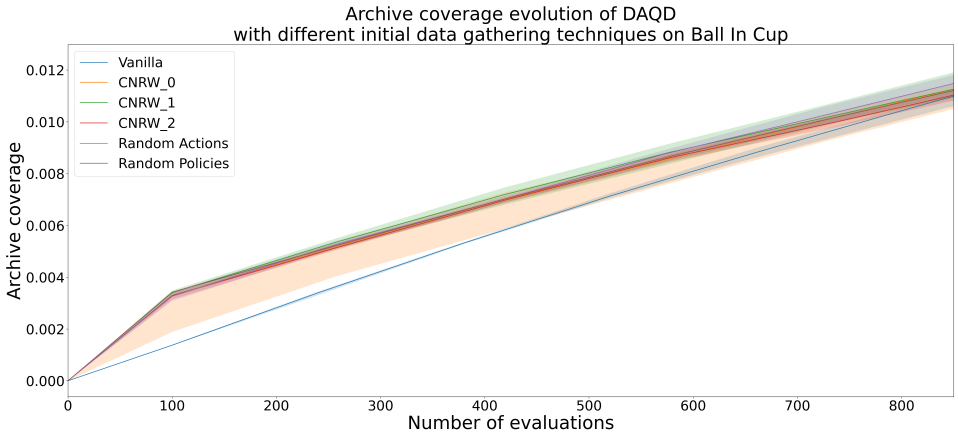
Nevertheless, even if we observe that all the initialization methods perform better than the Vanilla DAQD approach, at least on the first generation, the effect of the model bootstrapping is lost after a few generations. Choosing a particular way to gather initial data for model training in DAQD on the two-wheeled robot maze environment does not seem to impact the policy search approach. As DAQD gathers much more data than the fixed training budget, the effect of the pre-training is not so consequent and show the robustness of DAQD against various model initialization techniques.



(a) Two-Wheeled Robot Maze

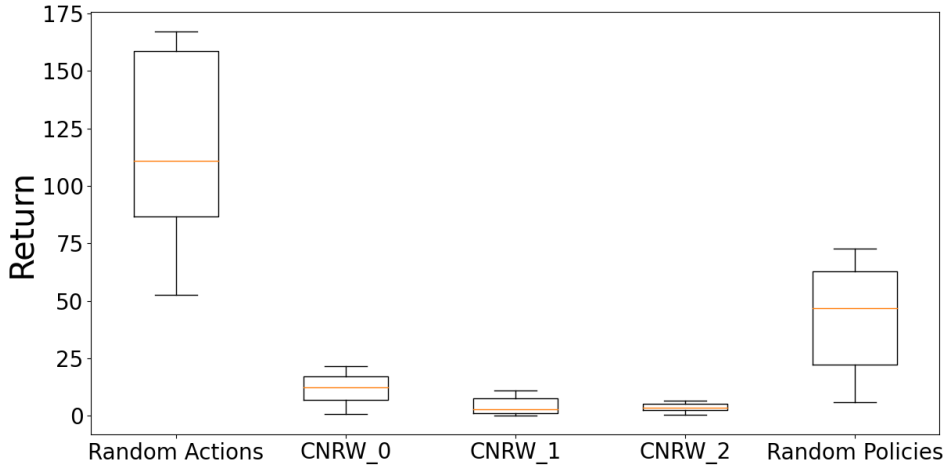


(b) Two-Wheeled Robot Maze Traps

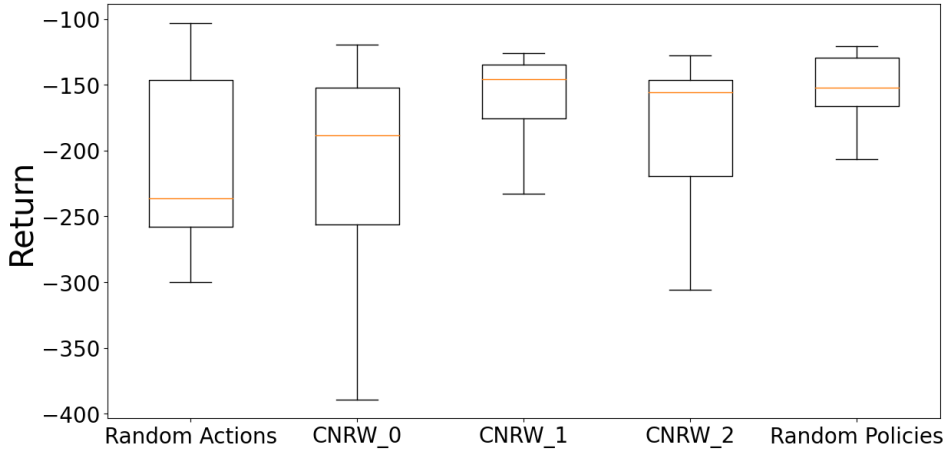


(c) Ball In Cup

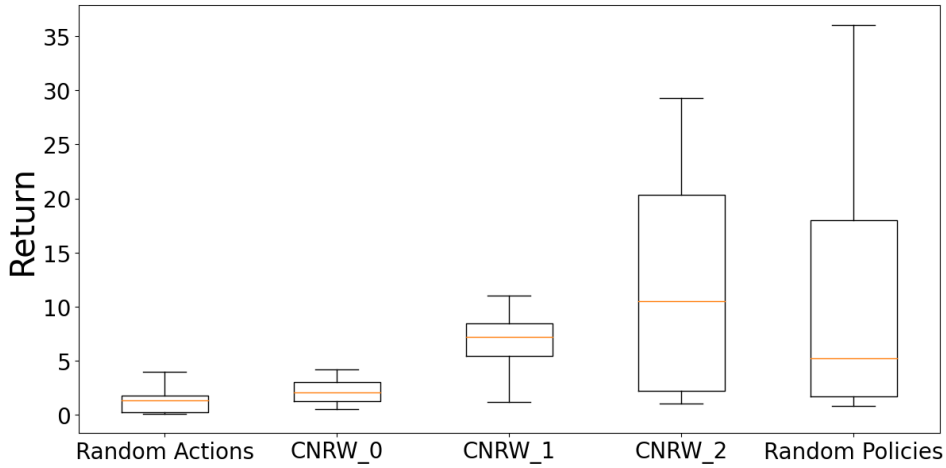
Figure 3.10: Impact of different initial data gathering methods on DAQD algorithm ten first generations



(a) Cartpole



(b) Pusher



(c) Ball In Cup

Figure 3.11: Impact of different initial data gathering methods on PETS algorithm first iteration return

3.2.3.2 Impact on a step-based Model-Based Policy Search algorithm: PETS

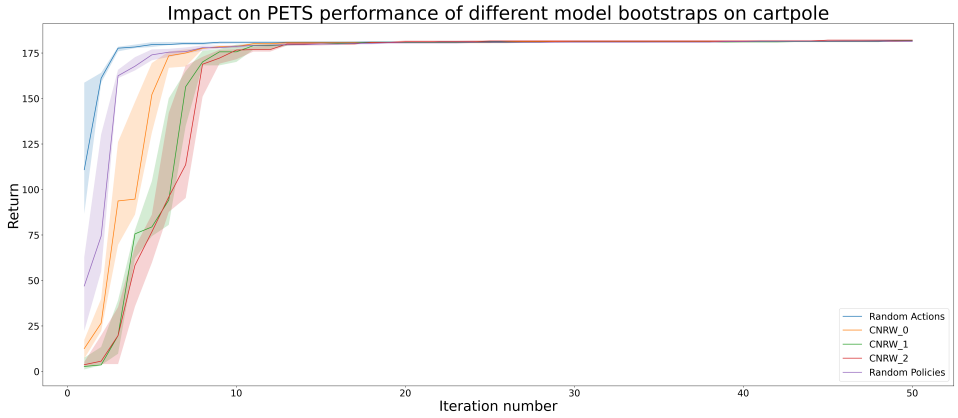
PETS [26] is the method that is used for demonstrating the impact of various initial data gathering methods. We use TS_1 particle propagation together with models

pre-trained with the different initial training data we mentioned. It is expected that PETS will yield a higher initial return when model prediction error is lower. Consequently, we expect PETS performance to be increased on environments with high consistency when initializing the model with lesser time-correlated initial data gathering techniques. The same reasoning goes for environments with low consistency, and intermediary environments should not advantage much any initialization technique.

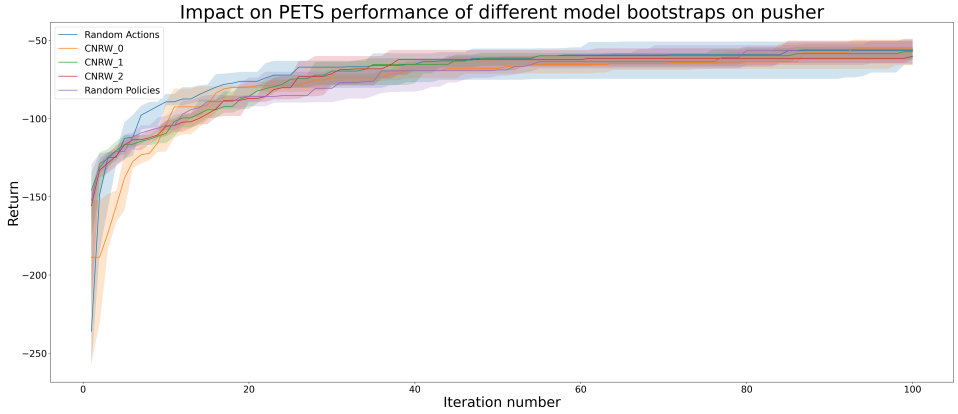
Figures 3.11a, 3.11b and 3.11c show the first iteration return obtained using PETS with different model bootstraps. Looking first at Figure 3.11a, we observe that the model prediction error order we observed on Figures 3.9a and 3.9b is conserved. Indeed, Random Actions perform 10 times better than lesser time-correlated methods on the first iteration. Similarly on the Ball In Cup environment, higher time-correlated initialization methods initially bring a return 4 to 6 times higher than lesser time-correlated initialization methods. Finally, all initial data gathering methods on the Pusher environment yield comparable initial return. It is worth noting that we observe on all these environments that the model prediction error shown in Figure 3.9b directly correlates with the model-based policy search performance, with an increasing proportion *w.r.t.* actual model prediction error. Figures 3.12a, 3.12b and 3.12c show the evolution of the return obtained by PETS over multiple algorithm iterations, with the model being trained after each iteration. Figure 3.12a shows that bootstrapping the model using Random Actions on the Cartpole task helps the algorithm converge after only 3 iterations, while it takes more than 10 to converge when bootstrapped with $CNRW_1$ or $CNRW_2$. Similarly, Figure 3.12c shows that boostrapping the model using Random Actions leads to almost no reward being found on the Ball In Cup task, while bootstrapping it with others methods helps lot more, yielding returns more than 5 times higher. Finally, Figure 3.12b shows what was observed on the first iteration with all methods more or less performing the same over iterations.

3.3 Conclusions

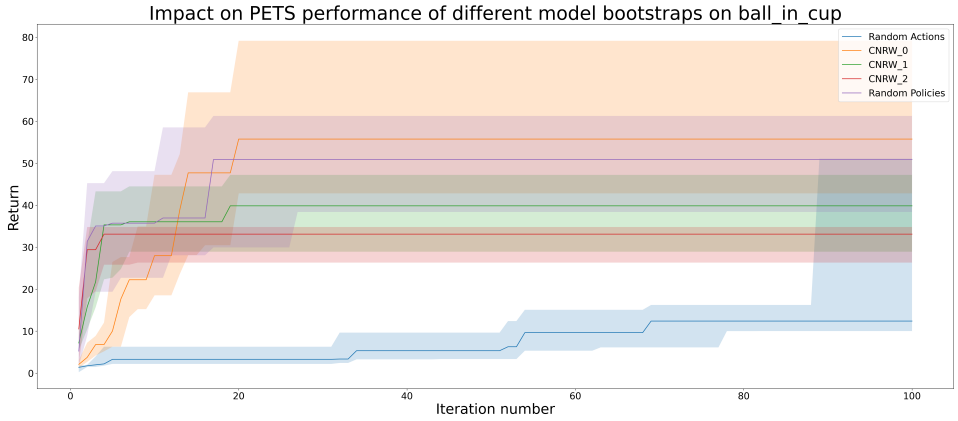
In this chapter, we proposed a consistency metric based on the coefficient of variation that measures the action consistency in a Markovian Decision Process. A link between the consistency and the dynamics model prediction error when initialized with a suited initial data gathering method has been made, as we empirically evaluated these on five environments of varying consistency. Clear conclusions cannot be



(a) Cartpole



(b) Pusher



(c) Ball In Cup

Figure 3.12: Impact of different initial data gathering methods on PETS algorithm return over evaluations

drawn from the two Two-Wheeled Robot Maze environments and from the Pusher environment. However, the two more extreme environments in terms of consistency, the Ball In Cup and the Cartpole, do seem to go in the direction that the proposed metric seems to characterize the environment dynamics to some extent. It was also shown that time-correlation in the data gathering action sequences can be selected

depending on the consistency measure, so as to minimize model prediction error.

We looked at the impact of the initial model prediction error and the evolution of the coverage using the DAQD algorithm, but aside from a small difference at the first iteration, no clear impact on the algorithm performance in terms of coverage can be drawn as methods that yielded a lower coverage quickly catch up with the better ones. This can be due to the high budget given to the DAQD algorithm compared to the considered initialization budget, as well as being due to the novelty pressure coming from the archive itself, which gets filled pretty well even with randomly parameterized policies as shown with the Vanilla DAQD version.

Finally, we saw evidence of a direct link between the initial model prediction error and the initial return with the PETS algorithm, with obtained return differences being of a factor of up to 10 for the two extremes of the time-correlation in the initializing action sequences. Even though the results are pretty clear for the first iteration of the Model-Based Policy Search algorithm, it is worth noting that during our experiments we also saw inversions in the return trends over the course of the policy search algorithm training. Such dynamics are inherent to the method used, and could be a subject of further studies.

Chapter 4

Archive Bootstrapping For Sample-Efficiency

As seen in the previous chapter on learned model bootstrapping, initialization can be a crucial phase of any learning method, for example when applied to regression problems like learning a robotic system dynamics. Indeed, from our conclusions on learned model initialization in the previous chapter 3, we saw that properly bootstrapped models could lead to an important increase in performance of a subsequent model-based policy search algorithm. Nevertheless, it was still a requirement to interact with the real system to gather data so as to bootstrap the model, but could it be possible to initialize a random model that would pertain a similar dynamics than that of a low data regime bootstrapped model? If so, could it be possible to use such a random dynamics model to determine beforehand a diverse collection of policies to bootstrap a population-based algorithm?

To that end, we propose in this chapter to explore various random dynamics model representations and various policy representations together with a model-based Novelty Search loop ran on the random model, whose aim is to find generalist or evolvable individuals to bootstrap the initial population of a subsequent Novelty Search ran on the real system. The main study subject will thus be two-fold. Firstly, the initial coverage of the Outcome Space $\mathcal{B}$ of the population generated using the random dynamics model shall be higher than that of a randomly parameterized population. Secondly, the subsequent Novelty Search algorithm bootstrapped using that population as a starting point should cover the Outcome Space in a lower number of evaluations compared to a Novelty Search routine bootstrapped using randomly parameterized policies as its initial population. To create a selective pressure towards

individuals that are novel on a range of environments rather than on a single one, we propose to use ensemble of random dynamics models and to measure an individual novelty across the model ensemble, following various techniques detailed furthermore in the next section.

4.1 Methods

Our objective is thus to create an algorithm that replaces the initial population in Novelty Search from randomly parameterized policies to a population biased towards being more novel initially, resulting in a higher coverage, and being more evolvable, resulting in a faster coverage of the environment. The proposed method thus works around three axis:

- Representing random dynamics
- Searching for a diverse set of policies on the random dynamics ensembles
- Selecting policies for initialization

One thus asks firstly, how to represent random dynamics? The first concern doing so should be that, much like it is done with System Identification, the random dynamics should be as close as possible to those of the real system. The problem is that the dynamics of the target system are considered unknown. Considering models with close to unlimited representational capacity could thus be interesting. Non-linear neural networks are one model representation that fits this. Nevertheless, using randomly parameterized non-linear neural networks as models could lead to dynamics that are not smooth. Indeed, there is no guarantee that the state transition given by a Non-Linear Neural Network to a specific state-action pair will give similar state transition values even if changing very slightly the state or the action. In case using Non-Linear Neural Networks is too unstable, we propose to compare them to a simpler random dynamics model representation. We thus propose to use Spatial Random Fields [60] for that purpose.

More precisely, we consider using Gaussian Spatial Random Fields, which model spatial correlations in the considered variables. In our case, the variables is the state of the dynamical system, using such models thus enforces that each of the state variables evolve independently from one another and according to a Gaussian law, moreover output values for each state variable are correlated, meaning that the

output values do not abruptly change if two input state-action pairs are close from one another. This assumption simplify vastly the dynamics of a random dynamics model, potentially making it closer to reality in case of simple real system dynamics, or making it farther if the target system dynamics contain non-linearities. Nevertheless, whether using Non-Linear Neural Networks or Spatial Random Fields, using a single random dynamics model will not help in finding policies that are novel on several dynamical systems, most importantly the real target system. Indeed, when using a single random dynamics model, the real target dynamics could be too far from the source training domain and lead to poor results when transferring the learned behaviors.

One solution to mitigate this could be to use ensembles of random dynamics models, much like in Domain Randomization [156]. Indeed, using an ensemble of random dynamics models could help generate a collection of individuals whose diversity can be robust to various dynamical systems, even if non-linearities exists. However, this raises the question of how to search for such collection of individuals on the model ensemble. When looking for a diverse set of individuals on a single dynamical system *w.r.t.* a user-defined Behavioral Space $\mathcal{B}$, the Vanilla Novelty Search algorithm can be followed. However, when searching on multiple dynamical system at once, the strategy to maintain a diverse collection of individuals should be adapted.

We propose two ways of doing so:

- Computing the sum of the novelty on each of the $p_{rand,dyn}$ such that the novelty of individual x is $N_{sum}(b_x) = \sum_{i=1}^{R} N(b_x)_i^{p_{rand,dyn}}$
- Computing the minimum of the novelty on each of the $p_{rand,dyn}$ such that the novelty of individual x is $N_{min}(b_x) = min_{1,...,R}^{p_{rand,dyn}}(N(b_x))$

Using the classical Novelty Search algorithm, we thus propose to evaluate the considered individuals on each of the R random dynamics model and to use one of the two above novelty metrics, N_{min} and N_{sum}, to estimate the novelty of the individuals and otherwise proceed like usual with the Novelty Search algorithm optimization loop. As the models are random and do not represent directly real system dynamics, the prediction horizon does not have to be the actual episode length of the real system. Using the real episode length as prediction horizon could potentially bring the proposed individuals closer to the real system dynamics, but it could also increase the risk of divergence as the random dynamics models are unbounded. To limit that effect and enhance greatly the computation speed, we limit the prediction

horizon on the random dynamics models to horizons ranging between 10 and 100 steps, representing a fraction of the real system episode length.

Finally, once Novelty Search on the random dynamics model ensemble is done, it is required to select some individuals among all those evaluated to be transferred onto the real system as a bootstrap for a Novelty Search routine. To do so, we propose to explore two different selection methods:

- Selecting the final population, as it should be the most evolvable [44] thus potentially the bootstrap that should allow for the fastest evolution towards uniform Outcome Space coverage

- Selecting the most novel individuals could ensure that the transferred individuals are novel enough to generate an initial coverage that is higher of that of randomly parameterized policies, but only if the source random dynamical system distribution is close enough or contains the target dynamical system

We decide to rule out using random selection as the number of individuals generated on the random dynamics models is almost 100 times that of the individuals we select for transfer on the real system. Using random selection would thus sub-sample a lot from the generated solutions and not be representative of the policy search made on the random dynamics model ensemble. For more details, the general algorithm for our algorithm, $0DAB$ is outlined in 6.

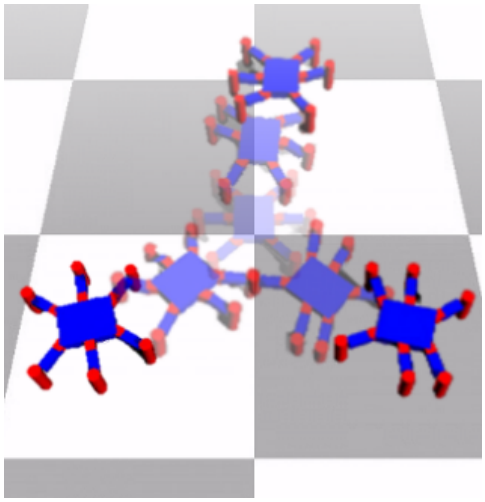
Algorithm 6 0DAB Algorithm

Initialize repertoire $\mathcal{A}_\Pi$ (to $\emptyset$), estimated repertoire $\widetilde{\mathcal{A}_\Pi}$ (to $\emptyset$), dynamics model $p_{rand,dyn}$, k (to 15)

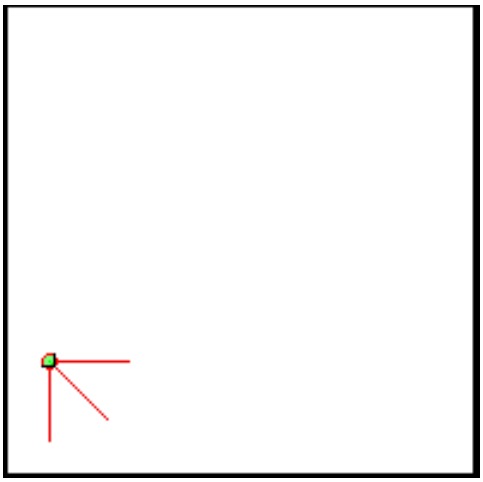
$\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*} \leftarrow$ random_parameters()
▷ *Performing NS Exploration on the model* $p_{rand,dyn}$
while evaluation budget is not exhausted **do**
    $\widetilde{b}_{i \in [1,b]} \leftarrow$ model_evaluate($\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*}$) using $p_{rand,dyn}$ ▷ *Evaluate using model. Rollouts up to horizon H on* $p_{rand,dyn}$ *to estimate* $\widetilde{b}_i$.
    $\widetilde{\mathcal{A}_\Pi} \leftarrow$ novelty_selection($\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*}, \widetilde{b}_1 \dots \widetilde{b}_b, \widetilde{\mathcal{A}_\Pi}$) ▷ *Add the k most novel* $\pi_{\boldsymbol{\theta}_i^*}$ *to* $\widetilde{\mathcal{A}_\Pi}$ *depending on estimated novelty from* $\widetilde{b}_i$ *(estimated using* N_{min} *or* N_{max}.
    $\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_b} \leftarrow$ select($\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*}$) ▷ *Selecting b most novel policies from offspring.*
    $\pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*} \leftarrow$ variation_operators($\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_b}$)

bootstrap $\leftarrow$ *select_bootstrap*($\widetilde{\mathcal{A}_\Pi}, \pi_{\boldsymbol{\theta}_1^*} \dots \pi_{\boldsymbol{\theta}_b^*}$)

return *bootstrap*



(a) Omnidirectional Hexapod Robot Locomotion



(b) 2-Wheeled Robot Navigation

Figure 4.1: Considered Environments

Nomenclature examples					
Model Type	Ensemble Size	Prediction Horizon on Model	Novelty Metric on Ensemble	Selection Method	Nomenclature
Non-Linear Neural Network	4	10	N_{sum}	Final Population	NN-4-h10-sum-fp
Spatial Random Field	40	10	N_{min}	Novelty	SRF-40-h10-min-nov

Table 4.1: Nomenclature examples for different 0DAB parameterizations

4.2 Experiments

To denote all of these different parameterizations for 0DAB, a nomenclature is defined and examples are provided in in Table 4.1

4.2.1 Experimental Setups

To assess the performance of the proposed bootstrapping algorithm, experiments are conducted on two environments. In the case of bootstrapping Novelty Search, a single metric interests us: coverage of the Outcome Space $\mathcal{B}$. Firstly, initial coverage attained with the output from the proposed bootstrap algorithm is compared to that of randomly parameterized policies, the bootstrapping method commonly used in the literature. Secondly, the coverage evolution depending on the number of real-world evaluations should demonstrate if the bootstrap has a longer term effect over the whole Novelty Search routine. Those two graphs should help us determine which

type of random dynamics model can be interesting to use in the case of robotics system of varying dynamics complexity, which bootstrap selection method is the most robust and finally if bootstrapping the model with such a technique has an impact on a subsequent Novelty Search routine. Real-system Novelty Search routines are systematically bootstrapped with 100 individuals, this amount being equivalent to the population size s_p. 0DAB bootstraps are therefore equivalent to initializing the population of Novelty Search with the output of the 0DAB algorithm rather than using randomly parameterized policies.

Two robotic environments are considered, which are a Two-Wheeled Robot Navigation task and an Omnidirectional Hexapod Locomotion task. The Two-Wheeled Robot Navigation task consists in a khepera-like mobile robot navigating a (vast) empty space over an episode with a long horizon of 1000 steps. Each robot wheel is independently speed controlled, making this environment Action Space two dimensional. The system state consists of the robot position, velocity and orientation, the latter being divided into its cosine and sine parts for continuity reasons. The Two-Wheeled Robot Navigation task thus has a State Space of dimension 6. In this environment, the observer function $o_{\mathcal{B}}$ maps each individual trajectory to its final x-axis and y-axis position, making the behavioral descriptor the final position of the robot at the end of the episode. An image of the setup is shown on Figure 4.1b. The simulator used for this task is the Fastsim simulator [119]. The dynamics of this environment are almost completely smooth, as they follow the kinematics of a differential drive robot. Non-linearities arise on the outskirts of the navigation map, as there are walls that the robot can collide with. This environment dynamics being pretty smooth and uniform across the whole State-Space, random dynamics model ensembles generated individuals could maintain their diversity when transferred on this environment.

The second environment considered is an Hexapod Robot Locomotion task [98]. It consists in an 18-DoF hexapod robot whose goal is to learn locomotion skills as shown on Figure 4.1a. An episode lasts 300 time-steps. The Hexapod has 3 controllable joints per leg, for a total of 18 independently controlled joints, making the Action Space of this task a 18 dimensional vector. The State-Space is comprised of 48 dimensions, being the current angular position of the joints, their angular velocity, the Cartesian position and velocity of the robot center of mass and the angular position and velocity of the robot center of mass. The considered Outcome Space in this task is the final x-axis and y-axis position reached by the robot at the end of the episode. This task uses the DART simulator proposed by Lee *et al.* [90]. This task has much more complex dynamics than that of the Two-Wheeled Robot

Navigation task, as it has many non-linearities in its transition function with the legs hitting the ground and allowing the robot to develop a walking gait using all six of its legs. This environment dynamics being of high dimension and non-linear in many regions of the State-Space, random dynamics model ensembles generated individuals might not be able to maintain their diversity when transferred on this environment.

4.2.2 Two-Wheeled Robot Navigation Task: Spatial Random Fields or Neural Networks as Random Dynamics Models

The Two-Wheeled Robot Navigation Task having the smoothest dynamics out of the two considered environments, it is insightful to firstly look at this environment to determine what random dynamics model representation and what bootstrap selection method are the most suited to accomplish our objectives. As such, initial coverages of the Outcome Space obtained with various model representations are observed in this section. Four different random dynamics model representations are considered: Spatial Random Fields ensembles of size 4 and 40, and Non-Linear Neural Networks ensembles of size 4 and 40, all with an evaluation prediction horizon of 10 steps on the Novelty Search routine on the random models ensembles. Coverages are plotted as boxplots, and are sorted in increasing order using the median coverage value of each bootstrapping method considered.

Figure 4.2 shows that Spatial Random Fields ensembles of size 4 fail to outperform randomly parameterized policies in terms of initial coverage. Indeed, Spatial Random Fields ensembles of size 4 with 0DAB generate individuals whose coverage is inferior or equivalent to that of randomly parameterized policies. We can thus rule out using such random dynamics models, as they do not seem to be able to generate diverse solutions even for the smoother dynamics of the Two-Wheeled Robot Navigation Task. On the contrary, Figure 4.3, which shows the initial coverage obtained with bootstraps outputs from 0DAB with Non-Linear Neural Networks ensembles of size 4, demonstrates that such random dynamics models can produce diverse solutions for the dynamics of the considered environment to some extent, such that Random Policies are outperformed in terms of initial coverage by 30%. Non-Linear Neural Networks ensembles thus seem to be an interesting random dynamics model representation methods for the robotic task considered that we will further study in the next section.

Even though Spatial Random Fields ensembles of size 4 did not yield interesting results, it might be due to the relatively small size of the ensemble considered. Figure

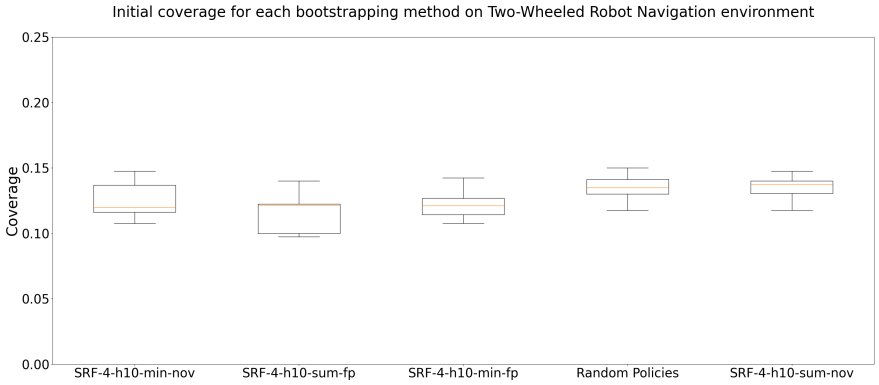


Figure 4.2: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Spatial Random Fields ensembles of size 4 on Two-Wheeled Robot Navigation Task

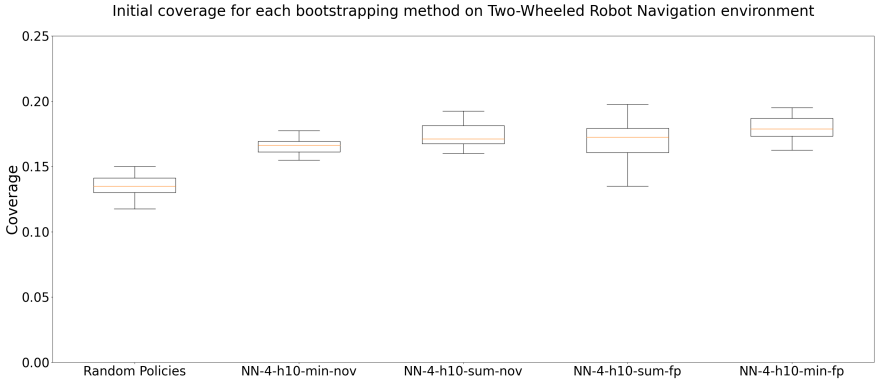


Figure 4.3: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Non-Linear Neural Network ensembles of size 4 on Two-Wheeled Robot Navigation Task

4.4 thus shows the initial coverages obtained with 0DAB with Spatial Random Fields ensembles of size 40. It shows that even with an increased ensemble size, Spatial Random Fields ensembles still fail to outperform randomly parameterized policies in terms of initial coverage. Indeed, Spatial Random Fields ensembles of size 40 with 0DAB generate individuals whose coverage is similar to the one obtained with ensembles of size 4, which does not increase the initial coverage compared to that of randomly parameterized policies. For the rest of this chapter, we thus decide to exclude Spatial Random Fields ensembles from our experiments, as they fail to encapsulate the dynamics of the simplest environment we consider. On the other hand, Figure 4.5 shows interesting results. Indeed, it demonstrates that increasing the ensemble size ameliorated the initial coverage obtained with the 0DAB outputted

bootstraps, as the median initial coverages with an ensemble size of 40 are slightly higher than with an ensemble size of 4.

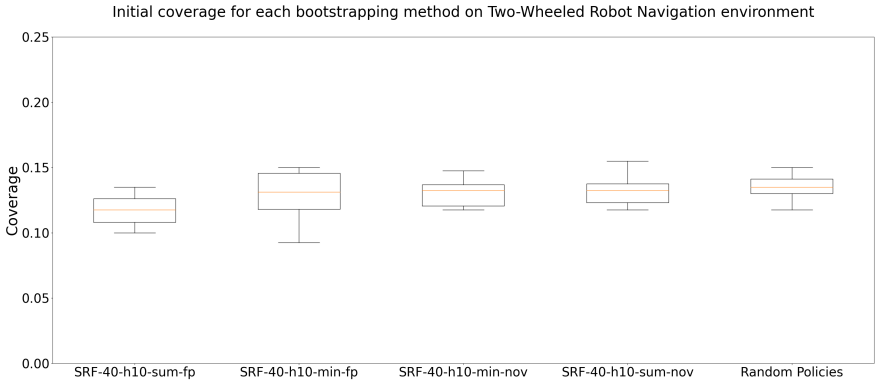


Figure 4.4: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Spatial Random Fields ensembles of size 40 on Two-Wheeled Robot Navigation Task

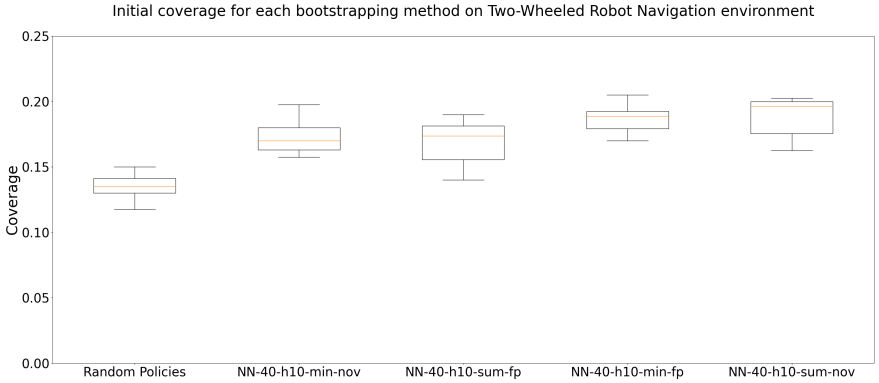


Figure 4.5: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Non-Linear Neural Network ensembles of size 40 on Two-Wheeled Robot Navigation Task

Finally, Figure 4.6 shows the results obtained with 0DAB coupled with Non-Linear Neural Networks ensembles of size 400. Increasing further the ensemble size does not seem to impact the reached initial coverage for most methods, but does seem to reduce the inter-quartile range for the 0DAB bootstraps selecting the initial population based on the novelty of the individuals. Having ensemble of size 40 being seemingly sufficient, we propose to keep 40 as the biggest ensemble size we will look at in this chapter. Overall, these results show that using Spatial Random Fields ensembles fails to generate diverse solutions for a simple robotic environment dynamics, even when increasing the ensemble size, thus driving us toward the decision to exclude them from our experiments. Nevertheless, it also showed that Non-Linear

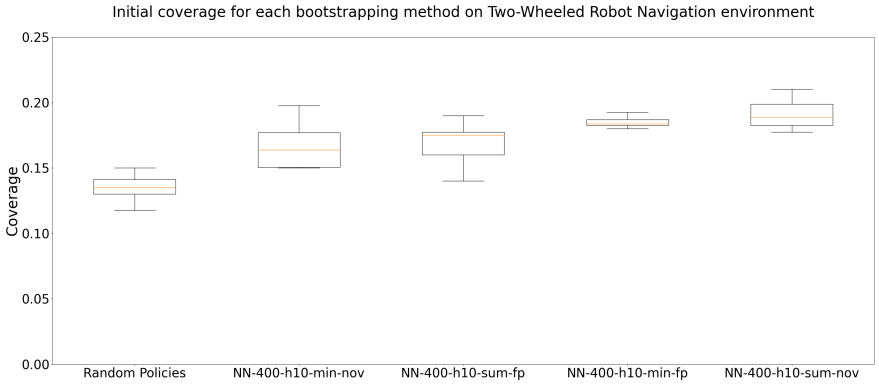


Figure 4.6: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Non-Linear Neural Network ensembles of size 400 on Two-Wheeled Robot Navigation Task

Neural Networks ensembles based random dynamics models could be promising, as an increase of up to 35% in initial coverage is observed with ensembles sizes of 4 and 40.

4.2.3 Two-Wheeled Robot Navigation Task: Influence of different bootstraps on NS

Having studied the impact on initial coverage of the various considered model representations and having restrained our study to a single dynamics model representation (being Non-Linear Neural Networks ensembles with an ensemble size of 40), it would be insightful to look at the actual impact of the bootstrap obtained with 0DAB on a subsequent Novelty Search routine. Figure 4.7 shows the coverage evolution of the 4 different combinations of novelty metric on the model and bootstrap selection methods at the end of the 0DAB procedure. As the initial coverage of each of the methods was already higher than that of randomly parameterized policies, it was expected that 0DAB bootstrapped Novelty Search routines would cover faster the environment. Nevertheless, we must also take into account that selecting particular policies as the starting point of Novelty Search can be detrimental to the policy search process. Indeed, such selected policies are not spread across the whole parameter space, biasing the parameter space exploration process towards specific regions that were identified as promising on the random dynamics ensembles.

Nevertheless, as observed on Figure 4.7 this is not the case on the Two-Wheeled Robot Navigation task for the considered bootstrapping selection methods and nov-

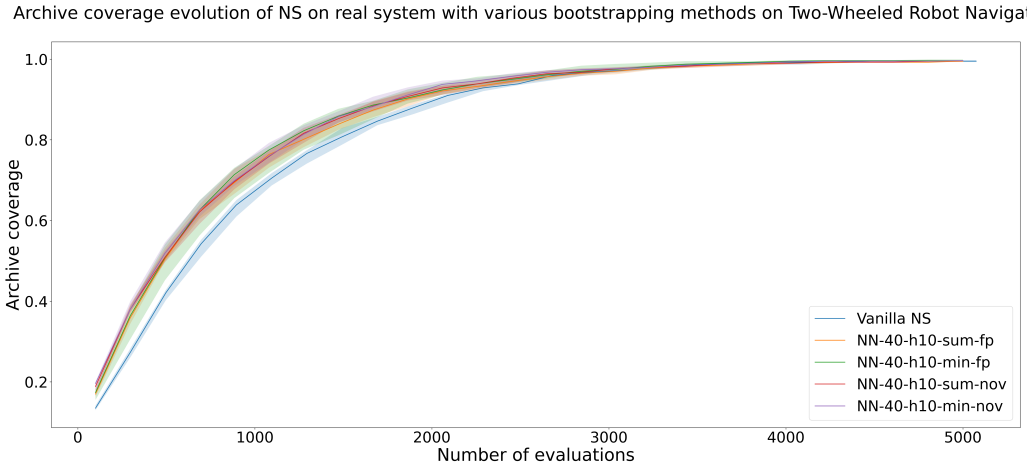


Figure 4.7: Novelty Search coverage evolution comparison between randomly parameterized policies and 0DAB selected bootstraps obtained with Non-Linear Neural Networks ensembles of size 40 on Two-Wheeled Robot Navigation Task

elty metrics on the model. We observe that even though there was a slight difference in terms of initial coverage between all the different combinations of parameterization for 0DAB considered, all four actually perform similarly on this task in terms of speed to cover the environment. A few hundred evaluations are saved using 0DAB on the Two-Wheeled Robot Navigation task, proving that 0DAB could potentially help to bootstrap the population of a Novelty Search algorithm slightly better than using randomly parameterized policies. However, it is interesting to note that the difference in sample-efficiency is not very significant even though this environment had the simpler dynamics out of the two on which experiments are made. Interesting insights were drawn from experiments on this environment, showing that ensembles should not be too small or they will not capture enough diversity in the dynamics space. Another interesting observation is that all the considered bootstrap selection methods and novelty metrics more or less perform the same, the novelty selection being a little more performing in terms of initial coverage than the others, especially when coupled with the N_{min} novelty metric.

4.2.4 Omnidirectional Hexapod Locomotion Task: Chosing the right Neural Network representation

The Omnidirectional Hexapod Locomotion Task is a task that could show the limits of the proposed method. Indeed, as its dynamics are less smooth than that of the Two-Wheeled Navigation Task, we would expect the performance observed before to be deteriorated, unless some Non-Linear Neural Network representation manages

to remain pertinent with more complex real system dynamics. To determine this, following the same process as for the Two-Wheeled Robot Navigation Task seems appropriated. Starting from simpler model representations, Figure 4.8 shows the initial coverage obtained with Non-Linear Neural Networks of size 4 with a prediction horizon of 10 timesteps. No bootstrap selection method outperforms Random Policies *w.r.t.* initial coverage with such a model representation, as all considered methods perform slightly worse or equivalently to randomly parameterized policies.

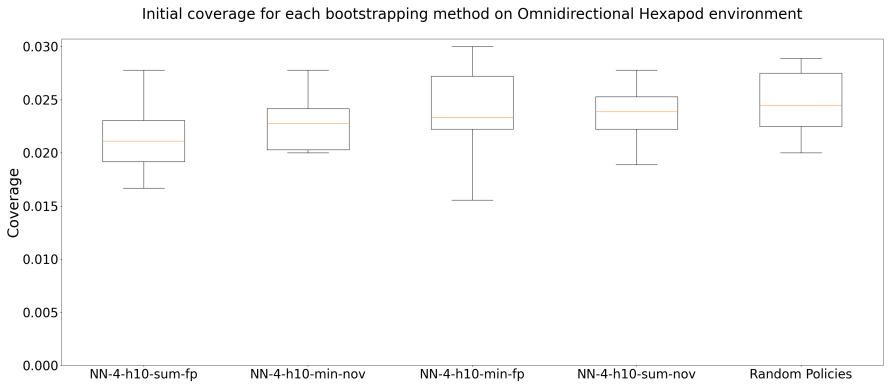


Figure 4.8: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Non-Linear Neural Network ensembles of size 4 and a prediction horizon of 10 time-steps on Omnidirectional Hexapod Locomotion Task

For the previous environment, initial coverage for the 0DAB bootstraps was already higher than that of randomly parameterized policies, even at a low ensemble size. Nevertheless, increasing the ensemble size had also increased the initial coverage reached by the bootstraps, as it helps capturing more diverse dynamics. Increasing the ensemble size to 40 and 400, Figure 4.9 shows the initial coverage obtained with such Non-Linear Neural Networks ensembles with a prediction horizon of 10 timesteps. Initial coverages still do not increase significantly higher than that of randomly parameterized policies even when increasing the ensemble size, whichever bootstrapping selection method and novelty metric is used.

Given that on the previous environment, using the N_{min} novelty metric together with the novelty selection metric gave the best initial coverages, we propose to study further the different possible model representations using only this bootstrap method and novelty metric combination. The only parameter left to consider that we decided we could adapt is the prediction horizon on the model. Increasing the model planification horizon could lead to find more diverse behaviors, but at the risk of increasing the risk of finding too many diverging behaviors. Figure 4.10 thus shows the initial coverages obtained with random model prediction horizon equal to 30 time-steps and

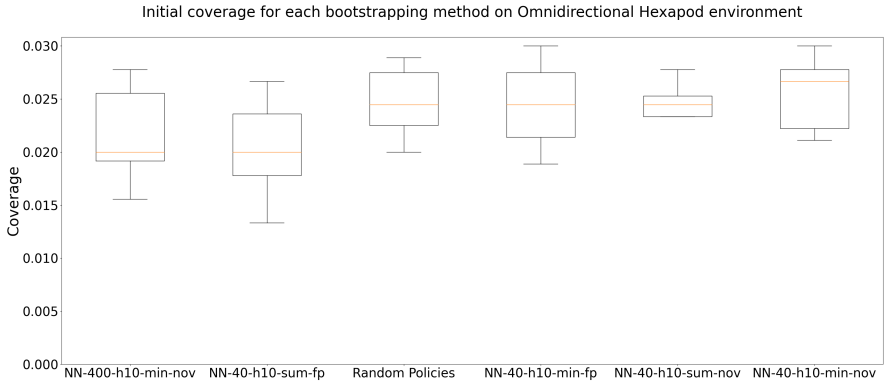


Figure 4.9: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Non-Linear Neural Network ensembles of size 40 and 400 and a prediction horizon of 10 time-steps on Omnidirectional Hexapod Locomotion Task

100 time-steps, representing 10% and 33% of the real system episode length.

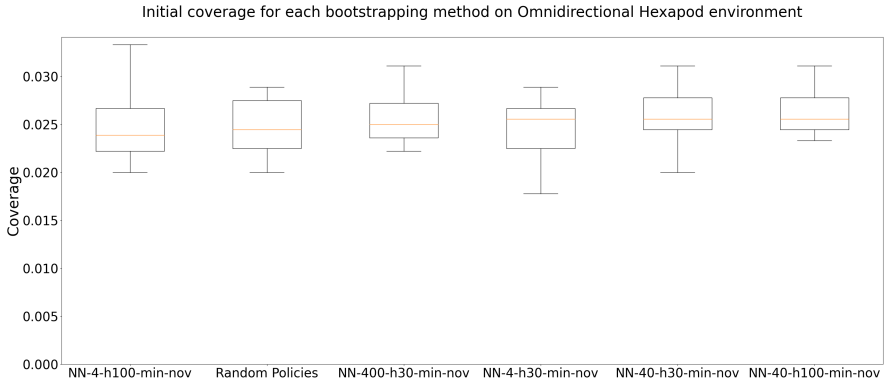


Figure 4.10: Coverage of 100 randomly parameterized policies compared to 0DAB selected bootstraps using Non-Linear Neural Network ensembles of size 4 and 40 and a prediction horizon of 30 and 100 time-steps on Omnidirectional Hexapod Locomotion Task

Once again, no model representation coupled with a different prediction horizon on the model reaches a significantly better coverage on the real system compared to randomly parameterized policies. It might not actually be that important, as what matters is the bias induced in the population which even with a similar coverage or even lower coverage could be a better starting point for Novelty Search on the real system. Nevertheless, it could also mean that the random dynamics model ensemble fail to capture some features of the target dynamics system in its dynamics distribution, which would mean that the bootstrap could be biased in the wrong

direction.

4.2.5 Omnidirectional Hexapod Locomotion Task: Influence of different bootstraps on NS

Figure 4.11 shows the impact on Novelty Search of the various bootstraps presented in Figure 4.8. Disappointingly, the lower initial coverage did seem to indicate that the random dynamics model representations proposed would not generate diverse solutions on the real system dynamics and wrongfully bias the initial population toward a sub-optimal one compared to randomly parameterized policies, as the coverage evolution of 0DAB bootstrapped Novelty Search is two times lower than that of Random Policies bootstrapped Novelty Search after 5000 evaluations. Sampling uniformly in the parameter space has the advantage to create a completely unbiased starting point which can then focus directly all the interesting regions of the parameter space, while biasing the population towards certain regions of the parameter space, which may not be interesting to the real system can be very detrimental to the policy search.

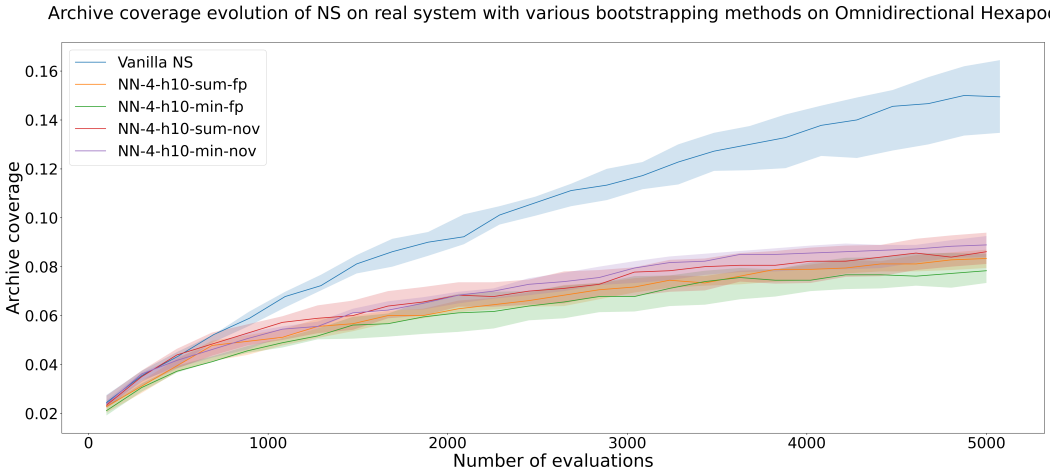


Figure 4.11: Novelty Search coverage evolution comparison between randomly parameterized policies and 0DAB selected bootstraps obtained with Non-Linear Neural Networks ensembles of size 4 on Omnidirectional Hexapod Locomotion Task

This problem might be fixed when using higher ensemble sizes. However, higher ensemble size do not reach our objective as well even though the coverage evolution is a little more promising than that of 0DAB bootstrapped with random dynamics models ensembles of size of 40 and 400, as shown on Figure 4.12. As expected, the 0DAB bootstrap leading to the best coverage among the various bootstraps considered is the one using as much as 400 models, which allow a better representation of

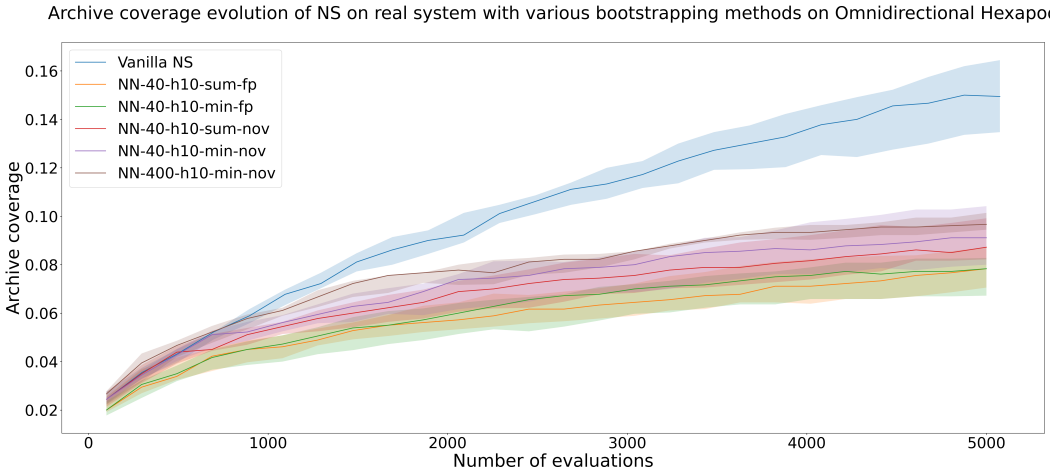


Figure 4.12: Novelty Search coverage evolution comparison between randomly parameterized policies and 0DAB selected bootstraps obtained with Non-Linear Neural Networks ensembles of size 40 and 400 on Omnidirectional Hexapod Locomotion Task

the dynamics space but still fails to generate a diverse set of individuals for the real system dynamics. Now, as stated in the previous section, the only parameter left to play around with is the prediction horizon on the model ensemble during 0DAB.

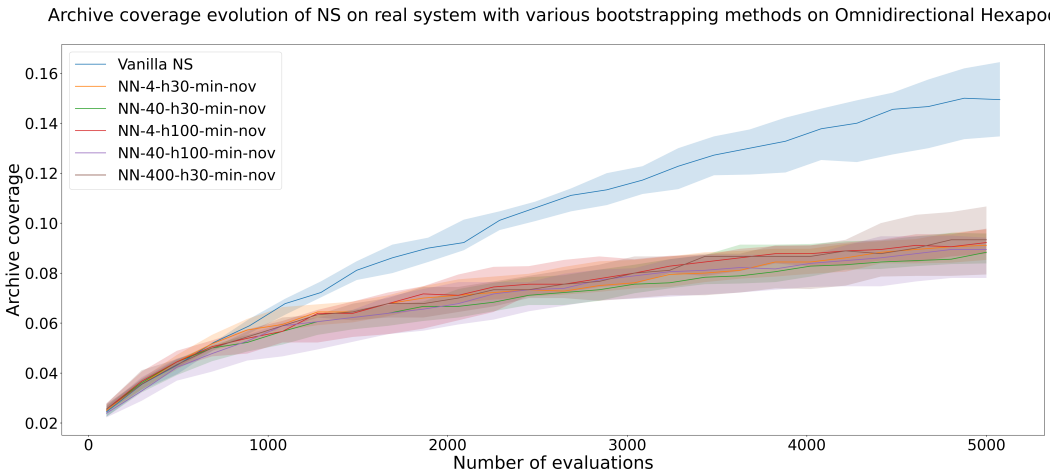


Figure 4.13: Novelty Search coverage evolution comparison between randomly parameterized policies and 0DAB selected bootstraps obtained with Non-Linear Neural Networks ensembles of size 4 and 40 and a prediction horizon of 30 and 100 time-steps on Omnidirectional Hexapod Locomotion Task

Previous section shown that even such a change did not improve the initial coverage, and Figure 4.13 shows as expected that the corresponding coverage evolution remains lower than that of Vanilla Novelty Search throughout the whole evaluation budget. A slight amelioration is observed, the final coverage after 5000 evaluations

of these parameterizations being a little higher than the one reached by all the previously considered ones. Nevertheless, all parameterizations that we considered have no significant difference between each other and all fail to enhance the sample-efficiency of the Novelty Search approach bootstrapped with randomly parameterized policies.

4.2.6 Reasons of failure

To try to understand the reasons of the failure on the more complex environment, an additional experiment is proposed. This experiment consists in transferring the final archive of a Novelty Search routine ran on the real system on the random dynamics model ensembles considered. This should show which regions of the behavior space on the random dynamics model ensembles is reached by the Novelty Search archive, which is comprised of the most diverse individuals relatively to the considered Behavioral Space. This way, a characterization of the solutions could be found if some structure emerges in our observations.

Firstly, Figure 4.14 shows what would happen in one of the ideal cases. Indeed, this figure is obtained with 0DAB bootstrapped with a Non-Linear Neural Network ensemble of size 40, which gave the best results on the Two-Wheeled Robot Navigation task. The first 8 models of the ensembles are shown on the figure, as well as the real system on the top left. 0DAB generated solutions are shown in the color blue on each figure. Novelty Search generated solutions, obtained on the real system after a complete run, are shown in orange on both the real system and the random dynamics models considered. Figure 4.14 shows that 0DAB and Novelty Search solutions almost overlap on most models.

But the model simply being a way to characterize the individual genotype, what matters most is how well the solutions are spread on the random dynamics models. Indeed, if all solutions are well spread, it means that the random dynamics models can represent the real system dynamics to some extent as the observer function on both the real system and the real models act in a similar way. Indeed, if Novelty Search generated individuals which are covering the real system also cover the random dynamics systems, it means that the random dynamics system observer function preserves the diversity properties of those individuals through an observer function that behaves similarly on all considered dynamical systems.

Furthermore, Figure 4.15 shows the coverage of Novelty Search on a Spatial Fields Random ensemble of size 40. Compared to what was obtained using Non-Linear Neural Networks, Novelty Search solutions are much more concentrated in a

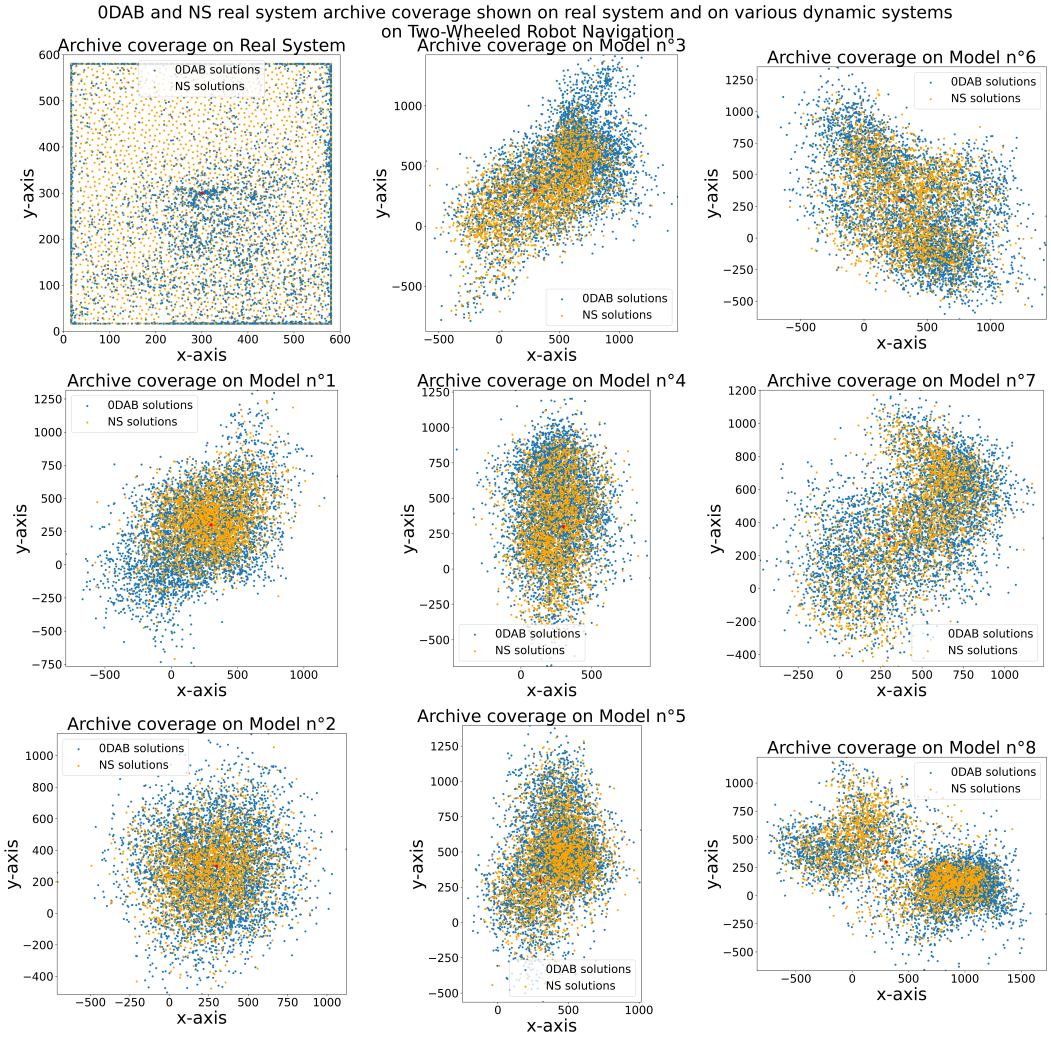


Figure 4.14: Archive Coverage after 0DAB on the real system and on the models of the Non-Linear Neural Network Ensemble of size 40 compared to that of Novelty Search on Two-Wheeled Robot Navigation Task

single zone. Having all the solutions concentrated in a single zone makes them hard to characterize on the model and thus to select for bootstrapping. This translates on the poor coverage reached by such methods as shown on the first top left figure in Figure 4.15, and echoes on the fact that if all the solutions that are diverse on the real system are not diverse on the random dynamics models, it means that the random dynamics models do not capture the dynamics properties of the real system and thus that Spatial Random Fields are not proper model representations to use for the problem at hand.

These results show that Non-Linear Neural Networks are not necessarily ideal random dynamics model representations, as shown on Figure 4.16. Indeed, the Novelty

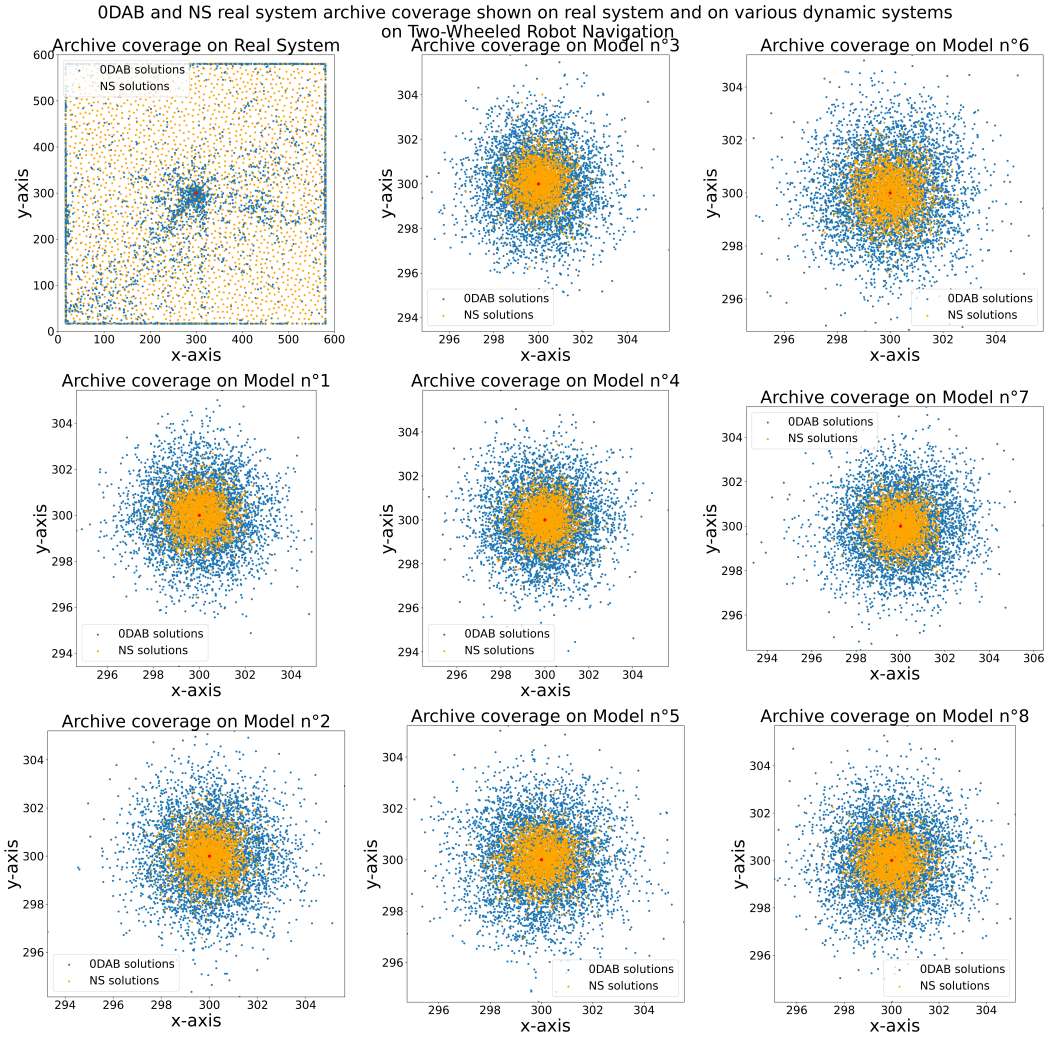


Figure 4.15: Archive Coverage after 0DAB on the real system and on the models of the Spatial Random Fields Ensemble of size 40 compared to that of Novelty Search on Two-Wheeled Robot Navigation Task

Search solution concentration is even more accentuated here as the actual dynamics of the system are completely different from the random dynamical systems generated using Non-Linear Neural Networks. The solution region is much more concentrated in the center of the model Behavioral Space, which once again makes such solutions hard to find as they all collapse to a singular region of the search space. The model representation does play an important role, as it was hypothesized at the beginning of this chapter, but determining beforehand which model representation is best suited for a specific task remains a complex task. Non-Linear Neural Networks might be of a too high representational capacity for most problems, causing a collapse of the best solutions depending on the size of the state space, while Spatial Random Fields

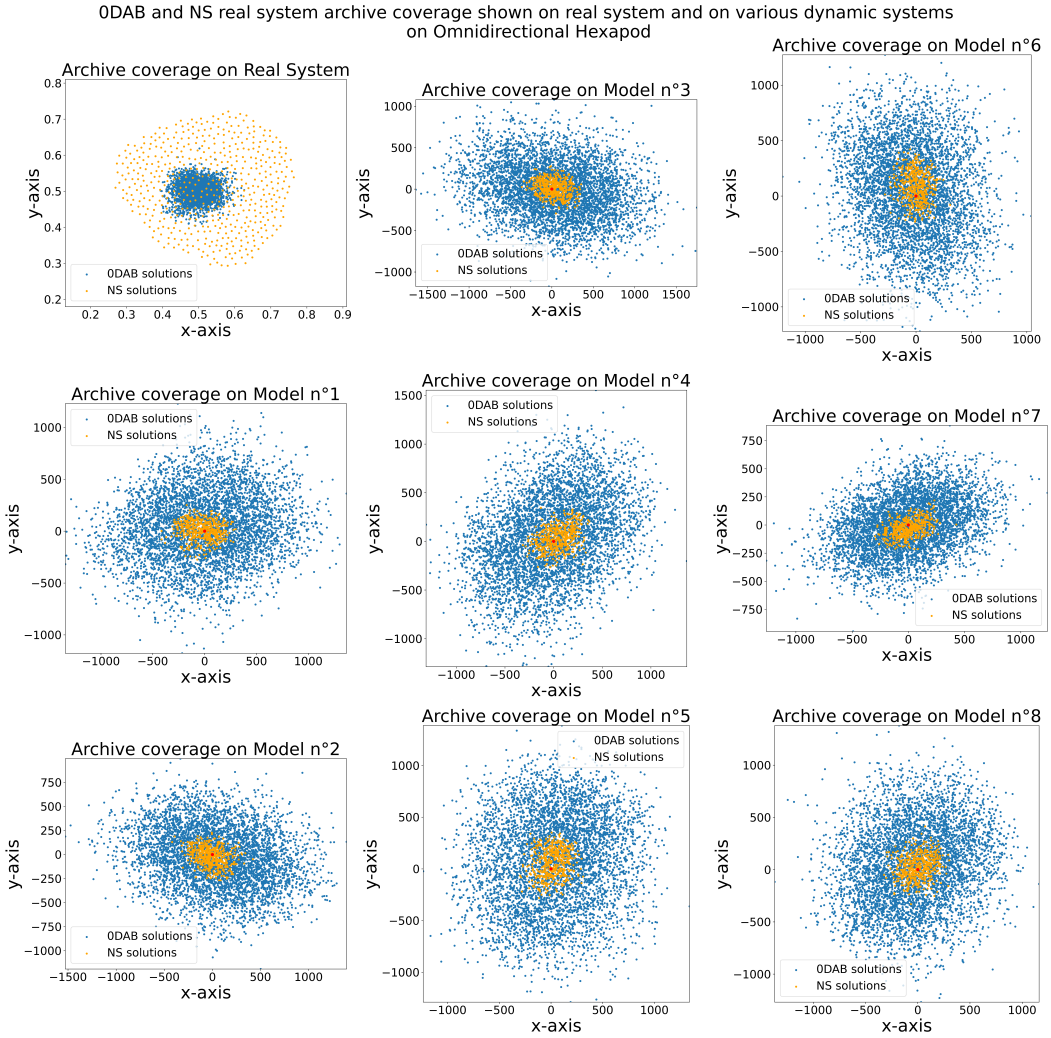


Figure 4.16: Archive Coverage after 0DAB on the real system and on the models of the Non-Linear Neural Network Ensemble compared to that of Novelty Search on Omnidirectional Hexapod Locomotion Task

might be too simple to accurately capture the dynamics of robotics systems with non-smooth dynamics.

4.3 Conclusions

In this chapter, we presented an algorithm whose aim is to generate a population bootstrap for Novelty Search called 0DAB. The proposed algorithm makes use of Novelty Search in combination with a random dynamics models ensemble to optimize for novelty on the whole model ensemble. We demonstrated that on a sim-

ple robotic navigation environment, given the appropriate random dynamics model representation, such an algorithm could help bootstrap the initial Novelty Search population by increasing the initial coverage up to 35% and reducing the number of samples required to cover completely the Outcome Space on the real system by a few hundred.

Nevertheless, we also observed that such an algorithm does not scale well to more complex robotics tasks like the Omnidirectional Hexapod Locomotion Task. Most importantly, we identified that the main limitation does not seem to be the algorithmic loop itself but rather the model representation used. Indeed, increasing the number of models and/or the prediction horizon in the 0DAB algorithm loop had a positive impact on the initial coverage and coverage evolution of a subsequent Novelty Search routine. However, no model representation was found in our works to capture properly the dynamics of the considered task, resulting in poor coverage evolution with a negative initial bias in the population used to bootstrap Novelty Search.

We analyzed the potential reasons for such results and identified that the used model representations failed to capture the system dynamics properly and resulted in solution behavioral descriptors collapsing to small regions of the Outcome Space on each of the random dynamics models of the ensemble, making characterization and selection for transfer of the interesting solutions found hard. Using more suited models, like Physics-Informed Neural Networks [33], could be an interesting lead as such models could help bring closer the considered random dynamics models closer to the reality, thus making the ensemble encapsulate better the real system dynamics in its dynamics distribution.

Chapter 5

Model-Based Novelty Search

As described in Section 2.2.2.1, Novelty Search stands out among Evolutionary Algorithms by emphasizing the exploration of the parameter space through the identification and promotion of novel solutions. Novelty Search diverges from traditional objective-based evolutionary algorithms by rewarding solutions based on their novelty rather than their performance. However, Novelty Search faces challenges in terms of sample-efficiency. Indeed, Novelty Search uniform covering properties of the Behavioral Space $\mathcal{B}$ [43] come at the cost of an extensive search, comprised of a number of uninteresting trials of the objective function. Novelty Search being an Evolutionary Algorithm, it is not surprising at all as such methods are known to be highly demanding in terms of required samples to reach their optimization objective.

In this context, it would be great to be able to characterize a potential solution derived from an already highly novel population before actually transferring it onto the real system. Using a model to that purpose is one solution. Indeed, the integration of a model in Novelty Search allows for predictive analysis of candidate solutions, enabling the algorithm to explore the solution space more intelligently. By leveraging the insights provided by the model, the algorithm can guide the search towards regions likely to contain solutions with high novelty, disregarding potentially non-novel solutions that are usually tested whatsoever on the real system as what drives the novelty would not be only the current population but also the characterization of the candidate solutions with the learned model of the system dynamics. Effectively, the idea behind adding a model to Novelty Search would be to estimate the behavioral descriptor of a larger pool of individuals to only transfer onto the real system the most novel portion. This should enhance the sample-efficiency as the algorithm would not waste as many evaluations as its model-free counterpart on

uninteresting regions of the Behavioral Space $\mathcal{B}$.

To that end, Lim *et al.* [98] proposed a model-based Quality-Diversity algorithm called Dynamics-Aware Quality-Diversity. It uses a learned dynamics model to transfer only a certain number of individuals that are deemed to be potentially added to the real archive of behaviors. This algorithm already demonstrated a faster coverage than its non model-based counterpart on their benchmark environment, and it would be interesting to compare it to our proposed algorithm to ensure that Model-Based Novelty Search is effectively faster than a model-based Quality-Diversity algorithm as Model-Based Novelty Search should maintain the strong exploration properties of its non-model-based counterpart, Novelty Search.

5.1 Methods

Our aim is to compare the performance of a Model-Based version of the Novelty Search algorithm against several other Diversity Algorithms. To that end, we decide to use three different Diversity Algorithms as baselines. Firstly, Novelty Search, as it is the algorithm we are trying to improve using a model. Secondly, we will compare to two Quality-Diversity algorithms, Dynamics-Aware Quality Diversity [98] and its non-model based counterpart as defined in their paper. The Novelty Search algorithm is detailed in Algorithm 1 and Dynamics-Aware Quality Diversity is detailed in Algorithm 5. The non model-based Quality-Diversity algorithm used, denoted Vanilla Quality-Diversity, has the same repertoire addition rules as Dynamics-Aware Quality-Diversity that were proposed by Cully *et al.* [32]. Those addition rules put the emphasis on both exploration with the preservation of novel behaviors in the archive $\mathcal{A}_\Pi$ and quality through local competition within a niche in the archive as proposed by Lehman *et al.* [92] and refined by Cully *et al.* [32]. This niching system thus keeps the Behavioral Space continuous rather than discretizing it like the MAP-Elites algorithm described in Algorithm 2, thus making the archive used unstructured. Both Dynamics-Aware Quality Diversity and Vanilla Quality-Diversity use a uniform selection operator and directional variation [162] to drive the Quality-Diversity search. Sets of hyperparameters for each algorithm and experiment will be further detailed in the next section.

We thus propose a Model-Based version of the Novelty Search algorithm. It consists in two phases, much like Dynamics-Aware Quality Diversity [98]. Initialization consists in generating a population of size s_p of randomly parameterized policies, to evaluate them and train the dynamics model using the foraged data. Afterwards, a

Novelty Search loop is performed on the dynamics model using T_{inf} inference. The starting population on the model Novelty Search procedure is set to the last batch of s_p individuals evaluated. From this population is generated s_o offspring policies using a selection and variation mechanism. The offspring is then evaluated on the dynamics model to estimate their behavior descriptor and the s_p most novel ones are selected as the new population. Moreover, k individuals are selected to be added to the archive $\widetilde{\mathcal{A}_\Pi}$, either randomly or based on their novelty metric and all the model evaluated individuals are saved in a container $\tilde{E}$ to allow for a more fine selection at the end of the model Novelty Search process.

This selection-variation-evaluation process is repeated $\tilde{G}$ times, $\tilde{G}$ being adaptive on the current real generation G inspired from the algorithm proposed by Janner *et al.* [68]. $\tilde{G}$ thus increases over the complete evaluation budget, starting from $\tilde{G}_0$ up to 10% of the maximum real evaluation budget and increasing linearly up to $\tilde{G}_{max}$ once 90% of the maximum real evaluation budget is reached. Once $\tilde{G}$ is reached, it is time to select which individuals to transfer onto the real system. To do so, we propose to transfer the s_p most novel individuals evaluated on the model $w.r.t$ to the current real archive of behaviors $\mathcal{A}_\Pi$ and all the model evaluations performed. Those selected individuals are then evaluated on the real system, the dynamics model is updated and k individuals are selected to be added to the archive $\mathcal{A}_\Pi$. The complete Model-Based Novelty Search algorithm is outlined in Algorithm 7.

5.2 Experiments

5.2.1 Experimental Setups

To evaluate the performance of the proposed Model-Based Novelty Search algorithm, experiments are conducted on three environments. The considered metric for all of the environments is the coverage of the user-defined Behavioral Space $\mathcal{B}$ depending on the number of evaluations. Ideally, our method should outperform at least Novelty Search on all considered environments with a lower number of evaluations to reach a certain coverage, and should also outperform its Model-Based Quality-Diversity counterpart, Dynamics-Aware Quality-Diversity [98], as we want to provide a Model-Based Novelty Search algorithm that keeps the strong exploration capabilities of Novelty Search when compared to Quality-Diversity algorithms. The three environments considered are a Two-Wheeled Robot Navigation task, an Omnidirectional Hexapod Locomotion task and a Ball In Cup task.

Algorithm 7 Model-Based Novelty Search (MBNS)

s_p is the population size, s_o is the offspring size, k is the number of individuals to add to the archive at each generation, G is the generation budget, $\tilde{G}$ is the generation budget in imagination
Initialize repertoire $\mathcal{A}_\Pi$ (to $\emptyset$), imagined repertoire $\widetilde{\mathcal{A}_\Pi}$ (to $\emptyset$), dynamics model p_{dyn}, and replay buffer $\mathcal{B}$ (to $\emptyset$)
$\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}} \leftarrow$ random_parameters() ▷ *Set the population to randomly parameterized policies*

$(\boldsymbol{sd}_i, R_i)_{i \in [1,N]}$, transitions $\leftarrow$ evaluation$(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_N})$ ▷ *Evaluate in environment; get transitions.*
$\mathcal{B} \leftarrow$ add_to_replay_buffer(transitions, $\mathcal{B}$)

▷ *Learning Dynamics Models*
Update p_{dyn} using $\mathcal{B}$ ▷ *Train dynamics model with transitions collected in replay buffer.*

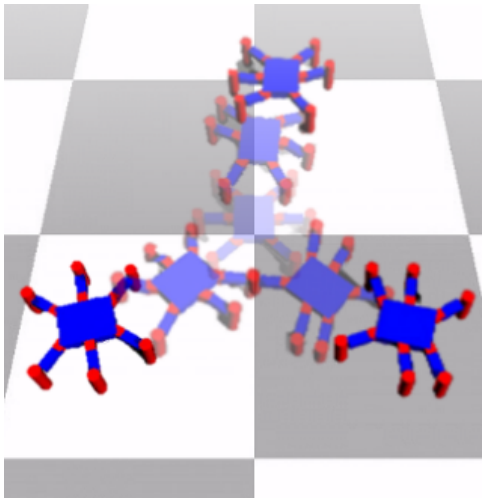
for g *in* $[1, \dots, G]$ **do**
    $\tilde{E} \leftarrow \emptyset$ ▷ Initialize model evaluations container
    for $\tilde{g}$ *in* $[1, \dots, \tilde{G}]$ **do**
        $\pi_{\boldsymbol{\theta}^*_1}, \dots, \pi_{\boldsymbol{\theta}^*_{s_o}} \leftarrow variation(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Generate the model offspring from the model population using the variation operator*
        $\widetilde{b}_{i \in [1,s_o]} \leftarrow$ model_evaluate$(\pi_{\boldsymbol{\theta}^*_1}, \dots, \pi_{\boldsymbol{\theta}^*_{s_o}}$ using $p_{dyn})$ ▷ *Evaluate the model offspring*
        $\widetilde{\eta}_{i \in [1,s_o]} \leftarrow N(1), \dots, N(s_o)$ ▷ *Compute the Novelty using the model offspring behavioral descriptors*
        $\tilde{E} \leftarrow \pi_{\boldsymbol{\theta}^*_1} \dots \pi_{\boldsymbol{\theta}^*_{s_o}}$ ▷ Add evaluated individuals to model evaluations container $\tilde{E}$
        $\pi_{\boldsymbol{\theta}_1}, \dots, \pi_{\boldsymbol{\theta}_{s_p}} \leftarrow select(\pi_{\boldsymbol{\theta}^*_1} \dots \pi_{\boldsymbol{\theta}^*_{s_o}}, \eta_1 \dots \eta_{s_o})$ ▷ *Update population with the s_p most novel individuals in the model offspring*
        $\widetilde{\mathcal{A}_\Pi} \leftarrow sample(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Select k individuals from the population to be added to the model archive*
        $\tilde{g} \leftarrow \tilde{g} + 1$
    $\pi_{\boldsymbol{\theta}_1}, \dots, \pi_{\boldsymbol{\theta}_{s_p}} \leftarrow select\_for\_transfer(\tilde{E})$ ▷ Select s_p most novel individuals to transfer on the real system from model evaluations container $\tilde{E}$

    ▷ *Acting in the Environment*
    $b_{i \in [1,s_p]}$, transitions $\leftarrow$ evaluation$(\pi_{\boldsymbol{\theta}_1}, \dots, \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Evaluate the selected population in environment; get transitions.*
    $\mathcal{B} \leftarrow$ add_to_replay_buffer(transitions, $\mathcal{B}$)
    $\mathcal{A}_\Pi \leftarrow sample(\pi_{\boldsymbol{\theta}_1} \dots \pi_{\boldsymbol{\theta}_{s_p}})$ ▷ *Select k individuals from the population to be added to the archive*
    $\widetilde{\mathcal{A}_\Pi} \leftarrow$ synchronise_repertoires$(\mathcal{A}_\Pi, \widetilde{\mathcal{A}_\Pi})$ ▷ *Erase content of $\widetilde{\mathcal{A}_\Pi}$ and replace it with the content from $\mathcal{A}_\Pi$.*

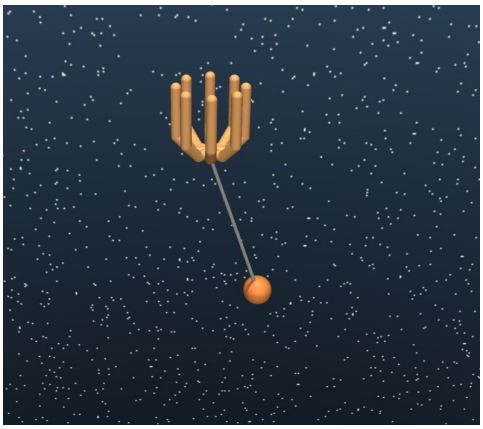
    ▷ *Learning Dynamics Models*
    Update p_{dyn} using $\mathcal{B}$ ▷ *Train dynamics model with transitions collected in replay buffer.*

return $\mathcal{A}_\Pi$

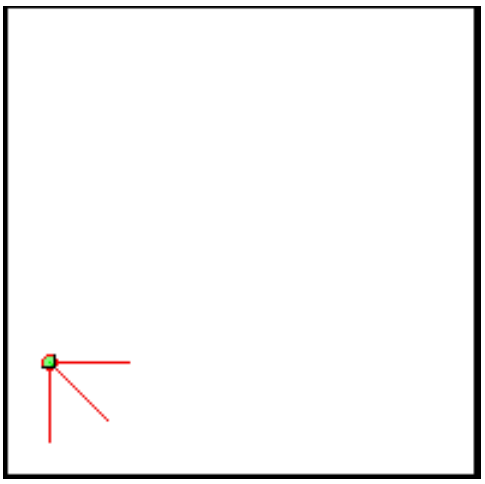
The Two-Wheeled Robot Navigation task is the same that was described in the previous chapter 4. An image of the setup is shown on Figure 5.1c. This environment



(a) Omnidirectional Hexapod Robot Locomotion



(b) Ball In Cup



(c) 2-Wheeled Robot Navigation

Figure 5.1: Considered Environments

is interesting as it has simple dynamics which are easy to learn as obstacles are only placed on the outskirts of the navigation map. The considered algorithms performance could be affected from the long episode length making mutated individuals farther from their parents in the Outcome Space and making model-based methods suffer from the long prediction horizons over which compounding error can lead to significant behavioral descriptor estimation errors.

The second environment considered is directly taken from the Dynamics-Aware Quality-Diversity paper and is the same that was used in the previous chapter as well 4. It consists in an 18-DoF hexapod robot whose goal is to learn locomotion skills as shown on Figure 5.1a. An episode lasts three seconds at a frequency of 100Hz,

making the prediction horizon 300 steps. In this task, non-linear neural networks are not used as a policy representation. Instead, Cully *et al.* [31] proposed to use periodic functions, as periodic functions allow to control each joint using only the three parameters defining a periodic function, those parameters being the phase, amplitude and duty cycle of a sinusoid wave. Such policies are particularly well suited to locomotion as they induce periodic movement priors. The prediction horizon being shorter than that of the Two-Wheeled Navigation task, this task should be solved with a little more ease by evolutionary algorithms and their model-based counterparts, making it an interesting benchmark to evaluate the unaffected performance of each algorithm.

Finally, the last environment considered is the Ball In Cup task as described in Section 3.2.1.2. This environment is particularly interesting to showcase the importance of thriving for diversity, as its dynamics are sparse. Thus, purely novelty driven methods should perform better on this environment compared to methods that also optimize for quality as they will tend to leave aside the sparse parts of the transition function or at least until its non-sparse parts are explored more thoroughly.

5.2.2 Hyperparameters

For our experiments, there is four type of hyperparameters: Evolutionary Algorithms parameters, common to all the policy search algorithms compared here, Novelty Search parameters, common to Novelty Search and Model-Based Novelty Search, Quality-Diversity parameters, common to Quality-Diversity and Dynamics-Aware Quality-Diversity and finally Model-Based parameters, common to the two considered model-based diversity algorithms Model-Based Novelty Search and Dynamics-Aware Quality-Diversity.

For all the Diversity Algorithms considered, policies parameters value are restrained in the range $[-5, 5]$ for all environments except the Omnidirectional Hexapod Locomotion task where policy parameters values are restricted to $[0, 1]$ as the policies are represented with periodic functions and not non-linear neural networks. Non-linear neural networks of 2 hidden layers of 10 neurons each are used as policies. A uniform selection mechanism is used together with the Iso+LineDD mutation [162] with a $\sigma = 0.01$ for the isotropic Gaussian part and $\sigma = 0.2$ for the directional Gaussian part to variate the individuals. The mutation probability is fixed to 20%,

EA Hyperparameters			
Parameter	TWR Value	OH Value	BIC Value
Policy Representation	Non-linear Neural Network	Periodic Functions	Non-linear Neural Network
Policy Parameter Range	$[-5, 5]$	$[0, 1]$	$[-5, 5]$
Selection Mechanism	Uniform	Uniform	Uniform
Variation Mechanism	Iso+LineDD [162]	Iso+LineDD [162]	Iso+LineDD [162]
Variation Probability	20%	20%	20%
nov_l	9	0.015	0.012
QD Hyperparameters			
Initialization Size	100		
Offspring Size	200		
NS Hyperparameters			
Population Size	100		
Offspring Size	200		
Individuals added to $\mathcal{A}_\Pi$	15		
Selection criteria to add to $\mathcal{A}_\Pi$	novelty		
MB Hyperparameters			
Ensemble Size	4		
Batch Size	512		
Learning Rate	0.001		
Model Training Rate	every 500 evaluations		

Table 5.1: Hyperparameters common to all diversity algorithms used

and as an unstructured archive of behaviors [92] (only for QD-based algorithms and just for coverage evaluation in NS-based algorithms) is used, the distance between competing individuals is set to 1.5% of the size of the Behavioral Space.

Quality-Diversity algorithms use a random initialization comprised of a 100 randomly parameterized individuals, and the batch size to generate after a mutation procedure is 200 individuals. Novelty Search algorithms use a population size of a 100 individuals, and an offspring size of 200. The number of individuals added to the archive at each generation is 15 and are selected based on their novelty. Model-Based techniques use a Probabilistic Neural Network Ensemble, comprised of 4 different neural networks of 2 layers, the first one consisting in 500 neurons and the last one 400. The batch size is set to 512, the learning rate to 0.001 and the model is trained every 500 real-world evaluations. Finally, Model-Based Novelty Search generations $\tilde{G}$ on the model increase linearly with the current budget and maximum budget ratio from $\tilde{G}_0 = 4$ generations to $\tilde{G}_{max} = 20$ generations.

5.2.3 Results

5.2.3.1 Navigation Task

Figure 5.2 shows the results obtained on the Two-Wheeled Robot Navigation task obtained on a total of 25000 evaluations and 10 repetitions for each method. All methods reach the maximum coverage in that amount of evaluations. That coverage is shown for each of the considered methods on Figure 5.3, which shows the coverage of 10 separate runs on the same figure for each considered algorithm. All methods indeed achieve a similarly dense coverage of the complete Outcome Space. To analyze the performance in terms of speed to reach this coverage, it is interesting to look at the archive coverage evolution in function of the number of evaluations on Figure 5.4. The view is restricted to the 5000 first evaluations for clarity purposes. On this figure, two interesting things are observed. Firstly, Model-Based Novelty Search is around 50% faster than Novelty Search and Dynamics-Aware Quality-Diversity. Indeed, Model-Based Novelty Search reaches the final Novelty Search coverage in 3300 evaluations instead of 5000 evaluations. Similarly, it reaches the final Dynamics-Aware Quality-Diversity coverage in 3500 evaluations instead of 5000.

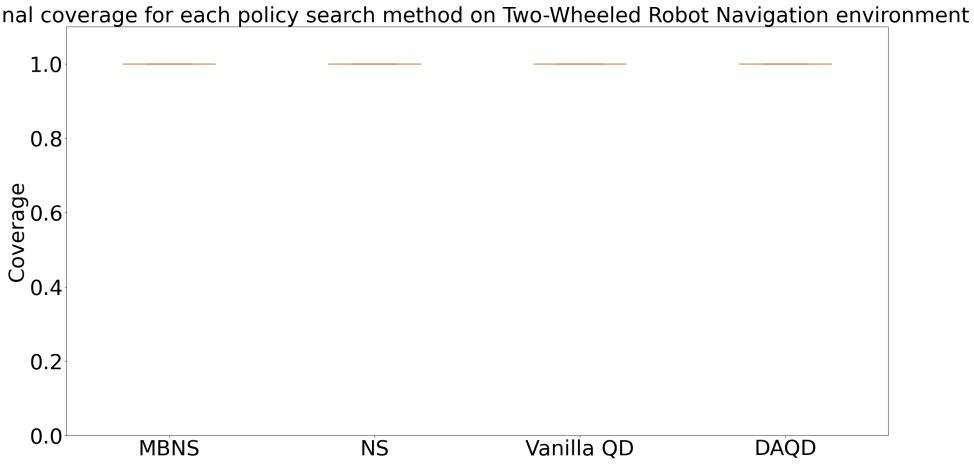
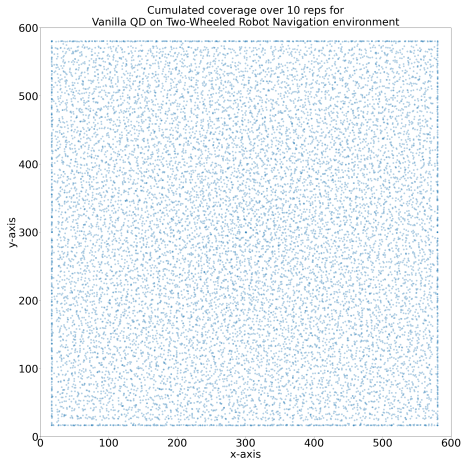
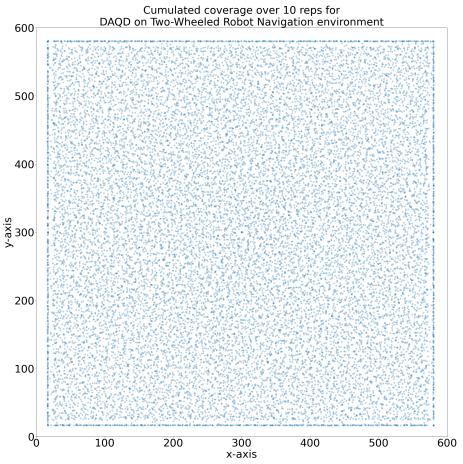


Figure 5.2: Final Coverage of Diversity Methods on Empty Maze task

Secondly, Model-Based Novelty-Search is slower than all other Policy Search methods during the first 600 evaluations, which seems to be due to over-exploitation of the model. To explain this, Figure 5.5 shows the descriptor estimation error evolution of individuals transferred after the optimization procedure on the learned model for both Model-Based Novelty Search and Dynamics-Aware Quality-Diversity. Descriptor estimation error is calculated as the L2 norm of the difference between the

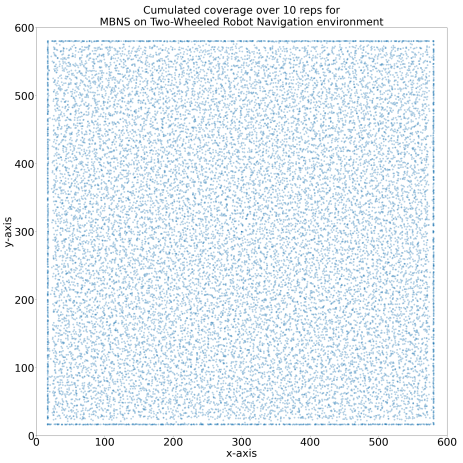


(a) QD archive cumulated coverage



(b) DAQD cumulated coverage

(c) NS cumulated coverage



(d) MBNS cumulated coverage

Figure 5.3: Cumulated coverage over 10 repetitions for each of the Diversity Algorithms considered on the Two-Wheeled Robot Navigation task

real behavioral descriptor and the estimated behavioral descriptor using the learned model such that: $\epsilon_{desc}(\pi_{\boldsymbol{\theta}_i}) = \sqrt{(b_i - \widetilde{b_i})^2}$. On this figure, at the same step where the model is updated (after 500 evaluations), a steep drop in descriptor estimation error happens. This fall is way more pronounced in Model-Based Novelty Search, with descriptor estimation error values going from 150-300 to 30-60, while Dynamics-Aware Quality-Diversity had descriptor estimation error values that remained in the 50-125 range. Moreover, the median descriptor estimation error of selected individuals for Model-Based Novelty Search quickly goes down after training the model with more gathered data and goes below that of Dynamics-Aware Quality-Diversity. This

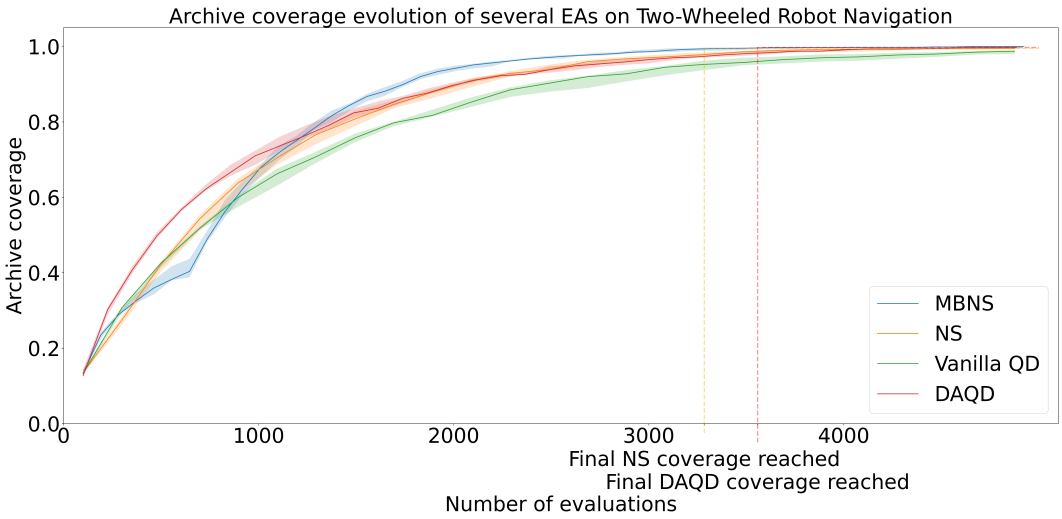


Figure 5.4: Coverage of Diversity Methods on the Two-Wheeled Robot Navigation task depending on number of evaluations

shows that on the Two-Wheeled Robot Navigation task, Model-Based Novelty Search is able to exploit the model more robustly than Dynamics-Aware Quality-Diversity once the model has lower prediction errors.

Nevertheless, why Model-Based Novelty Search has a poor initial coverage over the first 500 evaluations still needs to be investigated. This can be explained with the different way that each Model-Based algorithm utilizes its model. Indeed, in Dynamics-Aware Quality-Diversity, as soon as the model estimates that at least 100 individuals could be added to the archive $\mathcal{A}_\Pi$, those individuals are transferred. This can lead in the early generations to simply performing a single or a few selection and mutation iteration, as the model can predict behavioral descriptor values that are not reliable but remain diverse as no explicit objective that may overexploit the model is defined. This effectively allows Dynamics-Aware Quality-Diversity to not overuse the model and simply perform better than its non-model-based counterpart. On the contrary, Model-Based Novelty-Search has an explicit novelty objective, which will increase the over-exploitation of the model. Selecting the most novel policies in that case is not ideal as the model is far from good, which can bring a bias in the real diversity found in the solutions.

To support this claim, the evolution of the descriptor estimation error of selected and non-selected individuals is observed at the end of the Novelty Search routine on the learned model on the Two-Wheeled Robot Navigation task respectively shown on Figure 5.6 and Figure 5.7. If our claim is correct, on the first 500 evaluations a significantly higher descriptor estimation error on the selected individuals compared to

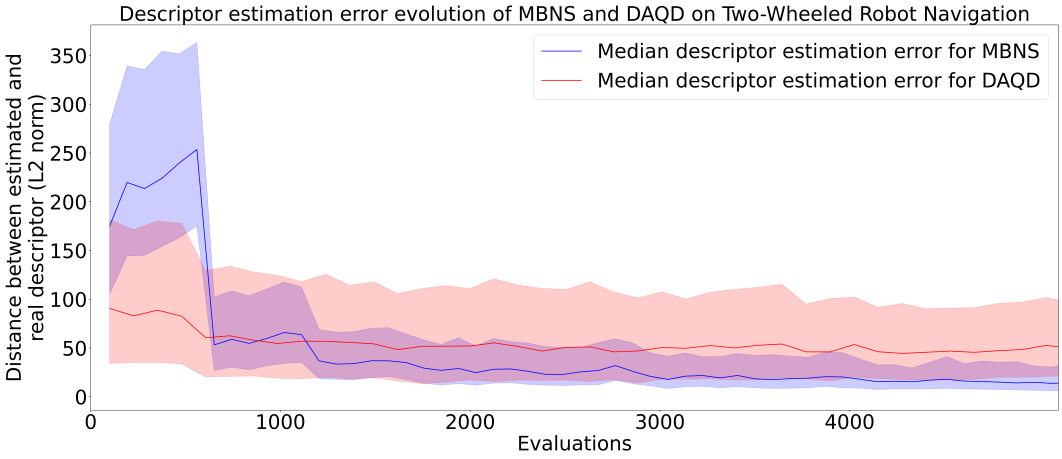


Figure 5.5: Comparison of the evolution of all transferred individuals estimated behavioral descriptor error between MBNS and DAQD on the Two-Wheeled Robot Navigation task. N.B.: In Model-Based Novelty Search, the transferred individuals are the most novel.

the non-selected individuals should be observed, error which should then be reduced after enough model training is done. Furthermore, the novelty of selected individuals should remain significantly higher than that of non-selected individuals throughout all evaluations. On Figure 5.6, the gap between selected and non-selected individuals descriptor estimation error is of more than 100, selected individuals almost having twice the descriptor estimation error than non-selected individuals. After 500 evaluations, this gap is drastically reduced, selected individuals only having a descriptor estimation error that is only 20-30% higher at maximum compared to non-selected individuals. On Figure 5.7, the gap in estimated novelty between selected and non-selected individuals remains more or less the same throughout all evaluations, showing that Model-Based Novelty Search is able to use the model to find high novelty individuals once the model is enough trained.

Moreover, Figure 5.8 shows the L2 error of dynamics models trained during Model-Based Novelty Search or Dynamics-Aware Quality-Diversity routines against a set of diverse trajectories obtained using Novelty Search on the real system considered. Those trajectories thus form an independent and diverse test set representing the real system dynamics, ideal to evaluate the dynamics model performance. The model error is thus computed by taking the L2 norm of the difference between the real next state (obtained from the trajectory data) and the estimated next state (obtained using the learned dynamics model) for a given state and action as input. The idea is that if the model L2 Error is the same for both Model-Based Novelty Search and Dynamics-Aware Quality-Diversity while the transferred individuals L2

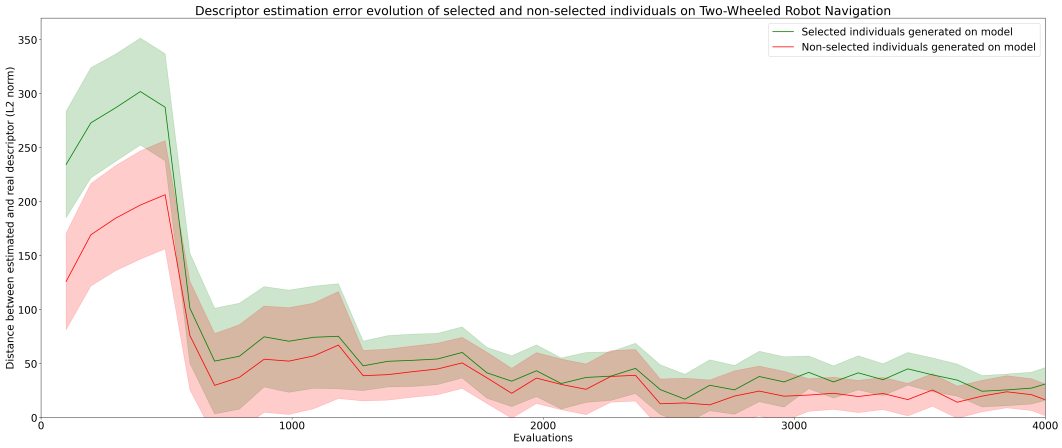


Figure 5.6: Evolution of selected vs non-selected individuals estimated behavioral descriptor error with MBNS on the Two-Wheeled Robot Navigation task

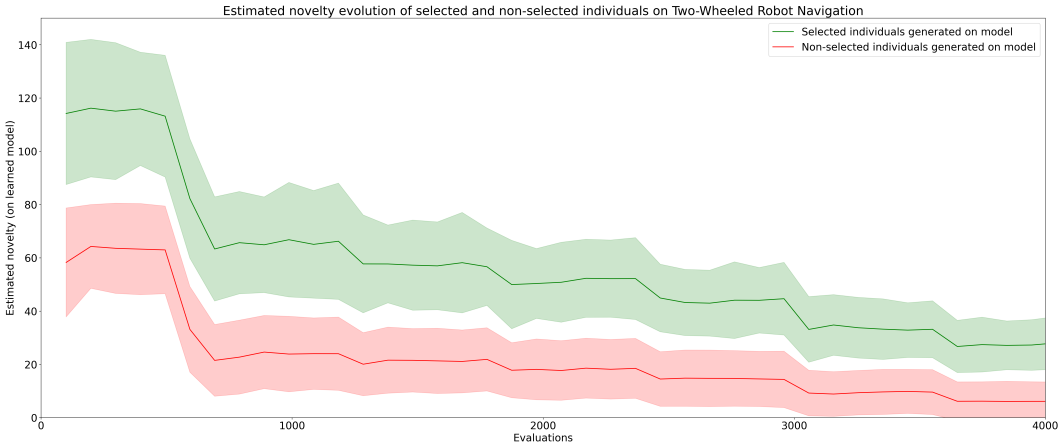


Figure 5.7: Evolution of selected vs non-selected individuals model novelty with MBNS on the Two-Wheeled Robot Navigation task

Error on the descriptor is way higher it definitely means that Model-Based Novelty Search initial lower coverage is due to model exploitation. This is confirmed as when looking at Figure 5.8, dynamics model trained with either of the two methods behave similarly *w.r.t.* the diverse set of trajectories considered. This validates our claim that the model is over-exploited in the first 500 evaluations, and explain the poor performance in terms of coverage of Model-Based Novelty Search when the model has prediction errors that are too high.

5.2.3.2 Omnidirectional Hexapod Locomotion Task

On the Locomotion task, the control of an Omnidirectional Hexapod robot, results are obtained on a total of 50 000 evaluations and 10 repetitions for each method con-

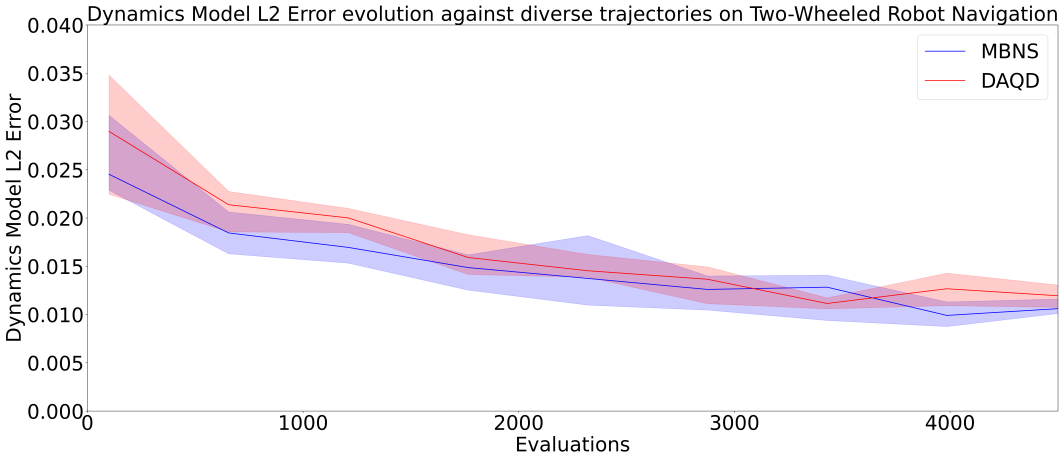


Figure 5.8: Evolution of the Dynamics Model L2 Error on a diverse set of 100 unseen trajectories for Dynamics Models trained independently with MBNS and DAQD

sidered. On Figure 5.9, the final coverage reached for a budget of 50 000 evaluations with Model-Based Novelty Search is twice the final coverage reached with Novelty Search, 50% higher than the one reached by Dynamics-Aware Quality-Diversity and thrice the one reached by the non model-based QD algorithm considered. The corresponding coverage is shown for each of the considered algorithms on Figure 5.10, where Model-Based Novelty Search indeed reaches regions of the Outcome Space that are never reached by any other considered method. This metric indicates that Model-Based Novelty Search can cover faster the Behavioral Space of this task than any other method considered. Now Figure 5.11, helps to analyze the performance of Model-Based Novelty Search in terms of speed to reach the final coverage of each of the other methods considered.

Model-Based Novelty Search is more than 4 times faster than Novelty Search on this task. Indeed, at 12 000 evaluations, Model-Based Novelty Search has reach a 30% coverage of the environment, while it takes all 50 000 evaluations for Novelty Search to reach a similar coverage. Compared to Dynamics-Aware Quality-Diversity, Model-Based Novelty Search is 2.5 times faster, as it reaches the final Dynamics-Aware Quality-Diversity coverage of 40% of the Outcome Space at 19 500 evaluations only instead of 50 000 evaluations. Model-Based Novelty Search is thus much more sample-efficient than its non-model based counterpart and than a non novelty incentivized model-based diversity algorithm on the Omnidirectional Hexapod Locomotion Task. This task being the one where we expected the benchmarked algorithms to perform best, it is interesting to see such a gap in speed of coverage for our proposed exploration oriented Model-Based Diversity Algorithm.

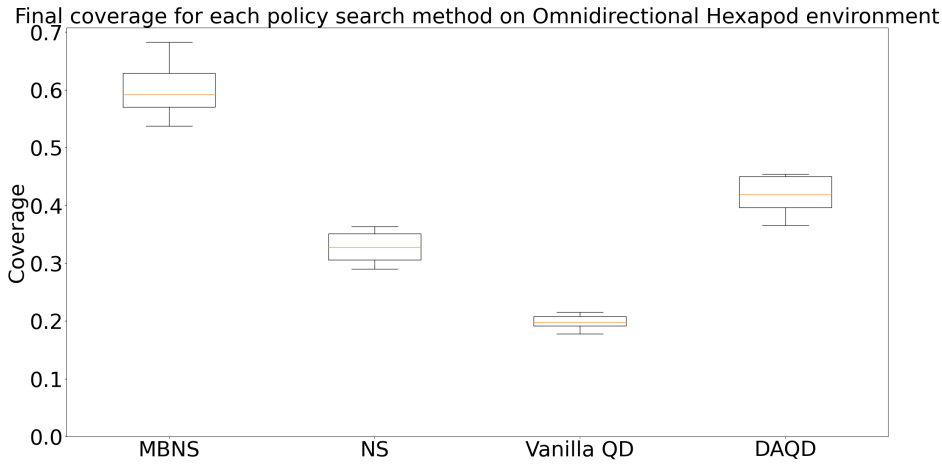
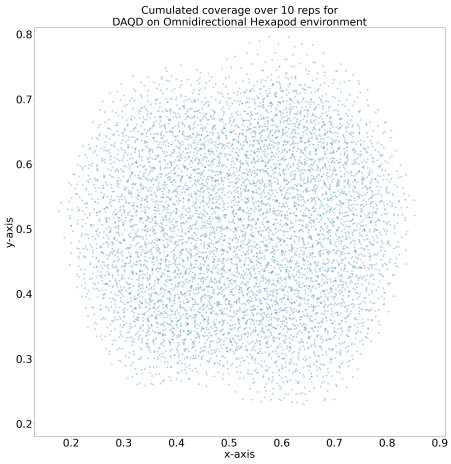


Figure 5.9: Final Coverage of Diversity Methods on Omnidirectional Hexapod task

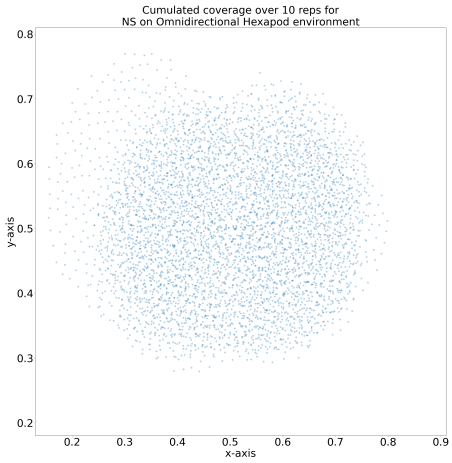
Nevertheless, on this task like on the previous one, Model-Based Novelty Search is once again slower than any other method during the first thousand evaluations. Over-exploitation of the model seems to be the culprit again, as the descriptor estimation error for individuals selected for transfer is twice the one of Dynamics-Aware Quality-Diversity on the first 1000 evaluations as shown on Figure 5.12. Indeed, Model-Based Novelty Search behavioral descriptor estimation error values are in the range of 0.05-0.20, while Dynamics-Aware Quality-Diversity tends to select individuals with a better estimation of their behavioral descriptor with estimation errors values lying in the range of 0.025-0.10. This limitation is quickly addressed as the model becomes better at predicting individuals trajectories. Indeed, the median descriptor estimation error of selected individuals for Model-Based Novelty Search joins that of Dynamics-Aware Quality-Diversity after 5000 evaluations, but remains slighlty higher than that obtained with Dynamics-Aware Quality-Diversity over all the evaluations.

Once again, to emphasize the reason of that initial under performance of Model-Based Novelty Search in terms of coverage, we propose to look at Figures 5.13 and 5.14 which show the mean and standard deviation of the descriptor estimation error and the estimated novelty obtained on the model before transferring any individual. Again, after each model training, the gap in descriptor estimation error between selected and non-selected individuals shrinks, thus bringing better novelty estimates that are properly exploited by Model-Based Novelty Search to generate novel solutions. Indeed, the gap reduces to the same ranges observed on the Two-Wheeled Robot Navigation task, going from almost twice the error in the first 500

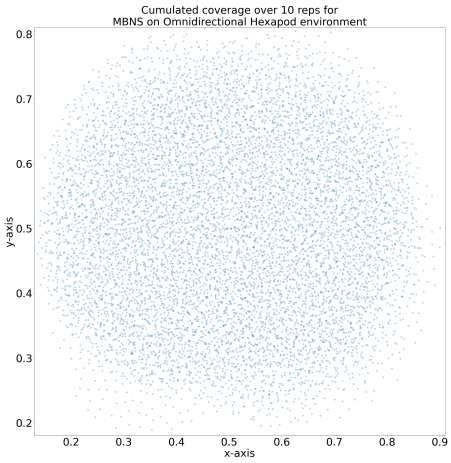
(a) QD cumulated coverage



(b) DAQD cumulated coverage



(c) NS cumulated coverage



(d) MBNS cumulated coverage

Figure 5.10: Cumulated coverage over 10 repetitions for each of the Diversity Algorithms considered on the Omnidirectional Hexapod Locomotion task

evaluations to less than 30% after 2000 evaluations. Moreover, Figure 5.15 shows that the L2 error of dynamics models trained during Model-Based Novelty Search or Dynamics-Aware Quality-Diversity routines against a set of diverse trajectories are similar, meaning that the only explanation possible for lower coverage is indeed over-exploitation of the model through the novelty objective.

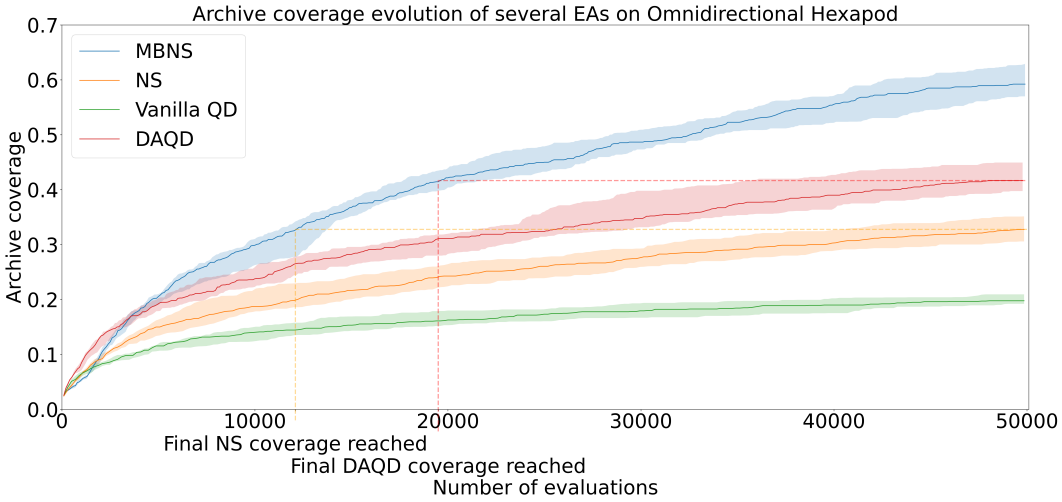


Figure 5.11: Coverage of Diversity Methods on Omnidirectional Hexapod task depending on number of evaluations

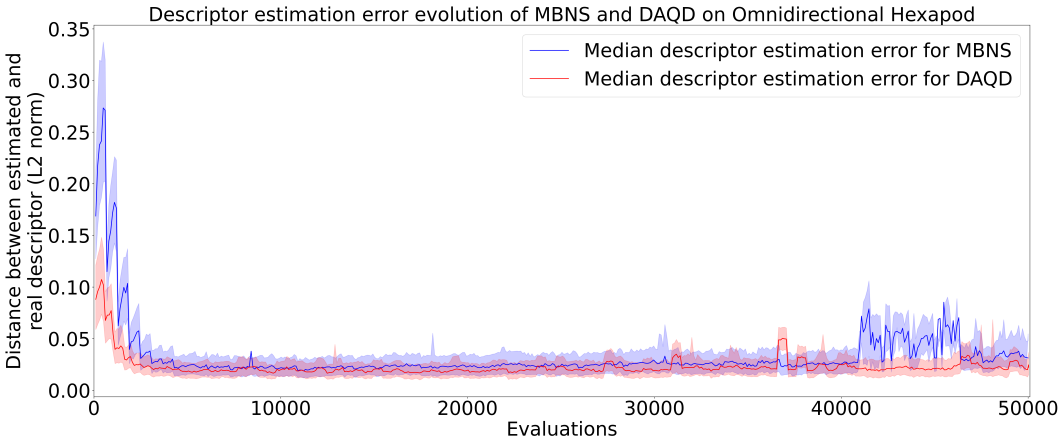


Figure 5.12: Comparison of the evolution of all transferred individuals estimated behavioral descriptor error between MBNS and DAQD on Omnidirectional Hexapod Locomotion Task. N.B.: In Model-Based Novelty Search, the transferred individuals are the most novel.

5.2.3.3 Ball In Cup Task

Finally, the results obtained on the Ball In Cup task are analyzed. Results are obtained on a total of 25 000 evaluations and 10 repetitions for each method considered. Figure 5.16 shows that Model-Based Novelty Search reaches a final coverage of 0.19, Novelty Search reaches a final coverage of 0.16 and Dynamics-Aware Quality-Diversity reaches a final coverage of 0.12, which represent respectively an increase of 30% and 90% in terms of final coverage for the considered budget. Nevertheless, it is interesting to note that here Novelty Search reaches a coverage that is higher

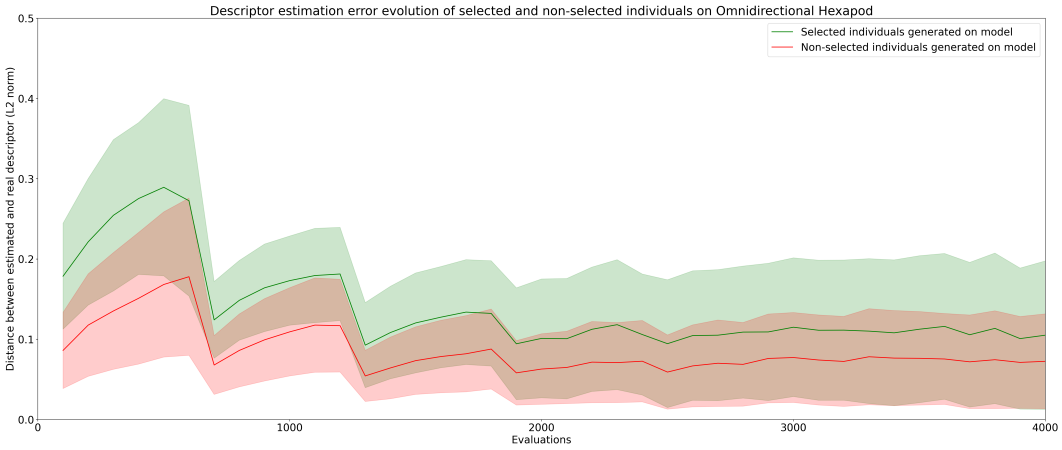


Figure 5.13: Evolution of selected vs non-selected individuals estimated behavioral descriptor error with MBNS on Omnidirectional Hexapod Locomotion Task

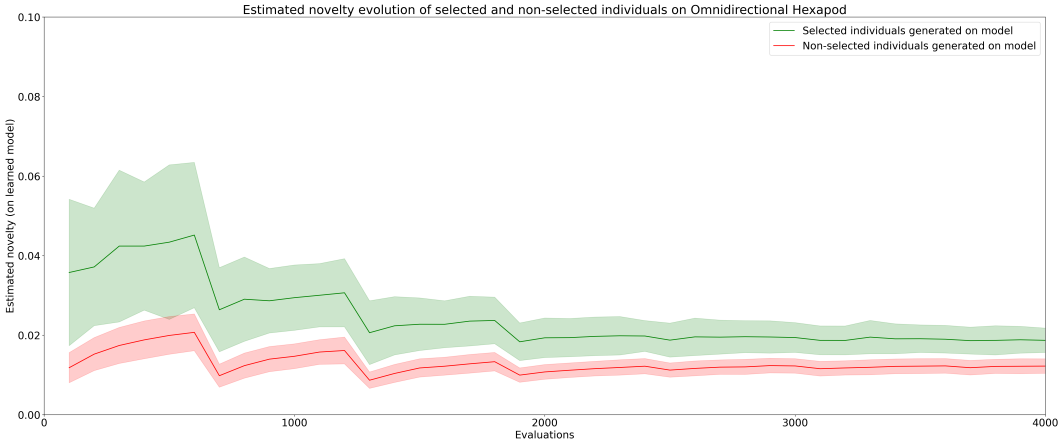


Figure 5.14: Evolution of selected vs non-selected individuals model novelty with MBNS on Omnidirectional Hexapod Locomotion Task

than the one reached by Dynamics-Aware Quality-Diversity. As the Ball In Cup task is a task with sparse dynamics, meaning that some transitions appear scarcely in the transition function T, it was expected that more exploratory methods would perform better as observed when looking at final coverages. Indeed, looking at the corresponding coverage shown for each of the considered algorithms on Figure 5.17, it is shown that both QD methods have a lower archive density on the lower part of the z-axis, corresponding to end positions that lift the ball above the cup. On the other hand, Novelty Search based methods have a more densely populated archive in all the directions, with Model-Based Novelty Search being the most dense of the two.

Figure 5.18 shows the evolution of the coverage for each of the four considered

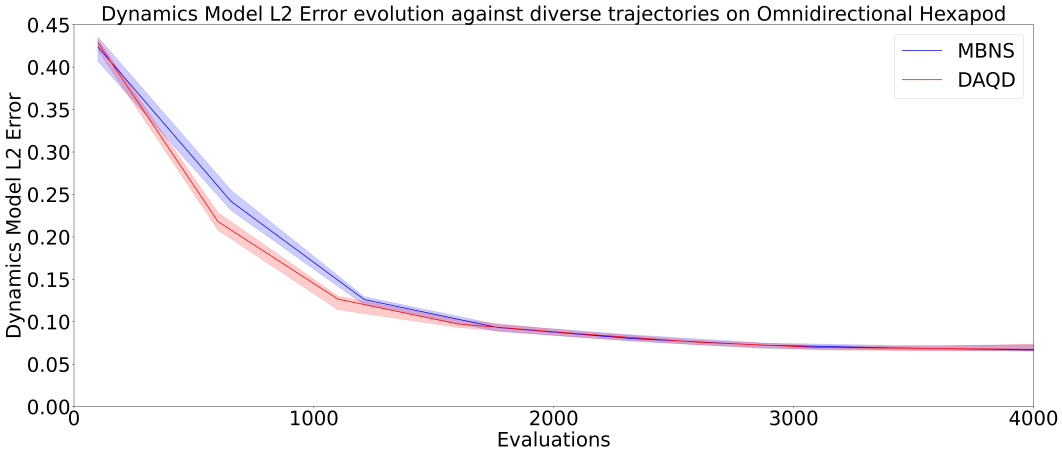


Figure 5.15: Evolution of the Dynamics Model L2 Error on a diverse set of 100 unseen trajectories for Dynamics Models trained independently with MBNS and DAQD on Omnidirectional Hexapod Locomotion Task

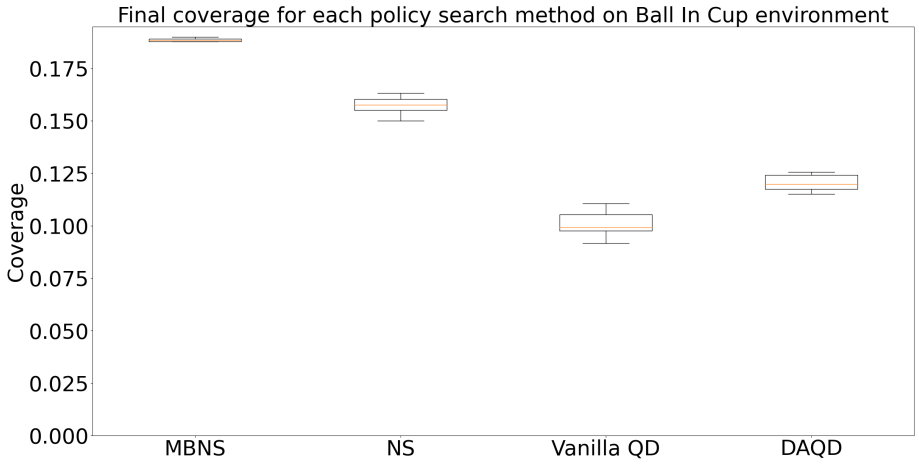
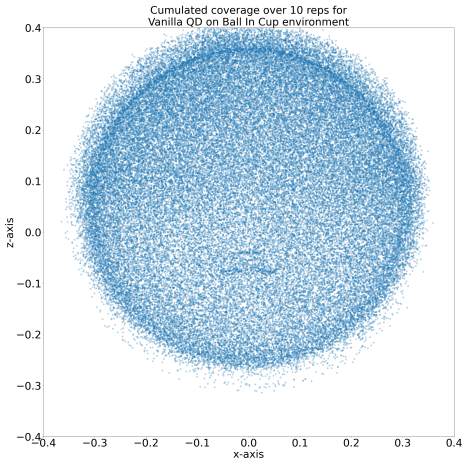
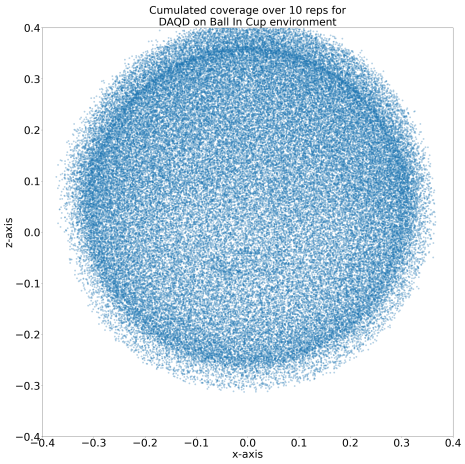


Figure 5.16: Final Coverage of Diversity Methods on Ball In Cup task

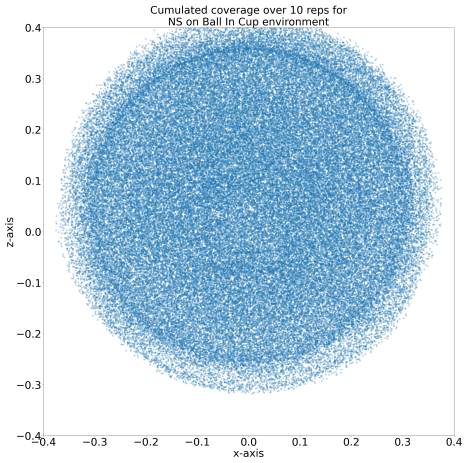
Diversity Algorithms on the Ball In Cup environment. Model-Based Novelty Search is 2 times faster than Dynamics-Aware Quality-Diversity to reach its final coverage as it attains 12% of coverage at 12 800 evaluations, but only 30% faster than Novelty Search to reach its final coverage of 16% of the Outcome Space. Model-Based Novelty Search remains more sample-efficient than Novelty Search and Dynamics-Aware Quality-Diversity, but not as much as we would expect. This could be explained by the fact that the transition dynamics of the Ball In Cup task is more complex than those of the two other environments, as we recall that Ball In Cup was the least consistent environment as shown in Chapter 3 meaning that actions yield inconsistent behaviors depending on the state the system is in, most likely leading to an increased



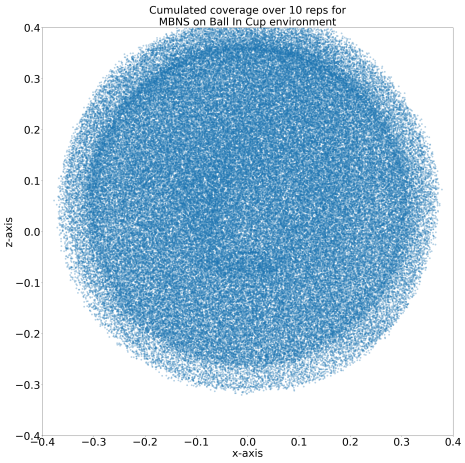
(a) QD cumulated coverage



(b) DAQD cumulated coverage



(c) NS cumulated coverage



(d) MBNS cumulated coverage

Figure 5.17: Cumulated coverage over 10 repetitions for each of the Diversity Algorithms considered on the Ball In Cup task

error in the descriptor estimation for the model evaluated individuals.

Indeed, when looking at Figure 5.12, two things are observed. Firstly, just like on the two previous environments, Model-Based Novelty Search over-exploits the model when it is not yet enough trained. Nevertheless, it seems to affect slightly the performance for the first 1500 evaluations but not as much as on the Hexapod environment for example. The second remark that can be made is that the descriptor estimation errors for the transferred individuals, for both Model-Based Novelty Search and Dynamics-Aware Quality-Diversity, remain high through the whole policy

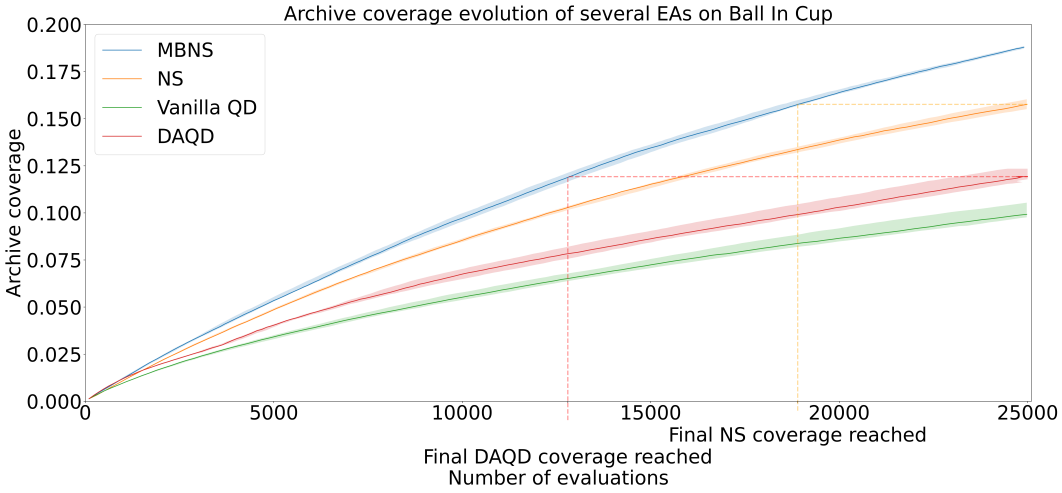


Figure 5.18: Coverage of Diversity Methods on Ball In Cup task depending on number of evaluations

search process. Indeed, error remains around 0.3, which is high for the environment as it has limits that are in the $[-0.4, 0.4]$ across each dimension. As stated earlier, this is most likely due to the fact that this task is hard to model. Indeed, actions do not have the same impact at all in different regions of the State-Space, thus making generalization for the model way harder than in more consistent environments like the Two-Wheeled Robot Navigation task or the Omnidirectional Hexapod. After 3000 evaluations, looking at the median descriptor estimation error of transferred individuals for Model-Based Novelty Search compared to that of Dynamics-Aware Quality-Diversity, Model-Based Novelty Search descriptor estimation error is smaller and continue to decreases until exhaustion of the allocated budget, showing the ability of Model-Based Novelty Search to explore more thoroughly the complex dynamic of this task.

Nevertheless, the descriptor estimation error of selected individuals in Model-Based Novelty Search was significantly higher than that of Dynamics-Aware Quality-Diversity on the first 1500 evaluations, even if the performance of the algorithm is not highly impacted on this environment. Figures 5.20 and 5.21 show that the gap between selected and non-selected individuals reaches as high as almost thrice the descriptor estimation error. This is however quickly reduced after 1500 evaluations, descriptor estimation errors being identical after that while higher novelty is maintained in the selected individuals, ensuring a good functioning of Model-Based Novelty Search once the model has enough training data. Finally, Figure 5.22 shows that the L2 error against a set of diverse trajectories is similar for models trained both using Dynamics-Aware Quality-Diversity and Model-Based Novelty Search, show-

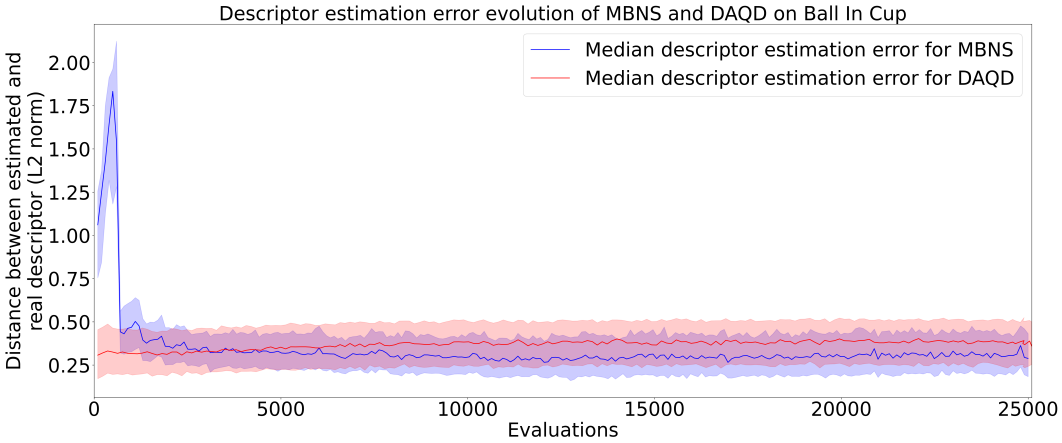


Figure 5.19: Comparison of the evolution of all transferred individuals estimated behavioral descriptor error between MBNS and DAQD on Ball In Cup task. N.B.: In Model-Based Novelty Search, the transferred individuals are the most novel.

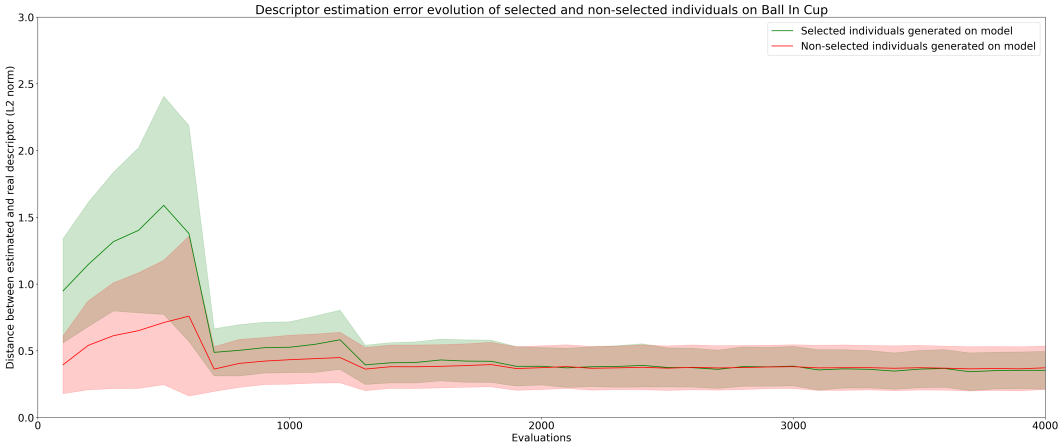


Figure 5.20: Evolution of selected vs non-selected individuals estimated behavioral descriptor error with MBNS on Ball In Cup task

ing that the model is effectively being over-exploited through the novelty objective initially.

5.3 Conclusions

In this chapter, we proposed a new Model-Based Diversity Algorithm based on the Novelty Search algorithm called Model-Based Novelty Search. We proposed to perform a Novelty Search loop directly on a learned dynamics model, using the model to estimate the behaviors of the individuals generated through selection-variation, and to select for transfer onto the real system the most novel individuals found *w.r.t.*

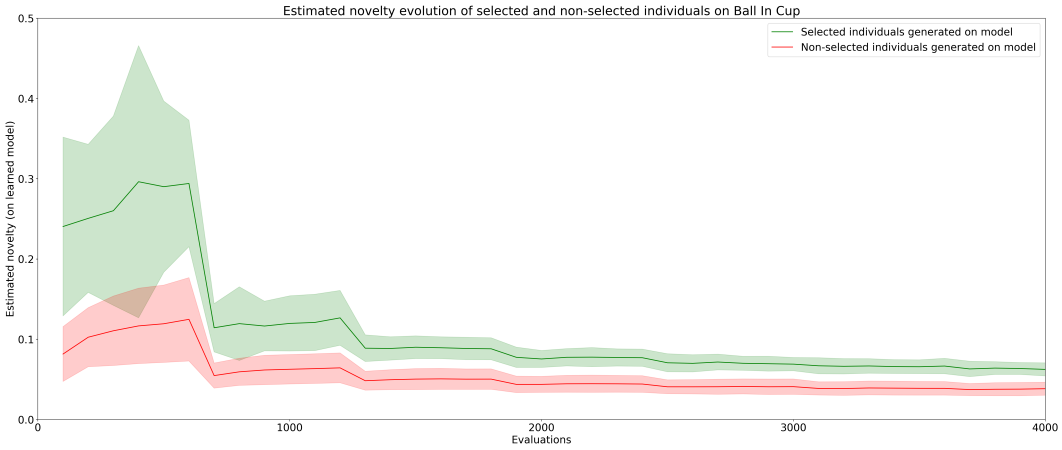


Figure 5.21: Evolution of selected vs non-selected individuals model novelty with MBNS on Ball In Cup task

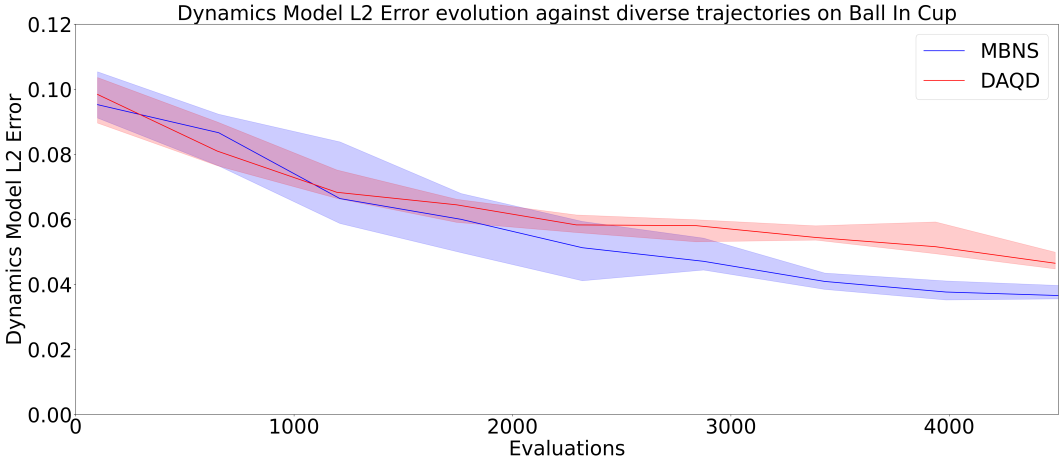


Figure 5.22: Evolution of the Dynamics Model L2 Error on a diverse set of 100 unseen trajectories for Dynamics Models trained independently with MBNS and DAQD on Ball In Cup task

all the model evaluations performed and the current real Novelty Search archive $\mathcal{A}_{\Pi}$. We proposed to compare this method against three baselines, two regular Diversity Algorithms, Novelty Search and Quality-Diversity with an unstructured archive, and a state-of-the-art Model-Based Diversity Algorithm Dynamics-Aware Quality-Diversity.

We measured the performance of each algorithm in terms of coverage of the user-defined Outcome Space $\mathcal{B}$ on three different environments, an environment with pretty uniform dynamics that are easy to learn, a benchmark environment taken from the Dynamics-Aware Quality-Diversity paper and an environment with sparse interactions in its dynamics making it harder to learn a dynamics model for. We

showed that the proposed algorithm, Model-Based Novelty Search, outperformed all other 3 baselines in all three environments on the considered evaluations budgets in terms of coverage. The increase in sample-efficiency against Novelty Search ranged from a 30% increase to a 4 times increase in speed to reach a similar coverage. The increase in sample-efficiency against Dynamics-Aware Quality-Diversity ranged from a 50% increase to a 2.5 times increase in speed to reach a similar coverage.

Finally, we analyzed the model prediction error and showed that Model-Based Novelty Search tends to over-exploit the dynamics model when it is not yet accurate enough, a known problem for novelty-driven methods that could be addressed using uncertainty metrics obtained using dynamics model ensembles. Indeed, one could argue that over-exploiting the model is not a problem if it brings informative data that may help train the model to make more accurate predictions faster, however using uncertainty metrics as discussed in the next chapter would yield better results. Indeed, using novelty directly does not necessarily gather the most informative data, as most of the individual's trajectory might remain in the already known training support. Inversely, an individual could be not novel at all but be very informative if some parts of its trajectory end up in unseen parts of the environment dynamics. Future works addressing this over-exploitation issue could lead to a new Model-Based Novelty Search method that is more robust to the environments dynamics.

Chapter 6

Discussion

6.1 Models and priors

In Chapter 4, we proposed to use two different model representations, Spatial Random Fields and Non-Linear Feed-Forward Neural Networks. Such model representations are useful as they are relatively simple and can represent efficiently a system dynamics, the first representation being able to model Gaussian transitions and spatial correlations in the transition function, the second being able to model non-linearities and handle more complex transition functions. However, such representations quickly faced limitations when encountering robotic systems with a bigger State-Action Space.

Indeed, even the most powerful representation of the two proposed, Feed-Forward Neural Networks, failed to incorporate and respect the underlying physical principles governing the system. However integrating the priors on system dynamics in the learning pipeline can improve several aspects of the learning process, especially sample-efficiency [140]. Novel approaches like Physics-Informed Neural Networks [33] help integrating seamlessly such constraints directly in the model learning. Indeed, Physics-Informed Neural Networks is an approach that integrates the strengths of neural networks with a strong emphasis on physics-based constraints. Unlike classic neural networks that may lack interpretability and struggle with generalization to new scenarios, Physics-Informed Neural Networks leverage prior knowledge of the system's physics during training.

By embedding the governing physical laws directly into the neural network architecture, Physics-Informed Neural Networks can learn more accurately the system

dynamics with less training data and generalize better outside of the training data [33]. Indeed, Physics-Informed Neural Networks are trained to both fit observed data and reduce the residual of a Partial Differential Equation governing the system dynamics. Nevertheless, such methods require prior knowledge over the task dynamics and can also be hard to engineer depending on the considered robotic task.

Hopefully, most robotic tasks dynamics have been broadly studied and equations describing the various considered systems do exist, even if they might include many parameters. Incorporating them into Physics-Informed Neural Networks could be lengthy, but could help reduce vastly the data cost of learning an accurate dynamics model from scratch. In the context of this thesis, incorporating such priors in the models could help enhancing the proposed 0DAB algorithm, by using random dynamics models that are closer to the reality as they directly integrate some knowledge of the target system, without needing any data. But the cost of developing such techniques should be properly evaluated against using directly simulators and approaches inspired from Domain Randomization. Moreover, Physics-Informed Neural Network offer all the aforementioned advantages at the cost of an increased computational cost for training the model, as multiple and different in nature training objectives are optimized at once.

6.2 Promoting Novelty

In Chapter 4, we proposed two novelty metrics on ensembles of environments. Such novelty metrics aimed at finding generally novel individuals, *i.e.* individuals that are more novel than most other individuals on all the considered environments. We proposed to use N_{min}, the minimum of normalized novelty obtained on all environments, and N_{sum} the sum of normalized novelty obtained on all environments.

We supposed that such novelty metrics would promote generally novel individuals, and it did seem to as coverage on most environments was uniform. Nevertheless, it is hard to determine if the obtained individuals did reach our goal of being novel on unseen environments in the same environment distribution as the one the policies were found on. Such behaviors can be found in various ways, and other authors have tried to reach that goal in different ways.

Salehi *et al.* [139] propose in their algorithm FAERY to explicitly create a prior population of individuals with the objective of making that prior population both versatile (the number of solutions derived from the prior population) and adaptive

(the number of mutations required to reach a solution from the prior population). Those two fitness functions are optimized using the multi-objective optimization method NSGA-II [36].

The objectives here are much more explicit than ours and could lead to interesting results if instead of optimizing the population through either N_{min} or N_{max}, such an optimization scheme was used in 0DAB. Indeed, the idea is that using such methods could make up better for the discrepancies existing between the source and target domains, as the prior population could capture more generalist characteristics compared to our proposed novelty metrics.

6.3 Long horizon prediction

Across chapters 3, 4 and 5 we observed in several occurrences that a long prediction horizon is detrimental to the overall trajectory prediction. Indeed, the type of models we used as dynamics models, as they work in a recursive manner where the previous prediction is fed back into the prediction model, compound error over the prediction horizon. Thus, the longer the horizon, the more inaccurate the prediction, which is problematic in the case of Diversity Algorithms.

Indeed, as Diversity Algorithms compute the novelty and fitness of an individual over a complete episode, rather than computing a reward step-by-step like in Reinforcement Learning, using predictive models in this context requires to accurately predict the complete trajectory of the candidate solutions. Moreover, in the context of the robotic tasks that we considered in this thesis, the final position of the candidate solutions is often used as the behavioral descriptor. This amplifies even more the effect of a model whose error increases overtime as the Outcome Space is exactly the most error-prone part of the predicted trajectory of the model-evaluated individual.

One could thus propose to directly predict the behavioral descriptor of an individual, but the mapping between the parameter space and the outcome space is often hard to learn, and using a dynamics model which directly represents the considered Markov Decision Process transition function did increase considerably the sample efficiency of model-based Quality Diversity algorithms [98]. Nevertheless, it does not mean that the dynamics models that were used up to this point in the literature are ideal for the considered environments and prediction horizons.

Others model representation exists to make more accurate predictions over long horizons, leveraging their ability to capture complex patterns and dependencies in temporal data. Most of these representation take inspiration from Recurrent Neural Networks [108]. Recurrent Neural Networks are designed to handle sequential data by maintaining hidden states that capture information from previous time steps. This makes Recurrent Neural Networks suitable for long-term prediction tasks as they can retain context over extended periods. However, traditional Recurrent Neural Networks suffer from the vanishing gradient problem [58], limiting their effectiveness in capturing long-range dependencies.

To address the vanishing gradient problem in Recurrent Neural Networks, Long Short-Term Memory (LSTM) networks [59, 175] were introduced. LSTMs incorporate memory cells and gating mechanisms, allowing them to capture and store information over longer time horizons. This makes LSTMs well-suited for tasks requiring accurate long-term predictions, such as dynamics modeling [137, 171]. Such model representations have thus been used in time series forecasting with higher accuracy than regular Neural Networks [15]. Other model representations exploiting recurrence have been proposed and used in robotics task dynamics learning contexts like RSSM [57] that demonstrated that using both a stochastic and a deterministic inference path in their model was crucial to long horizon accurate predictions. Such models have also been successfully applied to real robotic systems with very high sample-efficiency [170].

Nevertheless, one should not undermine choosing the right model representation for the right problem. Indeed, in the case of physical systems when all the relevant variables are available and stochasticity is non-existent or relatively low, long term predictions should be accurate enough. Some authors proposed using Generative Models for such long term prediction horizons and demonstrated results that even surpassed that of LSTMs or other recurrent architectures by several range of magnitude over prediction horizons of several hundred time-steps [55].

Another interesting approach is to leverage a mechanism called attention [163]. Transformers are a type of model representation that use this mechanism integrated into neural network architectures to focus on specific parts of the input sequence when making predictions. This selective attention mechanism enhances the model ability to capture relevant long-term dependencies, making it particularly useful for dynamics modeling tasks. An increasing number of works uses such models and some integrate them as dynamics models with promising results in applications like industrial settings [158] and inverse kinematics modeling of soft robots [4] and

grasping [126]. While we did not explore most of the aforementioned methods in this thesis, some deserve to be explored thoroughly as they could deal with some of the limitations observed in our works.

6.4 Information Guided Search

In Chapter 5, we proposed to guide a Model-Based Novelty Search approach using a simple novelty metric transferring the most novel individuals found on the model onto the real system. Such approach proved to enhance greatly the sample efficiency of the Vanilla Novelty Search algorithm. We also noticed that the behavioral descriptor estimation on the model was sometimes very far from the real behavioral descriptor. This can be imputed to the model not being trained with enough data, not being the best representation for modeling time series and to the novelty focused approach selecting behaviors at the limit and even out of the training data support.

The model not being trained with enough data is inherent to the problem at hand, so this reason can be ignored as of now. The model representation problem has been addressed in the previous section. Finally remains the question of the approach used to select the individuals generated using the model to transfer. Guiding the search through the objective of gathering the best data to diminish the model prediction error could be interesting. Indeed, it could kill two birds with one stone: ameliorate the model performance and find novel individuals. Most novel individuals behavioral descriptors should end in unexplored regions of the Outcome Space, and data from outside the training data support is exactly the one that could enhance the model prediction performance.

As described in section 2.4.2.4, some dynamics model representations allow to compute such an information gain metric from disagreement between the different predictions outputted by each model in a model ensemble [144]. Indeed, the epistemic uncertainty that can be dissociated from the aleatoric uncertainty using ensemble disagreement is a direct proxy for information gain, and maximising it has been used in the Reinforcement Learning community as an self-supervision exploration objective [128, 144]. Authors thus proposed to use such a metric in a Quality-Diversity context [99] with the goal of enhancing exploration. However, they found that such metrics, when used for optimization on the learned model skewed the distribution of the solutions obtained on the model towards that single objective.

Using such metrics can be also be deceptive. For example, in the case where

the behavioral descriptor is only a part of the end state of the individual trajectory, if someone were to use the overall information gain over the whole trajectory, the two objectives could oppose themselves. Indeed, if the behavioral descriptor is in the already explored region of the Behavioral Space, but that most part of the trajectory are outside of the training data support, the information gain based metric could suggest to select that individual while the novelty metric would suggest to discard it. Alignment in the considered observer function on the trajectory, both on the behavioral side and on the information gain seems crucial and might explain why Lim *et al.* [99] had results that seemed to indicate that disagreement metrics could be unfruitful, as they computed the disagreement over the whole trajectory and the behavioral descriptor solely as the final Cartesian position of their robot, a clear misalignment in objectives. Optimizing for information gain directly thus does not seem to be a particularly good idea, and more well-thought methods should be explored, but not before studying the impact of alignment or misalignment between objectives that might seem similar in the first place but can end up pretty opposed if one is not careful enough in their design.

Chapter 7

Conclusion

In this Ph.D thesis, several ways of decreasing the number of evaluations used by Novelty Search to cover a user-defined Behavioral Space were studied. We started by explaining the reasons leading scientists to use Model-Based techniques, the main one being the Reality Gap. Several possibilities to overcome it exist, but Model-Based approaches set a general and natural framework in which the robot iteratively interacts with its environment, plans in imagination using its model of the world its next course of action and takes action again, updating its view of its environment.

A crucial point of such techniques is the number of interactions with the environment used, that should be lessen as much as possible. In the realm of Evolutionary Algorithms, sample usage is often very important, and techniques reducing the sample-efficiency not explored as much as they could be. We thus studied to focus on the Novelty Search algorithm, and to propose different ways of enhancing its sample-efficiency, as well as the sample-efficiency of all model-based techniques through a generic study of the impact of initial model training data.

7.1 Initial Data Gathering techniques impact on MBPS

Indeed, we firslty proposed to study the impact of initial training data on Model-Based Policy Search algorithms. To do so coherently, we proposed a metric evaluating a Markov Decision Process consistency of actions over its State-Space. We then demonstrated a link existing between the consistency and the prediction error of the dynamics model when it is trained using an appropriated initial data gathering method. This was empirically measured on five different robotic tasks of

varying consistency metric. Clear conclusions cannot be made very clearly on the three environments with a medium consistency, but the significantly more consistent and significantly less consistent environments did show that this metric could help characterize the environment dynamics. Nevertheless, enhancing that metric to be more precise could help identify better the environments dynamics and chose a suited degree of time correlation in the initial data gathering technique to bootstrap the model.

The impact of the initial model training has been studied on two state-of-the-art Model-Based Policy Search algorithms: DAQD [98] and PETS [26]. On DAQD, aside from small coverage difference at the first iteration, no significant impact was observed. The evaluation budget given to DAQD being very high compared to that of regular Model-Based techniques, it is not really surprising that the model initialization impact does not propagates on the complete run since the model is being retrained with much more samples than the initialization budget considered after a few hundred evaluations. However, on PETS the impact was much more clear as the observed return differences after the first episode were up to 10 times higher when using a suited initial data gathering technique. The actual impact from model initialization thus seems to vary greatly depending on the evaluation budget, the optimization method used and the overall dynamics of the considered task.

7.2 0DAB: Zero-Shot Diverse Archive Bootstrapping

Secondly, we proposed an algorithm that could help bootstrap the population of a Novelty Search routine better than randomly parameterized policies. 0DAB, as it is dubbed, combines random dynamics models ensembles with Novelty Search to find individuals that are novel across the whole model ensemble. On a simple robotic navigation task and with a suited random dynamics model representation, 0DAB was able to increase the initial coverage of up to 35% when compared to randomly parameterized policies, also reducing the total number of samples required to reach complete coverage of the Outcome Space by several hundred evaluations.

However, when considering more complex dynamics like that of an hexapod robot locomotion task, 0DAB failed to increase the initial coverage and to reduce the number of evaluations to reach the target coverage of the Behavioral Space. Actually, even when the initial coverage was slightly improved, the coverage evolution was highly impacted by the bias introduced in the initial population compared to a completely unbiased population generated through randomly parameterized policies.

No model representation was found that could capture the dynamics of the target system to some extent, even though progress seemed to be made when increasing the prediction horizon on the model ensembles and increasing the size of the ensemble.

The potential reasons of this failure were analyzed through the scope of the Outcome Space on the random dynamics models used, and it was identified that the diverse set of solutions found by Novelty Search on the real system, when transferred onto the random dynamics models, were collapsing to small regions of the Behavioral Space on every model of the considered ensemble. This makes selection for transfer of interesting solutions very hard as all diverse solutions actually fell in the same regions of the Outcome Space and highlighted the need to use better suited model representations or a different selection or optimization mechanism on the random dynamics model ensemble to successfully overcome such limitations.

7.3 Model-Based Novelty Search

Finally, a new Model-Based Diversity Algorithm, simply dubbed Model-Based Novelty Search was proposed. A dynamics model of the considered Markov Decision Process transition function is learned and used to perform in imagination a Novelty Search loop. At the end of this loop, whose budget is adaptive on the current total evaluation budget spent, the generated solutions that are estimated to be the most novel through evaluation of their behaviors on the dynamics model are transferred on the real system. The transferred solutions are then evaluated on the real system, and used as the starting population for the next Novelty Search loop on the learned dynamics model.

This method was compared against three baselines, Novelty Search, Vanilla Quality-Diversity and Dynamics-Aware Quality Diversity. The performance of each algorithm in terms of coverage was measured on three different environments, an hexapod locomotion task, a robotic navigation task and a ball-in-cup task. The proposed algorithm outperformed in terms of coverage all other three baselines on all the three considered environments on the evaluations budgets defined. Moreover, the number of evaluations needed to reach a certain coverage against Novelty Search was reduced by 30% up to 75% depending on the considered environment. Against Dynamics-Aware Quality Diversity, the state-of-the-art Model-Based Diversity Algorithm using a dynamics model, the number of samples required to reach its final coverage was reduced from 30% up to 50% on the different robotic tasks we considered. To understand better the observed coverage evolutions depending on the

number of evaluations, the model prediction error was analyzed and showed that Model-Based Novelty Search tend to over-exploit the dynamics model when it is not accurate enough in its early stages. Model-Based Novelty Search thus paves the way to sample-efficient exploration methods by setting a simple framework that could be enhanced in many different aspects.

Bibliography

[1] Pieter Abbeel, Varun Ganapathi, and Andrew Ng. Learning vehicular dynamics, with application to modeling helicopters. *Advances in Neural Information Processing Systems*, 18, 2005.

[2] Hervé Abdi. Coefficient of variation. *Encyclopedia of research design*, 1(5), 2010.

[3] Brian Acosta, William Yang, and Michael Posa. Validating robotics simulators on real-world impacts. *IEEE Robotics and Automation Letters*, 7(3):6471–6478, 2022.

[4] Abdelrahman Alkhodary and Berke Gur. Kinematics transformer: Solving the inverse modeling problem of soft robots using transformers. *arXiv preprint arXiv:2211.06643*, 2022.

[5] Franklin Allen and Risto Karjalainen. Using genetic algorithms to find technical trading rules. *Journal of financial Economics*, 51(2):245–271, 1999.

[6] Chae H An, Christopher G Atkeson, and John M Hollerbach. *Model-based control of a robot manipulator*. MIT press, 1988.

[7] Christopher G Atkeson and Juan Carlos Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of international conference on robotics and automation*, volume 4, pages 3557–3564. IEEE, 1997.

[8] Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20, 1997.

[9] Andrew G Barto. Intrinsic motivation and reinforcement learning. *Intrinsically motivated learning in natural and artificial systems*, pages 17–47, 2013.

[10] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[11] Bernhard Bermeitinger, Tomas Hrycej, and Siegfried Handschuh. Representational capacity of deep neural networks–a computing study. *arXiv preprint arXiv:1907.08475*, 2019.

[12] Aude Billard and Danica Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446):eaat8414, 2019.

[13] Verónica Bolón-Canedo and Amparo Alonso-Betanzos. Ensembles for feature selection: A review and future trends. *Information Fusion*, 52:1–12, 2019.

[14] Gary Boone. Efficient reinforcement learning: Model-based acrobot control. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 229–234. IEEE, 1997.

[15] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[16] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[17] Nicolas Bredeche, Jean-Marc Montanier, Wenguo Liu, and Alan FT Winfield. Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129, 2012.

[18] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[19] Eduardo F Camacho, Carlos Bordons, Eduardo F Camacho, and Carlos Bordons. *Model predictive controllers*. Springer, 2007.

[20] Jacopo Castellini, Frans A Oliehoek, Rahul Savani, and Shimon Whiteson. The representational capacity of action-value networks for multi-agent reinforcement learning. *arXiv preprint arXiv:1902.07497*, 2019.

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

[22] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[23] Ran Cheng, Cheng He, Yaochu Jin, and Xin Yao. Model-based evolutionary algorithms: a short survey. *Complex & Intelligent Systems*, 4(4):283–292, 2018.

[24] Ran Cheng, Yaochu Jin, Kaname Narukawa, and Bernhard Sendhoff. A multiobjective evolutionary algorithm using gaussian process-based inverse modeling. *IEEE Transactions on Evolutionary Computation*, 19(6):838–856, 2015.

[25] Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17, 2004.

[26] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

[27] Tinkle Chugh, Karthik Sindhya, Jussi Hakanen, and Kaisa Miettinen. A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing*, 23:3137–3166, 2019.

[28] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431, 2021.

[29] Guillaume Corriveau, Raynald Guilbault, Antoine Tahan, and Robert Sabourin. Bayesian network as an adaptive parameter setting approach for genetic algorithms. *Complex & Intelligent Systems*, 2:1–22, 2016.

[30] Erwin Coumans. Bullet physics simulation. *ACM SIGGRAPH 2015 Courses*, 2015.

[31] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

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

[33] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

[34] Charles Darwin and William F Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt New York, 2009.

[35] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134:19–67, 2005.

[36] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

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

[38] Marc Peter Deisenroth, Roberto Calandra, André Seyfarth, and Jan Peters. Toward fast policy search for learning legged locomotion. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1787–1792. IEEE, 2012.

[39] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.

[40] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.

[41] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *International Conference on Machine Learning*, pages 1184–1193. PMLR, 2018.

[42] Stephane Doncieux, Nicolas Bredeche, Jean-Baptiste Mouret, and Agoston E Eiben. Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2:4, 2015.

[43] Stephane Doncieux, Alban Laflaquière, and Alexandre Coninx. Novelty search: a theoretical perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 99–106, 2019.

[44] Stephane Doncieux, Giuseppe Paolo, Alban Laflaquière, and Alexandre Coninx. Novelty search makes evolvability inevitable. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 85–93, 2020.

[45] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM review*, 41(4):637–676, 1999.

[46] Agoston E Eiben and Jim Smith. From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482, 2015.

[47] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015.

[48] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R Arabnia. A brief review of domain adaptation. *Advances in data science and information engineering: proceedings from ICDATA 2020 and IKE 2020*, pages 877–894, 2021.

[49] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. Data-efficient design exploration through surrogate-assisted illumination. *Evolutionary computation*, 26(3):381–410, 2018.

[50] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[51] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, page 25, 2016.

[52] Ioannis Giagkiozis and Peter J Fleming. Pareto front estimation for decision making. *Evolutionary computation*, 22(4):651–678, 2014.

[53] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. Devising effective novelty search algorithms: A comprehensive empirical study. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 943–950, 2015.

[54] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7:115–144, 2013.

[55] Sebastian Gomez-Gonzalez, Sergey Prokudin, Bernhard Schölkopf, and Jan Peters. Real time trajectory prediction using deep conditional generative models. *IEEE Robotics and Automation Letters*, 5(2):970–976, 2020.

[56] Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

[57] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[58] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[59] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[60] Dionissios T Hristopulos. *Random fields for spatial data modeling*. Springer, 2020.

[61] Eduardo Raul Hruschka, Ricardo JGB Campello, Alex A Freitas, et al. A survey of evolutionary algorithms for clustering. *IEEE Transactions on systems, man, and cybernetics, Part C (applications and reviews)*, 39(2):133–155, 2009.

[62] Yanrong Hu and Simon X Yang. A knowledge based genetic algorithm for path planning of a mobile robot. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 4350–4355. IEEE, 2004.

[63] Wenbing Huang, Deli Zhao, Fuchun Sun, Huaping Liu, and Edward Chang. Scalable gaussian process regression using deep neural networks. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[64] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021.

[65] Vugar E Ismailov. A three layer neural network can represent any multivariate function. *Journal of Mathematical Analysis and Applications*, 523(1):127096, 2023.

[66] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995.

[67] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[68] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

[69] Jing Jiang. A literature survey on domain adaptation of statistical classifiers. *URL: http://sifaka. cs. uiuc. edu/jiang4/domainadaptation/survey*, 3(1-12):3, 2008.

[70] Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing*, 9(1):3–12, 2005.

[71] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[72] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, 2022.

[73] Juraj Kabzan, Lukas Hewing, Alexander Liniger, and Melanie N Zeilinger. Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, 2019.

[74] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[75] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[76] Rituraj Kaushik, Pierre Desreumaux, and Jean-Baptiste Mouret. Adaptive prior selection for repertoire-based online adaptation in robotics. *Frontiers in Robotics and AI*, 6:151, 2020.

[77] Leon Keller, Daniel Tanneberg, Svenja Stark, and Jan Peters. Model-based quality-diversity search for efficient robot learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9675–9680. IEEE, 2020.

[78] Seungsu Kim, Alexandre Coninx, and Stéphane Doncieux. From exploration to control: learning object manipulation skills through novelty search and local adaptation. *arXiv preprint arXiv:1901.00811*, 2019.

[79] Seungsu Kim, Alexandre Coninx, and Stéphane Doncieux. From exploration to control: learning object manipulation skills through novelty search and local adaptation. *Robotics and Autonomous Systems*, 136:103710, 2021.

[80] Seungsu Kim and Stéphane Doncieux. Learning highly diverse robot throwing movements through quality diversity search. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1177–1178, 2017.

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

[82] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[83] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. Automatic system identification based on coevolution of models and tests. In *2009 IEEE congress on evolutionary computation*, pages 560–567. IEEE, 2009.

[84] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2012.

[85] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[86] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

[87] Chih-Chin Lai and Chuan-Yu Chang. A hierarchical evolutionary algorithm for automatic medical image segmentation. *Expert Systems with Applications*, 36(1):248–259, 2009.

[88] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[89] Pedro Larrañaga and Jose A Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2001.

[90] Jeongseok Lee, Michael X. Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S. Srinivasa, Mike Stilman, and C Karen Liu. Dart: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22):500, 2018.

[91] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[92] Joel Lehman and Kenneth O Stanley. 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, 2011.

[93] Joel Lehman, Kenneth O Stanley, et al. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[94] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information fusion*, 58:52–68, 2020.

[95] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. *Advances in neural information processing systems*, 27, 2014.

[96] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. 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, 2018.

[97] Li Li, Gary G Yen, Avimanyu Sahoo, Liang Chang, and Tianlong Gu. On the estimation of pareto front and dimensional similarity in many-objective evolutionary algorithm. *Information Sciences*, 563:375–400, 2021.

[98] Bryan Lim, Luca Grillotti, Lorenzo Bernasconi, and Antoine Cully. Dynamics-aware quality-diversity for efficient learning of skill repertoires. *arXiv preprint arXiv:2109.08522*, 2021.

[99] Bryan Lim, Alexander Reichenbach, and Antoine Cully. Learning to walk autonomously via reset-free quality-diversity. *arXiv preprint arXiv:2204.03655*, 2022.

[100] Hod Lipson and Jordan B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[101] Xiaofeng Liu, Chaehwa Yoo, Fangxu Xing, Hyejin Oh, Georges El Fakhri, Je-Won Kang, Jonghye Woo, et al. Deep unsupervised domain adaptation: A review of recent advances and perspectives. *APSIPA Transactions on Signal and Information Processing*, 11(1), 2022.

[102] Daniel J Lizotte, Tao Wang, Michael H Bowling, and Dale Schuurmans. Automatic gait optimization with gaussian process regression. In *IJCAI*, volume 7, pages 944–949, 2007.

[103] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[104] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[105] Ruben Martinez-Cantin, Nando de Freitas, Arnaud Doucet, and José A Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and systems*, volume 3, pages 321–328, 2007.

[106] Maja J Mataric. Reward functions for accelerated learning. In *Machine learning proceedings 1994*, pages 181–189. Elsevier, 1994.

[107] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 734–743. PMLR, 2018.

[108] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

[109] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR, 2020.

[110] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[111] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[112] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[113] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[114] Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13:103–130, 1993.

[115] Aurélien Morel, Yakumo Kunimoto, Alex Coninx, and Stéphane Doncieux. Automatic acquisition of a repertoire of diverse grasping trajectories through behavior shaping and novelty search. In *IEEE International Conference on Robotics and Automation 2022*, 2022.

[116] Jun Morimoto and Christopher G Atkeson. Nonparametric representation of an approximated poincaré map for learning biped locomotion. *Autonomous Robots*, 27(2):131–144, 2009.

[117] Peter Mörters and Yuval Peres. *Brownian motion*, volume 30. Cambridge University Press, 2010.

[118] Amirhosein Mosavi, Yaser Faghan, Pedram Ghamisi, Puhong Duan, Sina Faizollahzadeh Ardabili, Ely Salwana, and Shahab S Band. Comprehensive review of deep reinforcement learning methods and applications in economics. *Mathematics*, 8(10):1640, 2020.

[119] J-B Mouret and Stéphane Doncieux. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1):91–133, 2012.

[120] Jean-Baptiste Mouret and Konstantinos Chatzilygeroudis. 20 years of reality gap: a few thoughts about simulators in evolutionary robotics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1121–1124, 2017.

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

[122] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. 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, 2018.

[123] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[124] Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz E Khan, Anirudh Jain, Runa Eschenhagen, Richard E Turner, and Rio Yokota. Practical deep learning with bayesian principles. *Advances in neural information processing systems*, 32, 2019.

[125] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[126] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[127] Giuseppe Paolo, Alban Laflaquiere, Alexandre Coninx, and Stephane Doncieux. Unsupervised learning and exploration of reachable outcome space. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2379–2385. IEEE, 2020.

[128] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.

[129] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018*

IEEE international conference on robotics and automation (ICRA), pages 3803–3810. IEEE, 2018.

[130] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. *arXiv preprint arXiv:2008.06389*, 2020.

[131] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[132] Antonin Ponsich, Antonio Lopez Jaimes, and Carlos A Coello Coello. A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications. *IEEE transactions on evolutionary computation*, 17(3):321–344, 2012.

[133] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[134] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[135] James B Rawlings and Rishi Amrit. Optimizing process economic performance using model predictive control. *Nonlinear Model Predictive Control: Towards New Challenging Applications*, pages 119–138, 2009.

[136] John Rieffel and Jean-Baptiste Mouret. Adaptive and resilient soft tensegrity robots. *Soft robotics*, 5(3):318–329, 2018.

[137] Elmar Rueckert, Moritz Nakatenus, Samuele Tosatto, and Jan Peters. Learning inverse dynamics models in o (n) time with lstm networks. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 811–816. IEEE, 2017.

[138] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. Sim2real view invariant visual servoing by recurrent control. *arXiv preprint arXiv:1712.07642*, 2017.

[139] Achkan Salehi, Alexandre Coninx, and Stephane Doncieux. Few-shot quality-diversity optimization. *IEEE Robotics and Automation Letters*, 7(2):4424–4431, 2022.

[140] Achkan Salehi and Stephane Doncieux. Data-efficient, explainable and safe payload manipulation: An illustration of the advantages of physical priors in model-predictive control. *arXiv preprint arXiv:2303.01563*, 2023.

[141] Jeff Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. *Advances in neural information processing systems*, 9, 1996.

[142] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[143] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[144] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

[145] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

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

[147] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[148] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[149] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[150] Susanne Still and Doina Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 131:139–148, 2012.

[151] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[152] Mohammad Tabatabaei, Jussi Hakanen, Markus Hartikainen, Kaisa Miettinen, and Karthik Sindhya. A survey on handling computationally expensive multiobjective optimization problems using surrogates: non-nature inspired methods. *Structural and Multidisciplinary Optimization*, 52:1–25, 2015.

[153] Chin Sheng Tan, Abhishek Gupta, Yew-Soon Ong, Mahardhika Pratama, Puay Siew Tan, and Siew Kei Lam. Pareto optimization with small data by learning across common objective spaces. *Scientific Reports*, 13(1):7842, 2023.

[154] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[155] Sebastian Thrun. Efficient exploration in reinforcement learning. *Technical Report. Carnegie Mellon University*, 1992.

[156] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[157] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[158] Minh Trinh, Mohamed Behery, Mahmoud Emara, Gerhard Lakemeyer, Simon Storms, and Christian Brecher. Dynamics modeling of industrial robots using transformer networks. In *2022 Sixth IEEE International Conference on Robotic Computing (IRC)*, pages 164–171. IEEE, 2022.

[159] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

[160] Matias Valdenegro-Toro and Daniel Saromo Mori. A deeper look into aleatoric and epistemic uncertainty disentanglement. In *2022 IEEE/CVF Conference on*

Computer Vision and Pattern Recognition Workshops (CVPRW), pages 1508–1516. IEEE, 2022.

[161] Jur Van Den Berg, Stephen Miller, Daniel Duckworth, Humphrey Hu, Andrew Wan, Xiao-Yu Fu, Ken Goldberg, and Pieter Abbeel. Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In *2010 IEEE International Conference on Robotics and Automation*, pages 2074–2081. IEEE, 2010.

[162] Vassiiis Vassiliades and Jean-Baptiste Mouret. Discovering the elite hypervolume by leveraging interspecies correlation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 149–156, 2018.

[163] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[164] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog*, 2:20, 2019.

[165] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[166] Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020.

[167] Lilian Weng. Domain randomization for sim2real transfer, May 2019.

[168] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[169] Benjamin Wilson, David Cappelleri, Timothy W Simpson, and Mary Frecker. Efficient pareto frontier exploration using surrogate approximations. *Optimization and Engineering*, 2:31–50, 2001.

[170] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning*, pages 2226–2240. PMLR, 2023.

[171] Long Xin, Pin Wang, Ching-Yao Chan, Jianyu Chen, Shengbo Eben Li, and Bo Cheng. Intention-aware long horizon trajectory prediction of surrounding

vehicles using dual lstm networks. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1441–1446. IEEE, 2018.

[172] Akihiko Yamaguchi and Christopher G Atkeson. Neural networks and differential dynamic programming for reinforcement learning problems. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5434–5441. IEEE, 2016.

[173] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36, 2021.

[174] Wenhao Yu, C Karen Liu, and Greg Turk. Policy transfer with strategy optimization. *arXiv preprint arXiv:1810.05751*, 2018.

[175] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[176] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020.