Titre: Title:	QRNN-ASNN-CEM/PF: A Sample Efficient Model Predictive Control Method In Model Based Reinforcement Learning for Modeling, Control, and Planning in Robotics
Auteur: Author:	Nicolas Leblanc
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Leblanc, N. (2025). QRNN-ASNN-CEM/PF: A Sample Efficient Model Predictive Control Method In Model Based Reinforcement Learning for Modeling, Control, and Planning in Robotics [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/68406/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/68406/
Directeurs de recherche: Advisors:	Marco Bonizzato, & Isabeau Prémont-Schwarz
Programme: Program:	Génie électrique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

QRNN-ASNN-CEM/PF: A Sample Efficient Model Predictive Control Method In Model Based Reinforcement Learning for Modeling, Control, and Planning in Robotics

NICOLAS LEBLANC
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique
Août 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

QRNN-ASNN-CEM/PF: A Sample Efficient Model Predictive Control Method In Model Based Reinforcement Learning for Modeling, Control, and Planning in Robotics

présenté par **Nicolas LEBLANC**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

David SAUSSIÉ, président
Marco BONIZZATO, membre et directeur de recherche
Isabeau PRÉMONT-SCHWARZ, membre et codirecteur de recherche
Bowen YI, membre

DEDICATION

To my father, who left us so young, this is for you. I hope you well wherever you are in this universe, and I hope you are proud. I love you.

ACKNOWLEDGEMENTS

I want to thank my supervisor, Marco Bonizzato, for allowing me to work with him in a UPIR project during my bachelor's studies, which helped kick-start my Master's, and for the opportunity to pursue this Master's research under his guidance. Thank you for believing in my ambitious plan to finish my Master's in only four terms and for always being there for discussions, ideas, and support.

I would also like to thank my co-supervisor, Isabeau Prémont-Schwarz, for his infinite patience, support, guidance, and knowledge. Without you, this project wouldn't have been what it is, and I wouldn't have learned as much. I am also grateful for the time you took to read my thesis in-depth, as well as for the comments and suggestions you provided.

I want to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for financing my research, as well as the Digital Research Alliance of Canada and Calcul Québec for the computing resources I had access to for running my experiments. Having access to powerful servers allowed me to run many more tests than I could have on any computer in the lab.

I want to thank all my friends from the Cepsum gym—Erwan, Pascale, Abbie, Félix, Clara, and Pierre for being there to talk with me every morning about everything and nothing, and for asking about updates on my Master's. Your support means a lot to me.

To my friends from Génie Physique, including Arnaud P. and Benjamin P.-R., who have remained friends to this day, thank you for our discussions about my project, your suggestions on improving my workflow, and for being great friends.

Thank you to my mother for her support and belief in me.

To my grandmother, thank you for always being there to talk, for your immense belief in me, your support, all the delicious food you cook for me, and for always being there, especially when I needed it the most.

Thank you to my godmother and godfather, as well as their family, for always supporting me, teaching me so many life lessons, and believing in me. I wouldn't be the person I am today, nor would I have been able to complete this Master's without you all.

Thank you to all my fellow lab members for making lunch times fun and full of entertaining discussions. I want to extend special thanks to Rose G.-H. and Lison K. for their guidance, support, and encouragement, particularly during the more challenging times of this Master's

program, when they were available to discuss ideas and offer valuable feedback. I want to thank Juan G. for connecting me with Dr. Prémont-Schwarz. I want to thank Mauricio R. and Théodore de L. for their support and the time they took to discuss various ideas that could be applied to my project. I would like to thank Rima E. for her support and our numerous discussions on various topics. Thank you to all the other lab members.

RÉSUMÉ

L'apprentissage par renforcement utilisant un modèle (MBRL) résout généralement de manière plus efficace des problèmes que l'apprentissage par renforcement sans modèle (MFRL). La commande prédictive (MPC), qui peut être formulée comme un algorithme de MBRL, peut être utilisé pour résoudre des problèmes où l'agent à pour but d'atteindre une position précise. Nous proposons la méthode QRNN-ASNN-CEM/PF pour modéliser, résoudre et planifier des problèmes de contrôle et de robotique avec des actions continues ou discrètes. L'algorithme proposé utilise un réseau de neurones quantile (QRNN) comme un modèle de l'environnement qui prédit une distribution sur les prochains états. De plus, la méthode utilise un réseau de neurones pour générer des actions (ASNN) au lieu d'échantillonner une distribution uniforme, et soit le filtrage particulaire (PF) ou la méthode de l'entropie croisée (CEM) pour optimiser les séquences d'actions dans MPC. Nous comparons QRNN-ASNN-CEM/PF à plusieurs algorithmes qui sont ablations de la méthode proposée, ainsi qu'à de méthodes provenant de la littérature du MBRL, MFRL et de l'optimisation de trajectoire.

ABSTRACT

Model-based reinforcement learning (MBRL) generally solves tasks more sample-efficiently than model-free reinforcement learning (MFRL). Model predictive control (MPC), which can be formulated as an MBRL algorithm, can be used to solve problems with a clear goal state. In this work, we propose QRNN-ASNN-CEM/PF method to model, solve, and plan in control and robotics problems with continuous or discrete action spaces. The method uses a model a quantile regression neural network (QRNN) as a model of the environment that predicts a distribution over next states. It also uses an action sequence neural network (ASNN) to generate actions instead of sampling a uniform distribution, and either particle filtering (PF) or the cross-entropy method (CEM) to optimize the action sequences in MPC. We compare it to multiple ablations of the proposed method, as well as algorithms from the MBRL, MFRL, and trajectory optimization literature.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

GP	Gaussian process
μ	GP output mean
σ	GP output standard deviation
BO	Bayesian optimization
GPBO	Gaussian process-based bayesian optimization
RL	Reinforcement Learning
μ_{RL}	Discrete RL policy
π_{RL}	Stochastic RL policy
s	State
s'	Next state
s_t	State at time step t
s_{pred}	Predicted next state
s_{env}	Environment next state
a	Action
a_{dim}	Dimention of action space
a_{low}	Action lower bounds
a_{high}	Action upper bounds
N_a	Number of actions (discrete action space)
AS	Action sequences/particles
N_{AS}	Number of action sequences/particles
R_t	Reward at time step t
G_t	Episodic return from time step t
γ	Discount factor
DRL	Deep reinforcement learning
NN	Neural network
DNN	Deep neural network
MFRL	Model-free Reinforcement Learning
MBRL	Model-based Reinforcement Learning
DQN	Deep-Q Network
DDQN	Double Deep-Q Network
SARSA	State-action-reward-state-action algorithm
IV-DQN	Inverse-variance DQN
QR-DQN	Quantile regression DQN

NN	Neural Network
DNN	Deep Neural Network
DPG	Deep deterministic policy
DDPG	Deep deterministic policy gradient
TD3	Twin delayed DDPG
SAC	Soft actor-critic
TQC	Truncated quantile critic
A2C	Advantage actor-critic
PPO	Proximal policy optimization
QRNN	Quantile Regression Neural Network
50NN	Neural Network predicting the 50% quantile
MSENN	Neural Network that predicts a single value and uses the mean square error
MPC	Model Predictive Control
env	Environment of an RL problem
seed	Number used to reset the RL environment to a common start state
PF	Particle filtering
CEM	Cross-entropy method
iCEM	Improved cross-entropy method
iLQR	Iteration linear quadratic regression
ASNN	A neural network to generate an action sequence
MPPI	Model path integral method
RS	Random shooting
PETS	Probalistic ensemble trajectory sampling
GP-MPC	Gaussian process model predictive control
RBF	Radial basis function
BFGS	Broyden–Fletcher–Goldfarb–Shanno algorithm
H	MPC time horizon
$\mathcal{U}$	Uniform distribution
$\mathcal{N}$	Normal/Gaussian distribution
P	Action probability distribution used when generating action sequences
N_q	Number of quantiles
τ_i	Qauntile i used in quantile regression loss
L	Loss function used to update neural networks
l_n	Loss at step n
N_{batch}	Bath size

p	Probability of change (used in the discrete action sparce version pf PF)
Bern	Bernouilli distribution (used in the discrete action sparce version pf PF)
σ^2	Gaussian noise variance (used in the continuous action space version of PF)
ε	Gaussian noise (used in the continuous action space version of PF)
N_{top}	The number of top action sequences
AS_{top}	Top action sequences (the ones with the smallest cost)
α	Laplace smoothing coefficient
c_t	Cost at time step t
C	Cost function
Σ	Covariance function
std	Standard deviation
D	Discrete action space
C	Continuous action space

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

This master's thesis was completed in Marco Bonizzato's sciNeurotech laboratory, where neurostimulation is a crucial component of the experimental side of the research laboratory, and Gaussian process-based Bayesian optimization (GPBO) is a specialty of the computational side [3]. The goal of neurostimulation is to aid in rehabilitation by stimulating the patient's brain or spinal cord via electrical signals [3]. Neurostimulation can be thought of as an optimization problem where an algorithm needs to find the simulation parameters to maximize an output, like the step height. GPBO is a black-box optimization method [4]. It uses a GP as a surrogate or an approximation of the objective function that is assumed to be unknown. A GP outputs a mean μ and an uncertainty σ. These parameters enable the creation of an acquisition map, from which the following query to evaluate is obtained using BO. GPBO has been used in various contexts, including material science [5], drug discovery [6], hyperparameter tuning [7], and neurostimulation [3]. In neurostimulation, standard experiments such as reaching and grasping [8] or treadmill walking [3] can be considered single-step problems where the goal is to maximize an output by selecting an input, like a multi-armed bandit problem [9]. More precisely, a set of stimulation parameters is chosen and input into the system, the corresponding output is measured, and the system is reset. The GPBO process involves BO selecting the following input parameter, and the system outputs a value representing the input's effect. This value, along with the system input, is used to train the GP model [4]. In neurostimulation experiments, the parameter space can be large, and we want to minimize the number of queries made [3]. This makes GPBO very useful since it is sample-efficient. However, when wishing to perform multiple stimulations one after another sequentially, like a Markov decision process (MDP) problem [9], GPBO can no longer be used, as Bayesian optimization (BO) generally plans one step at a time and therefore cannot consider delayed rewards. Moreover, GP struggles with computational complexity, making it a less suitable option for longer MDP problems [4]. On the other hand, reinforcement learning could be used to solve this type of task.

Reinforcement learning (RL) algorithms are made for solving sequential decision-making problems and handling delayed rewards [9]. However, they are often extremely sample inefficient, needing thousands or even millions of training iterations to learn a task, especially for model-free RL (MFRL) algorithms [10]. This can be explained by the fact that an RL method generally learns from a scalar reward, which can be ill-defined or sparse [11]. This slow learning is not inherently problematic in benchmark examples, as the environment has

no time constraints, costs, or risks. This is quite the opposite in real-world applications, such as robot control or neurostimulation, where the task must be solved as quickly and safely as possible. To enhance sample efficiency, model-based reinforcement learning (MBRL) can be employed [12]. In addition to learning a policy, which refers to the method by which actions are chosen, an MBRL algorithm must also learn a model of the environment. When the studied task has a clearly defined goal state, we can formulate the classic control method, model predictive control (MPC), as an MBRL method [13].

1.1 The problem we are trying to solve

By definition an MPC method assumes a model of the task is known and will be used to simulate action sequences to find the next action to take [13]. As mentionned previously, there is not a known model for neurostimulation problems. A model of the problem must be learnt, where it would be used to predict the following state based on the current state and the chosen simulation parameters. If we want a computational cheap version of a GP as a model of the task, we could use a neural network that predicts a distribution over the next states. We would then need to use a method to select the next state from the distribution. The model would be trained using the states, which should lead to being much more sample efficient than an model-free reinforcement learning (MFRL) method. The idea of MPC as an MBRL algorithm is to test multiple candidate action sequences or particles at each time step by simulating them using a model of the environment, computing their cost, and using the first action of the particle with the smallest cost in the given task [12]. Using a cost function that primarily depends on the agent's state components allows avoiding the need to model the rewards, which can be poorly defined or sparse for the given problem, and therefore complicate learning [11].

In this thesis, Model Predictive Control (MPC) as an MBRL algorithm is applied to solve control and robotics problems with a clearly defined goal state and no known model of the problem's dynamics, unlike in classic MPC. We decide to work with these problems, since they share similarities with a sequential neurostimulation problem for which no such problem has been developped and is ready for testing. Our work covers the following component of a robotics problem:

- Modeling [14], since a model of the environment is learnt
- Control [14, 15], since we use an MPC method adapted to MBRL to solve the tasks
- Planning [15], since we use MPC to simulate the action sequences using the model of

the environment.

Table 1.1 Comparison of the strengths and weaknesses of GPBO, RL, and an MPC method using a model of the environment in the form of a neural network that predicts a distribution of next states as presented above.

Characteristic	**GPBO**	**RL**	**MPC with a neural network that predicts a distribution over states**
Sample efficient	✓	✗	✓
Optimizes over multiple timesteps	✗	✓	✓
Predicts a distribution	✓	✗	✓
Not computationally complex	✗	✓	✓
Handles delayed rewards	✗	✓	✓
Learns from	Query x and $f(x)$	Scalar reward	States

1.2 Research objectives

The goals of our project were to:

1. Developp a MBRL type MPC method with a computationally cheap model of the problem that **predicts a distribution of next state** and that can be used **without any pre-training**.

2. Validate the developped method on control and robotics tasks with a:

   - Clearly defined goal state.
   - Continuous or discrete action spaces.

3. Compare the developped method with:

   - Multiple ablations that are obtained using different models of the problem dynamics, methods to generate action sequences, techniques used to optimize them in MPC.

- Algorithms from the model-free RL, model-based RL, and trajectory optimization literature.

1.3 Contributions

The contributions of this Master's thesis are:

1. A Model predictive control based method using a:
   - Quantile regression neural network (QRNN) as a model of the environment,
   - Neural network to generate an action sequence neural network (ASNN),
   - And Particle filtering (PF) or the cross-entropy method (CEM) to optimize action sequences in the MPC loop for continuous and discrete action space problems.
2. Multiple simplifications of the proposed method to show the impact of each component of a MPC method in MBRL, which are:
   - Different models of the problem dynamics
   - Techniques to generate action sequences
   - Methods to optimize the action sequences in MPC
   - Ways to modify action sequences before the following MPC optimization loop to find the next action
3. A thorough comparison of the proposed method with its simplifications and multiple standard MFRL, MBRL, and trajectory optimization methods from the literature.

1.4 Thesis outline

The following pages are subdivided into six chapters: Introduction, Literature review and background, Methodology, Results, Discussion, and Conclusion.

- The first chapter is the introduction we just covered.
- The second chapter will present a literature review and some background. The chapter is separated into four parts. The first part will present the necessary theory on reinforcement learning, Markov decision processes, and returns. The second part will cover the different model-free reinforcement learning algorithms present in the literature. The third section will give a background on model predictive control. The final section will present MBRL and trajectory optimization algorithms from the literature.

- The third chapter is the methodology. It will explain the studied benchmark problems and the methods used to compare the algorithms.
- In the fourth chapter, the results will be presented and analyzed.
- The fifth chapter goes over the general trends in the results.
- The final chapter will present a conclusion recapping the work done, the limits of the developed algorithm, and future work.

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

The background will cover the necessary theory and concepts of Reinforcement learning (RL) and Model predictive control (MPC). The literature review will cover the pertinent RL and MPC-based methods present in the literature relevant to the project.

2.1 Background of Reinforcement learning

2.1.1 Markov decision process and returns

Reinforcement learning is a subfield of artificial intelligence where an agent must learn to select the best action to take in each state, thereby maximizing its long-term return [9]. Figure 2.1 shows the different components of an RL problem: Agent, state, action, environment, and reward.

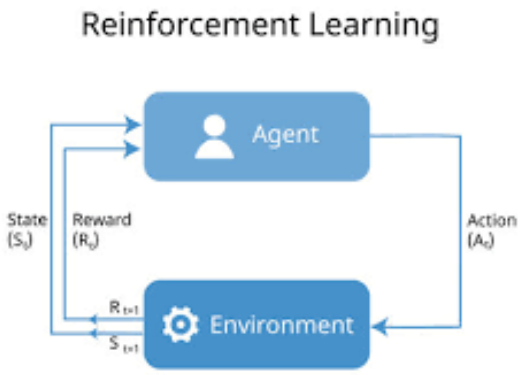


Figure 2.1 Components of a reinforcement learning problem [1]

The agent is what the RL algorithm controls [9]. The environment is the problem setting and what the agent interacts with. The different positions in the environment are referred to as the states defined as s or s_t. In each state s, the agent chooses an action a following a certain strategy known as stochastic policy π_{RL}. The agent will then transition following the environment's dynamics to the following state s' or s_{t+1}. The policy π_{RL} is generally a probability distribution of possible actions to take in a state s as defined in Equation 2.1 [16]:

$$a \sim \pi_{RL}(s_t) \tag{2.1}$$

The policy can also be deterministic, meaning that it will always suggest the same a for a given s. It is defined as μ and is shown in Equation 2.2 [16]:

$$a = \mu_{RL}(s_t) \tag{2.2}$$

The reward r is a number that quantifies how desirable or not the result of the sequence of actions is so far.

The RL problems considered in this thesis will have a finite time horizon, meaning there is a clear ending to an episode [9]. You can think of an episode as a run or a test of a certain length at solving a given task. In an episodic problem, it is possible to define the episodic return G_t obtained as the sum of all the obtained rewards in an episode as described in Equation 2.3 [9]:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{2.3}$$

where

- T is the last time step of the episode
- R_{t+i} is the reward at a the time step $t+i$, $i \in [t+1, T]$

When the problem does not have a clear ending, it is considered as a continuing task. For this type of task, the discounted return is used as presented in Equation 2.4 [9]:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} \tag{2.4}$$

where

- R_{t+k+1} is the reward at the $t+k+1$ time step
- γ is the discount factor.

The discounted return can also be used for episodic tasks. In this case, the summation upper bound would have to be replaced by the termination time step T.

For γ=0, the agent only considers immediate rewards since $G_t = R_{t+1}$. The agent is therefore myopic. For $\gamma = 1$, the agent equally considers immediate and future rewards.

$$G_t = \sum_{k=0}^{\infty} R_{t+k+1} = R_{t+1} + \sum_{k=1}^{\infty} R_{t+k+1} \tag{2.5}$$

For $\gamma \in]0, 1[$, the agent will give more importance to more recent rewards and progressively less importance to reward terms in the future.

The concept of a delayed reward is essential because an action's effect or result may not happen immediately in each problem [9]. An example is the mountain car example [2]. The agent must reach the top of the mountain as presented in figure 2.2. The mountain is such that the agent must gain momentum by moving left to reach the top. This is an example of delayed rewards since on the short-term moving to the left is bad but in the long term it is a good thing since it allows the agent to attain the goal.

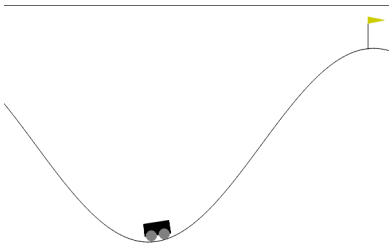


Figure 2.2 Mountain car [2]

2.1.2 Characterizing RL algorithms

Other important definitions in RL are off- versus on-policy algorithms and model-based (MBRL) versus model-free RL (MFRL).

The agent uses the same policy for action selection and learning in on-policy algorithms [9]. In off-policy algorithms, the agent uses a target policy for learning and a behaviour policy for action selection. Examples of off-policy algorithms are Q-learning, deep Q-network (DQN) [17], deep deterministic policy gradient (DDPG), twin delayed DDPG (TD3), soft actor-critic (SAC), and truncated quantile critic (TQC). Examples of on-policy algorithms include the state-action-reward-state-action (SARSA) algorithm, the proximal policy optimization method (PPO) and the advantage actor-critic (A2C) algorithm.

In a model-free RL (MFRL) algorithm, the agent learns by interacting with the environment and trying different actions in different states. Example MFRL algorithms are DQN [17], SAC [18], DDPG [19], TD3 [20], and PPO [21]. A model-based RL (MBRL) algorithm is either given or learns a model of the environment and uses it for planning in future time steps. More precisely, for a given state and a chosen action, the agent attempts to predict the next state, and the reward obtained. Example MBRL algorithms are PILCO [22], and model predictive control type (MPC) methods [13] like PETS-CEM [23], POPLIN [24], and

random shooting [25]. An MBRL algorithm is often much more sample-efficient since it can test actions before taking them, but this is only true if a quality model of the environment is learnt.

The following subsections provide a more in-depth discussion of different RL and control algorithms.

2.2 Literature review of RL algorithms

Below is a review of Deep reinforcement learning, different model-free RL methods for continuous state spaces and discrete or continuous action spaces.

2.2.1 Deep RL

Deep reinforcement learning (DRL) is the subfield of RL where deep neural networks (DNN), as applied in Deep learning, are integrated into RL algorithms [16, 26]. In Deep learning, one generally has a pre-collected dataset that will be used for analysis [27]. This isn't the case in RL [9]. Instead, the dataset used to train the neural networks is collected during the episodes. Using a DNN enables RL algorithms to be applied to problems with more than just discrete action spaces, such as those with continuous action spaces and other interesting inputs, like images [16, 26].

2.2.2 Discrete actions - DQN methods

Q-learning, SARSA, and DQN

Two classic RL methods for problems with discrete states and actions are Q-learning and the state-action-reward-state-action (SARSA) algorithm [9]. When the states are continuous, it is no longer possible to use tabular methods like Q-learning and SARSA. Function approximations, such as neural networks (NN), are used instead [9]. The deep Q-network (DQN) method extends Q-learning by using a neural network instead of a table to store the different Q-values for each state-action (s_t, a_t) pair [17]. The limitations of DQN are that it can diverge during learning, struggles with the maximization or overestimation bias, is sample-inefficient, and does not consider uncertainty [28–30]. More advanced DQN methods, such as double DQN (DDQN), quantile regression DQN (QR-DQN) [29] and inverse-variance DQN (IV-DQN) [30], address some of these problems.

QR-DQN

The quantile regression DQN (QR-DQN) uses a quantile regression neural network to get a distribution of Q-values instead of a single value [29]. This allows for better consideration of uncertainty and noise in environments by considering multiple predictions, which leads to a more sample-efficient method.

IV-DQN

The inverse-variance DQN (IV-DQN) predicts the Q values like a standard DQN would, but it also predicts the heteroskedastic noise in the target model predictions [30]. It uses importance sampling to update the Q-values. The weight associated with the different updated Q-values depends on the prediction variance, where a higher weight is given to predictions with a smaller variance. This leads to IV-DQN being more sample-efficient and less affected by the overestimation bias.

2.2.3 Continuous actions and off-policy

Some of the standard deep RL methods that are off-policy and compatible with continuous action spaces are the deep deterministic policy gradient (DDPG) [19], twin delayed DDPG (TD3) [20], soft actor-critic (SAC) [18], and truncated quantile critic (TQC) [31].

DDPG

The deep deterministic policy gradient (DDPG) improves on the DQN and the Deterministic Policy Gradient (DPG) [19] algorithms. Like DQN, DDPG uses a replay buffer and a target network to approximate the Q-values. Unlike DQN though, it is compatible with continuous action spaces. The learnt policy is deterministic, like DPG, but the method adds an actor-critic framework. The method's limitations include its susceptibility to the overestimation bias and its hyperparameter sensitivity.

TD3

The twin delayed DDPG (TD3) method is an off-policy algorithm that outperforms and is more stable than its predecessor, DDPG [20]. Key elements of the technique include adding noise to the actions, known as target policy smoothing, delayed policy updates to smooth out the agent's learning, where the actor is updated less frequently than the critic, and taking

the minimum of the predictions from twin Q-networks, similar to what was presented in double DQN [20, 28].

SAC

The soft actor-critic (SAC) method maximizes simultaneously the reward and the policy entropy [18]. The latter is a classic representation of disorder and a hyperparameter that affects the randomness in the actions taken and exploration, similar to ε in the ε-greedy algorithm. Compared to other MFRL algorithms, the method is known to be very stable and less sensitive to hyperparameters.

TQC

The truncated quantile critic (TQC) method solves continuous action space RL problems [31]. It combines quantile regression from QR-DQN [29], the actor-critic framework from SAC [18], and the truncation of the model outputs similar to TD3 [20]. The idea is to

- Use multiple quantile regression critic function approximation models,
- Create a mixture of the models by aggregating the outputs of the quantile critics,
- Truncate the predicted Q-values by removing the k-ones with the most significant values to lessen the overestimation bias.

The improvements brought by TQC are primarily demonstrated in complex, continuous control tasks [32], such as the MuJoCo Walker environment [2].

2.2.4 On-policy and any actions (discrete or continuous)

Two on-policy RL algorithms compatible with continuous or discrete action spaces are advantage actor-critic (A2C) and proximal policy optimization (PPO). For the PPO method, two sub-methods exist that use clipping or a penalty to prevent the policy from changing too much.

2.3 Background of Model Predictive Control in MBRL

Model predictive control (MPC) aims to optimize action sequences over a time horizon H to minimize a cost function C [13]. The cost function generally depends on the distance between the current state s_t and the goal state s_g. C can also depend on other problem-dependent

variables, such as velocity v, the action taken a, or the reward r. As MBRL algorithm, MPC selects the following action based on the current state s_t. This action is then taken in the environment, which can be described as $(s_t, a_t) \rightarrow s_{t+1}$. The process of selecting the next action is then repeated from s_{t+1}.

As Figure 2.3 shows, the three-part structure of applying an MPC-based method problem. The first is to generate action sequences, reinitializing the environment and initializing a model of the environment. The second is to simulate the action sequences using the model of the environment, calculate the cost of each action sequence, and then optimize them using a chosen method. The final part consists of taking a step in the environment with the best action sequence, which is the one with the smallest cost, and then modifying the action sequences or generating new ones before repeating the MPC optimization loop.

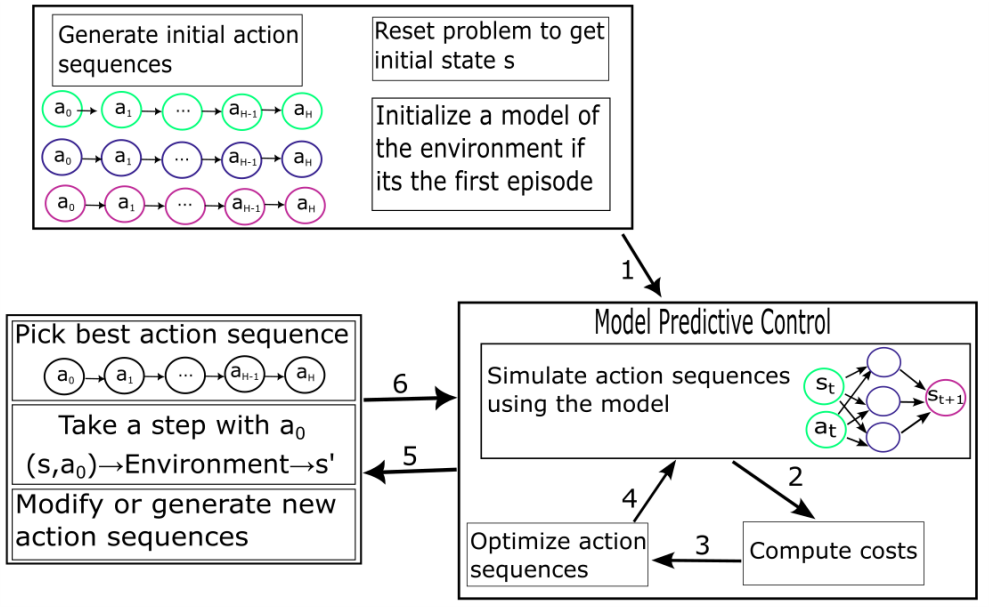


Figure 2.3 General idea of an MPC-based method applied to an RL problem

2.4 Literature review of some control methods

Below is a review of different MPC shooting and trajectory optimization control algorithms for continuous state spaces and discrete or continuous action spaces.

2.4.1 MPC shooting algorithms

Random shooting

The idea of random shooting (RS) is to [12]:

- Randomly generate action sequences at each step of an episode,
- Roll them out or simulate them using a model of the environment in an MPC fashion, and
- Take a step in the environment with the first action of the action sequence with the smallest cost

PETS

Probabilistic ensemble trajectory sampling with the cross-entropy method (PETS-CEM) uses multiple probabilistic neural networks to model the environment, a trajectory sampling method to determine which models are used, as well as the cross-entropy method (CEM) to generate action sequences [23]. The authors present two sampling methods: TS_1 and TS_∞. The first resamples a new model for each particle, defined as a state sequence, at each time step. The second samples a model at the first time step and sticks to it for the rest of the time for each particle. The multiple probabilistic neural networks allow to learn a good model of the environment, and the trajectory sampling considers the uncertainty in the state predictions.

POPLIN

The POPLIN method is known to improve PETS [24]. It can optimize in the action (POPLIN-A) or the parameter (POPLIN-P) space. POPLIN-A uses a policy network to generate initial action sequences, which is a major difference from PETS. The action sequences are modified with the addition of Gaussian noise. The noise parameters (μ, σ) are optimized using the cross-entropy method (CEM) for multiple optimization steps, where at each optimization step only the top ζ action sequences are used. The authors used a deterministic ensemble of neural networks as a model of the environment, but a probabilistic ensemble could be used instead. The POPLIN-P method instaed adds noise to the policy network parameters. Once again, CEM is used to optimize the noise.

GP-MPC

GP-MPC is an MPC method that utilizes a Gaussian process (GP) with a radial basis function (RBF) kernel to model the environment. S. Kamthe and M.P. Deisenroth proposed the idea [33]. Although their official implementation is not available, an unofficial implementation is [34]. The latter, which is based on Deisenroth's PhD thesis [35] and their article [33], is explained below.

The GP predicts the difference in the next state components in the form of an expected mean μ_t and a covariance matrix Σ_{s_t} [34]. The author of the implementation considers the state components to be independent, which is consistent with the work of Kamthe and Deisenroth [33]. In MPC, the GP model with an RBF kernel is used to roll out action sequences generated using the L-BFGS-B method [34], which is a version of BFGS with limited memory and that is compatible with constraints of the form $l_b \leq s \leq u_b$, where l_b is a lower bound and u_b is an upper bound [36]. The author decided to use a quadratic cost function as was presented in equation 3.58 of Deisenroth's PhD thesis [35] and given in Equation 2.6:

$$c(s_t) = (s - s_{goal})^T W^{-1} (s - s_{goal}) \tag{2.6}$$

where W^{-1} is a symmetric weight matrix.

Equations 2.7 and 2.8 represent the expected and the variance of the predicted cost for a given time step t, as described in equations 3.59 and 3.60 of Deisenroth's PhD thesis [35]:

$$\mathbb{E}_{s_t}[c(s_t)]C_{\mu,t} = e_t^T W^{-1} e_t + \text{tr}(W^{-1}\Sigma_{s_t,a_t}) \tag{2.7}$$

$$\text{var}_{s_t}[c(s_t)] = 2\text{tr}(W^{-1}\Sigma_{s_t,a_t}W^{-1}\Sigma_{s_t,a_t}) + 4e_t^T W^{-1}\Sigma_{s_t,a_t}W^{-1}e_t \tag{2.8}$$

where $e_t = \begin{bmatrix}\mu_{s_t}\\ \mu_{a_t}\end{bmatrix} - \begin{bmatrix}s_{target}\\ a_{target}\end{bmatrix}$ includes the error in the next state prediction s_t and the difference in the current action a_t and the goal action [34]. $\Sigma_{s_t,a_t} = \begin{bmatrix}\Sigma_{s_t} & 0\\ 0 & 0\end{bmatrix}$ is the combined state-action covariance matrix. In the $C_{\mu,t}$ calculation, constraints could be considered, but they are not in the given implementation.

For the last state of the prediction time horizon s_H, the idea is the same, but the weight matrix W_H may differ, and the cost functions are slightly different. The cost functions are given in equations 2.9 and 2.10, as described in equations 3.59 and 3.60 of Deisenroth's PhD

thesis [35]:

$$\mathbb{E}_{s_H}[c(s_H)] = e_H^T W_H^{-1} e_H + \text{tr}(W_H^{-1}\Sigma_{s_H}) + \text{Constraints} \tag{2.9}$$

$$\text{var}_{s_H}[c(s_H)] = 2\text{tr}(W_H^{-1}\Sigma_{s_H}W_H^{-1}\Sigma_{s_H}) + 4e_H^T W_H^{-1}\Sigma_{s_H}W_H^{-1}e_H \tag{2.10}$$

where $e_H = \mu_{s_H} - s_{target}$ includes the error in the next state prediction s_t [34].

Using the mean and variance of the cost at time step t, we can use the lower confidence bound (LCB) to determine a value of the cost while being optimistic [34, 37]. Equation 2.11 shows this:

$$c_{t,LCB} = \mathbb{E}_{s_t}[c(s_t)] - \kappa \cdot \text{var}_{s_t}[c(s_t)] \tag{2.11}$$

Based on the cost of each time step, the average cost of the trajectory of an action sequence can be computed using Equation 2.12 shows [34, 37]:

$$C_{LCB} = \frac{1}{H}\sum_{t=0}^{T} c_{t,LCB} \tag{2.12}$$

Using this cost function, multiple iterations of LBFGS are done, and the action used for the following N_{repeat} steps will be the first of the best action sequences, which is the one with the smallest cost [34]. After taking these steps in the environment, the action sequence is shifted to remove the action that was just taken, and the last action is unchanged $[a_0, a_1, ..., a_H] \rightarrow [a_1, a_2, ..., a_{H-1}, a_H, a_H]$. This serves as the starting point for the subsequent L-BFGS-B optimization.

Using a quadratic cost function in GP-MPC has multiple limitations. First, $\sigma_{s_t}[c(s_t)] = \sqrt{\text{var}_{s_t}[c(s_t)]}$ increases as μ_{s_t} moves farther from the target state $s_{t,target}$, which with exploration favors states further from the goal state [35]. Second, a quadratic cost function stronly depends on the worst state of a given sequence of states [35]. This means a sequence of states that went far from the goal but eventually reaches it will be considered as worse than a sequence of states that never attained the goal state.

2.4.2 Trajectory optimization control algorithms

CEM and iCEM

The cross-entropy method (CEM) is a sampling-based optimization method that can be used to generate action sequences to be optimized in MPC [38–40]. For each time step t over the time horizon H, the idea of CEM is:

1. To maintain a Gaussian distribution with a mean μ_t and a variance σ_t
2. Do multiple CEM optimization iterations, where each one consists of:
    - Sampling the distribution to generate action sequences
    - Simulate the action sequences using a model of the environment in MPC to obtain trajectories
    - Calculate the costs associated with the action sequence's trajectories
3. Update the distribution parameters μ and σ based on the top action sequences
4. Take a step in the environment with the first action of the best action sequence

The iCEM method uses an improved MPC-CEM method [41]. Some of the important improvements by the authors are [41, 42]:

- Saving the K top particles and reusing them for the next CEM optimization step to help accelerate things.
- For every first CEM optimization step, some of the top K particles obtained from the previous time step CEM optimization are used by shifting them to remove the action taken, and the last vacant action is replaced by a random one.
- Using colored noise when creating action sequences. This leads to the particles being more correlated and a better exploration method. The colored noise depends on a parameter β, the colored-noise scaling exponent, which is a value that must be tuned for a given problem.

MPPI

Model Predictive Path Integral (MPPI) is a sampling-based Model Predictive Control (MPC) method [43,44]. The method starts with an initial action sequence a_t sampled from a uniform distribution and generates K action sequences from it by adding noise sampled from a Gaussian distribution as equation 2.13 describes:

$$a_t^{(k)} = a_t + \varepsilon_t^{(k)} \text{ where } \varepsilon_t^{(k)} \sim \mathcal{N}(\mathbf{0}, \Sigma) \tag{2.13}$$

where $^{(k)}$ represents the k$^{\text{th}}$ action sequence.

The action sequences are then simulated using a model of the environment, such as a standard feed-forward neural network, and the total cost of a given action sequence $a^{(k)}$ is computed using Equation 2.14:

$$C_t^{(k)} = c_H(x_H) + \sum_{t=0}^{H-1} c_t(x_k, a_t^{(k)}) \tag{2.14}$$

where $c_t(x_k, a_k)$ is the cost used throughout the episode and $c_H(x_H)$ is the cost at the end of the planning time horizon.

Based on the costs, each action sequence is associated with a weight ω_k as shown by Equation 2.15 [43, 44]:

$$\omega^{(k)} = \frac{\exp\left(-\frac{1}{\lambda} \cdot \left(C^{(k)}\right)\right)}{\sum_{i=0}^{K-1} \exp\left(-\frac{1}{\lambda} \cdot \left(C^{(i)}\right)\right)} \tag{2.15}$$

where λ is a temperature parameter that affects the importance of lower cost trajectories [45]. With a lower λ, a more exploitative weighting scheme will be used, and more importance will be given to lower-cost trajectories. A higher λ corresponds to a more exploratory weighting scheme, which means larger cost trajectories are considered more.

From there, the best action sequence is obtained by using the Boltzmann weighing of the particles [46] presented in Equation 2.16 [43, 44]:

$$a_t^* = a_t + \sum_{k=0}^{K-1} \omega^{(k)} \varepsilon_t^{(k)} \tag{2.16}$$

The first action of the newly obtained best action sequence a_t^* is taken in the environment, and the method starts again a_t^*.

As the equations above show, MPPI is a gradient-free method, which makes it more robust to non-smooth environment dynamics.

iLQR

Iteration Linear Quadratic Regulator (iLQR) is a trajectory optimization method, and it can be integrated into a classic MPC algorithm where a model $f(s, a)$ of the task is known [47]. The idea is to consider the environment dynamics as locally linear and the cost function as locally quadratic. Based on an initial randomly generated action sequence, the method begins with the forward pass, which involves rolling out the action sequence using the known problem dynamics. From the obtained sequence of states $\{s_0, s_1, ..., s_H\}$, where H is the

prediction time horizon, the environment state dynamics are approximated by doing a linearization as described by equation 2.17:

$$\delta_{s_{t+1}} = \frac{\partial}{\partial s} f(s_t, a_t)\delta_{s_t} + \frac{\partial}{\partial a} f(s_t, a_t)\delta_{a_t} \tag{2.17}$$

The cost function is then calculated based on the approximate trajectory. Then comes the backward pass, which consists in calculating a key matrix $\boldsymbol{K}_t$ and a vector k_t, which can be obtained looking over the LQR literature. $\boldsymbol{K}_t$ and k_t are then both inserted into Equation 2.18:

$$\delta_{a_{t+1}} = \alpha \cdot k_t + K_t \cdot \delta_{s_{t+1}} \tag{2.18}$$

From here, a new action sequence is obtained, which can be rolled out from the start state s_0 using the actual environment dynamics. The process continues with the backward pass and forward pass until convergence.

Neural iLQR

The neural iLQR method utilizes a neural network to learn the environment's dynamics, rather than relying on the ground truth dynamics [48]. This is particularly useful in the iLQR backward pass, where the derivatives required for gradient and Hessian calculations must now be estimated using methods such as automatic differentiation. Neural iLQR can be used as an alternative to iLQR when the ground-truth dynamics are unknown. The author's code isn't available, and nobody else has tried to implement it.

2.5 Recap of the literature review

Table recaps the algorithms presented in the literature review. It shows the different methods compatible with continuous action spaces, the ones compatible with discrete action spaces, and those that learn a model of the environment.

Table 2.1 Recap of literature review algorithms

Algorithm	**Compatible with continuous action spaces**	**Compatible with discrete action spaces**	**Learns a model of the environment**
DQN	✗	✓	✗
QR-DQN	✗	✓	✗
IV-DQN	✗	✓	✗
DDPG	✓	✗	✗
TD3	✓	✗	✗
SAC	✓	✗	✗
TQC	✓	✗	✗
A2C	✓	✓	✗
PPO	✓	✓	✗
RS	✓	✓	✓
PETS-CEM	✓	✗	✓
POPLIN	✓	✗	✓
GP-MPC	✓	✗	✓
CEM	✓	✗	✓
iCEM	✓	✗	✓
MPPI	✓	✗	✓
iLQR	✓	✗	✗
Neural iLQR	✓	✗	✓

CHAPTER 3 METHODS

This section presents the proposed set of QRNN-ASNN methods, along with their multiple simplifications. We also explain the benchmark environments used to compare the different algorithms and provide essential information needed to replicate the results, including the cost functions used, the number of steps per episode, and the number of episodes for each benchmark environment.

3.1 Our different MPC methods and their components

We test a wide variety of MPC methods that differ in their model of the environment, as well as different techniques used to generate action sequences, modify them after taking a step in the environment, and improve the action sequences in MPC. Algorithm 1 presents a general pseudocode for the episodic loop of the MPC algorithms given below:

Algorithm 1 General MPC algorithm episodic loop

Require: N_{episodes}: Number of episodes, N_{steps}: Number of max steps per episode, env: Problem to be studied, seed: Environment reset seed value

```
for i = 0 to N_episodes do
    s, info ← env.reset(seed)                          ▷ Reset the environment
    Initialize model and its replay buffer
    Generate initial action sequences
    for j = 0 to N_steps do
        Get best and optimized action sequences from MPC
        The action to take a is the first action of the best action sequence
        reward, s', terminated, truncated, info = env.step(a)
        done ← (terminated or truncated)
        if done then
            break
        end if
        Train the model using (s, s')
        s ← s'
        Modify action sequences before next MPC optimization
    end for
end for
```

Algorithm 1 presents a simplified pseudocode of the MPC methods. The idea is to generate particles with one of the methods mentioned in Section 3.1.1. The generated action sequences are then passed to MPC, as described in Algorithm 9, where the model, among the ones

presented in Section 3.1.2, is used to predict the next state. The MPC action sequences are then optimized using one of the techniques presented in Section 3.1.3. The outputs of the MPC optimization are the optimized action sequences and the best action sequence, which is the one with the smallest computed cost. After taking a step in the environment with the first action of the best action sequence, the particles are modified by using one of the methods presented in Section 3.1.4.

The proposed novelties in 1 explained below are:

- Use the ASNN to generate initial action sequences and to modify the action sequences before the next action search with the MPC optimization. The ASNN is explained in section 3.1.1.
- QRNN and 50NN as models of the environment. They are presented in sections 3.1.2.
- A comparison of generating new action sequences at each step in the environment or modidying the ones from the previous MPC optimization loop. These ideas are presented in section 3.1.3.
- A comparison of particle filtering (PF) and the cross-entropy method (CEM) to optimize action sequences in MPC for continuous and discrete action spaces. These methods are presented in section 3.1.4.

We will now go over the different components of an MPC method we tested.

3.1.1 Methods to generate action sequences

The two methods to generate initial action sequences below can be used to replace the Generate initial action sequences module in Algorithm 1.

Uniform

To generate the initial action sequences, sampling a uniform probability distribution is often used. In the discrete action space case, an action among the possible ones is chosen. In the continuous action space, a uniform distribution with bounds (a_{min}, a_{max}) is used, where a_{min} is the action lower bound and a_{max} is the action upper bound.

Algorithm 2 presents how the QRNN model predicts the next state s' based on the current state s and an action a:

Algorithm 2 Randomly uniformly generating action sequences

Require: N_{AS}: Number of particles, H: Time horizon, N_a: Number of actions (discrete action space), (a_{min}, a_{max}): Action bounds (continuous action space), a_{dim}: Action dimension

1: **if** Action space is discrete **then**
2: $AS \leftarrow \text{randint}(0, N_a)$ ▷ Dimension (N_{AS}, H, a_{dim})
3: **else if** Action space is continuous **then**
4: $AS \leftarrow \text{U}(a_{min}, a_{max})$ ▷ Dimension (N_{AS}, H, a_{dim})
5: **end if**

ASNN

Instead of generating values by sampling a uniform distribution, a neural network can be used based on the current state s and the goal state s_g to generate a distribution of possible actions that can be sampled to create new particles. This action sequence generator neural network will be referred to as an ASNN from now on.

For discrete action space problems, the ASNN predicts a categorical distribution of possible actions, assigning a probability to the likely subsequent actions. Figure 3.1 presents the architecture used for the discrete ASNN:

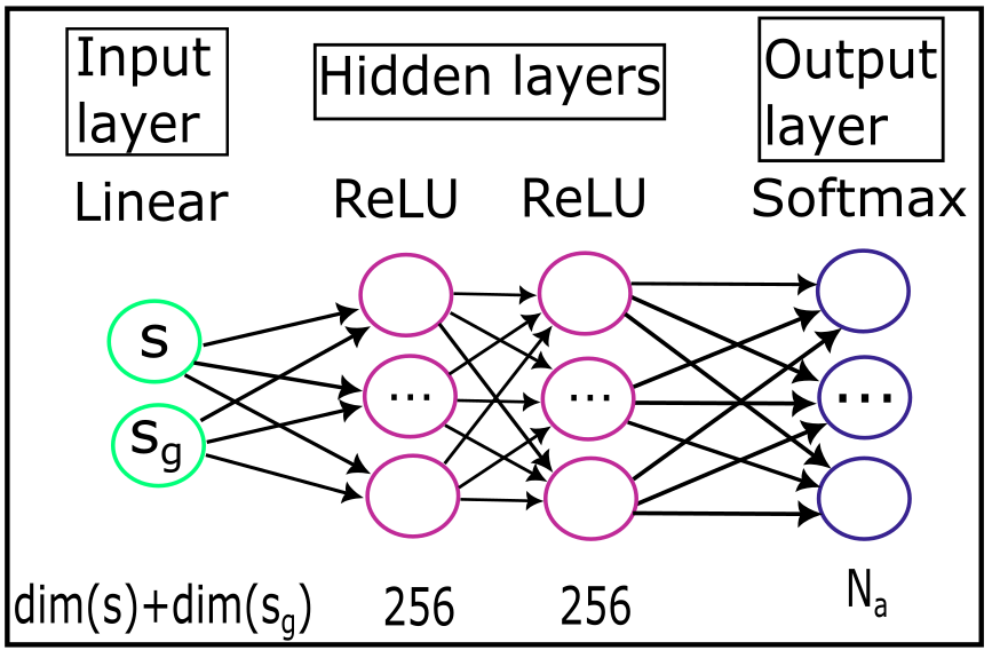


Figure 3.1 Discrete ASNN architecture

The negative log likelihood (NLL) loss is used to train the discrete ASNN neural network. Since the ASNN outputs a discrete probability distribution on the possible actions, we take

the negative logarithm of these probabilities, and the loss l_n for a given sample of the replay buffer is simply the value associated with the target action. Equation 3.1 shows this [49]:

$$l_n(\hat{a}, a) = -log(p_a) \tag{3.1}$$

where p_a is the ASNN predicted probability associated with the target action a obtained from the replay buffer. The ASNN is trained with the first action of the best particles found by MPC, thereby integrating the learning of the MPC into the ASNN.

When we consider N_{batch} elements sampled from the replay buffer, the mean loss is as shown in equation 3.2 [49]:

$$l(\hat{a}, a) = \frac{\sum_{n=1}^{N_{batch}} l_n(\hat{a}, a)}{N_{batch}} \tag{3.2}$$

For continuous action space problems, the ASNN predicts a mean μ_a and a standard deviation σ_a associated with the predicted next action to take. To be able to predict a μ_a and a σ_a we would need to represent the output of a Gaussian of the form [50]:

$$\pi(a|s, s_{goal}) = \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp\left\{ -\frac{1}{2}\frac{(a-\mu_a)}{\sigma_a^2}\right\} \tag{3.3}$$

The loss function $L(\pi)$ used would be [50]:

$$L(\pi) = -\log \pi(a|s, s_{goal}) = \frac{1}{2}\log(2\pi(a|s, s_{goal})\sigma_a^2) + \frac{1}{2}\frac{(a-\mu_a)}{\sigma_a^2} \tag{3.4}$$

At each step, there will be a new state s, and therefore a new μ_a and a new σ_a. Sampling a Gaussian distribution using these two parameters as inputs will allow the generation of new possible actions. At the start of an episode, the alternation between next action prediction based on the current state and next state prediction using a model of the environment will enable the generation of initial action sequences.

Figure 3.2 presents the architecture used for the continuous ASNN:

ASNN training

The ASNN model training is done via the following steps:

- The (s, s_g, a) triplets are stored at each step in the environment in a replay buffer

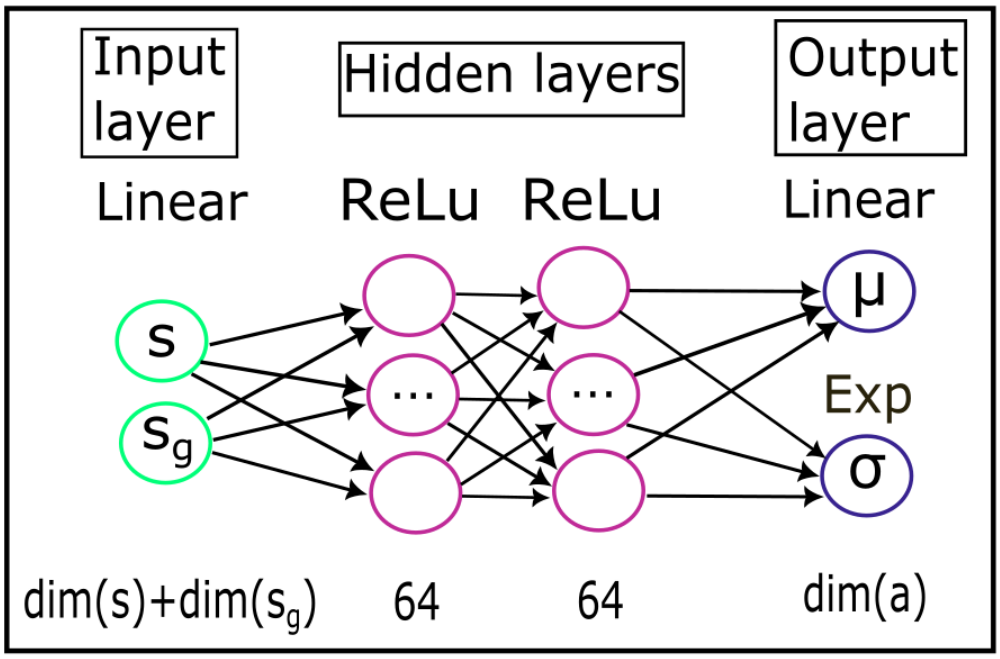


Figure 3.2 Continuous ASNN architecture

- At every $N_{batch} = 32$ steps, a batch of 32 elements is sampled from the replay buffer.
- Predict the probability distribution parameters using the ASNN(s, s_g).
- Compare the probability distribution parameters with the actions sampled the replay buffer using the categorical cross-entropy loss when the action space is discrete and the gaussian negative log likelihood loss when the action space is continuous.

Comparison of the computational burden of sampling a uniform distribution or the action sequence neural network

Sampling a uniform probability distribution does not require training a neural network to generate actions. For the ASNN, you must train a neural network, use it predict the parameters of a probability distribution, and then sample the probability distribution. It is clear that using the ASNN to generate an action is more costly than sampling a uniform distribution. However, using the ASNN to generate actions should improve to the agent's performance compared to a uniform distribution, since the ASNN is learning a distribution over the actions directly based on the states.

3.1.2 Models of the environment

The different models of the environment presented below can replace the model module in Algorithm 1.

QRNN

To model the environment dynamics, we use a quantile regression neural network (QRNN), similar to the one used in QR-DQN from W. Dabney's paper, entitled *Distributional Reinforcement Learning with Quantile Regression* [29]. A quantile is a number that separates data into subgroups of the same size [51]. For example, the value of a quantity representing the 90% quantile will have 90% of the rest of the data below it and 10% above it. The number of quantiles N_q is a hyperparameter. We chose to use 11 quantiles from 0% to 100% in 10% increments, where 0% and 100% are included. The critical difference in using a QRNN to obtain a distribution over next states, rather than next actions, enables our method to be applied to problems without any restriction on the type of state and action space, whether continuous or discrete. A distribution of N_q quantiles as predictions for the next state allows for the consideration of possible uncertainty in an environment by considering a distribution over next states. When making a prediction using the QRNN, we currently only use the mid or 50% quantile for simplicity. The purpose of predicting multiple quantiles is to ultimately utilize the entire distribution of the next states in MPC.

Figure 3.3 presents the architecture used for the QRNN:

Algorithm 3 presents how the QRNN model predicts the next state s' based on the current state s and an action a:

Algorithm 3 QRNN next state prediction

Require: QRNN: Quantile regression neural network, s: Current state, a: Action

1: $s'_Q = \text{QRNN}(s, a)$ ▷ Next state quantiles
2: $s' = s'_Q[:, 5, :]$ ▷ Mid quantile

The loss function used is the quantile regression loss, which contains a summation over the N_q quantiles as Equation 3.5 [52]:

$$L(s_{pred}, s_{env}) = \sum_{i=0}^{N_q} \max\Big(\tau_i(s_{pred} - s_{env}), (\tau_i - 1)(s_{pred} - s_{env})\Big) \tag{3.5}$$

where s_{env} is the true environment state, s_{pred} is the 50% quantile predicted by the QRNN, and τ_i is the quantile. The QRNN is trained by comparing its predictions with those of the

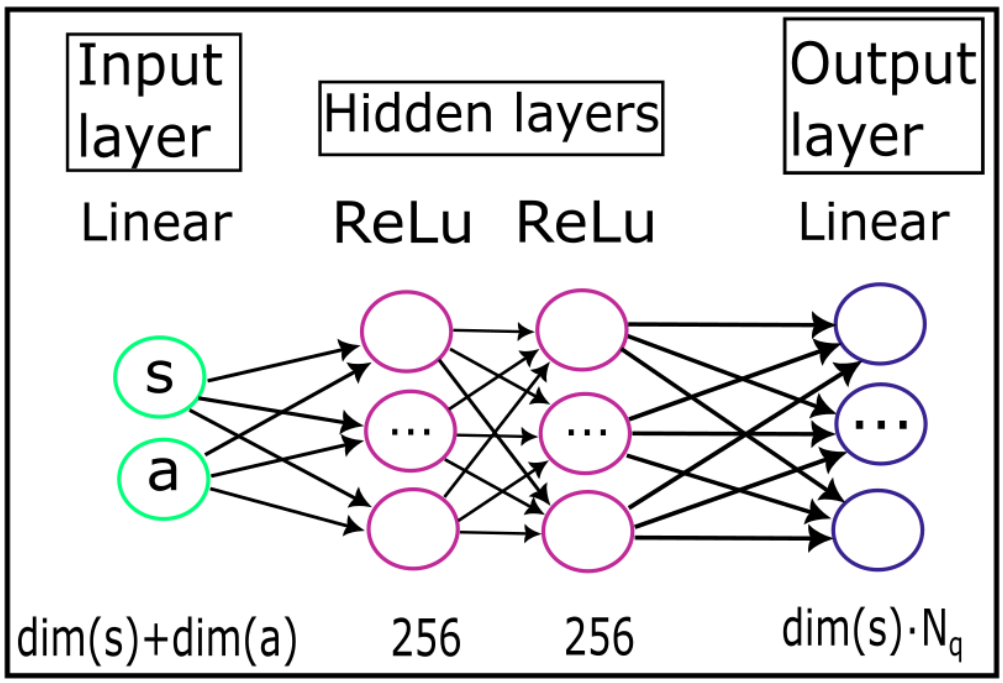


Figure 3.3 QRNN architecture

environment.

50NN

A simplification of the QRNN is only to predict the mid or 50% quantile. This NN is referred to as 50NN. The quantile loss is still used but only with $\tau = 0.5$. The 50NN is a test to determine if there is an advantage in predicting all quantiles compared to just predicting the 50% quantile.

Algorithm 4 presents how the 50NN model predicts the next state s' based on the current state s and an action a:

Algorithm 4 50NN next state prediction

Require: 50NN: Mid quantile neural network, s: Current state, a: Action

1: $s' = \text{50NN}(s, a)$

A simplification of the QRNN is to predict only the 50% quantile. The loss function used is still the quantile regression loss, with $\tau = 0.5$ as described in Equation 3.6, which is equivalent to the mean absolute loss multiplied by 0.5 [52]:

$$L(s_{pred}, s_{env}) = \max\Big(\tau(s_{pred,i} - s_{env,i}), (\tau - 1)(s_{pred} - s_{env})\Big) = 0.5|s_{pred} - s_{env}| \tag{3.6}$$

MSENN

MSENN is the standard feed-forward neural network used for next state prediction in multiple trajectory optimization methods [39, 41–44, 48, 53, 54]. It predicts a single value.

Algorithm 5 presents how the MSENN model predicts the next state s' based on the current state s and an action a:

Algorithm 5 MSENN next state prediction

Require: MSENN: Mean squared error neural network, s: Current state, a: Action

1: $s' = \text{MSENN}(s, a)$

The MSENN is trained using the mean-squared error [55], as described by equation 3.7:

$$L(s_{pred}, s_{env}) = (s_{pred} - s_{env})^2 \tag{3.7}$$

Model of the environment training

As a reminder, these models are used as part of a MPC method adapted for MBRL. Therefore, a model of the environment is not known and must be learnt online. The QRNN, 50NN, MSENN model training is done via the following steps:

- The (s, a, s') triplets are stored at each step in the environment in a replay buffer
- At every $N_{batch} = 32$ steps, a batch of 32 elements is sampled from the replay buffer.
- The model predicts the next states s_{pred} based on the sampled (s, a) pairs.
- The predicted next states s_{pred} are compared to the ones of the environment $s_{env} = s'$ using the loss function associated to the model as defined above.

3.1.3 How to modify the action sequences after taking a step in the environment

After taking a step in the environment with the first action of the best action sequence found using MPC, we compare three different methods to change the action sequences before the next MPC optimization. These can replace the Modify action sequences before next MPC optimization module of Algorithm 1.

Completely random

Completely random consists of forgetting the previous iteration's optimized action sequences and generating new ones from a uniform distribution.

Algorithm 6 presents how to generate new action sequences after taking a step in the environment:

Algorithm 6 Generate new action sequences using a uniform distribution

Require: N_{AS}: Number of particles, H: Time horizon, N_a: Number of actions (discrete action space), (a_{min}, a_{max}): Action bounds (continuous action space)

$AS \leftarrow MPC(s, AS)$ ▷ Discard the action sequences given by MPC
if Action space is discrete **then** ▷ Generate new action sequences
    $AS \leftarrow$ randint(0,N_a) ▷ Dimension (N_{AS}, H, a_{dim})
else if Action space is continuous **then**
    $AS \leftarrow \mathrm{U}(a_{min}, a_{max})$ ▷ Dimension (N_{AS}, H, a_{dim})
end if

Shift the action sequences and replace with one sampled uniformly

To consider the previous step's optimized action sequences, we shift them to remove the column of the action taken, and we replace the last column with the vacant action with ones sampled from a uniform distribution.

Algorithm 7 presents how to generate new action sequences after taking a step in the environment:

Algorithm 7 Shift old action sequences and replace vacant actions by sampling a uniform distribution

Require: H: Time horizon, N_a: Number of actions (discrete action space), (a_{min}, a_{max}): Action bounds (continuous action space), AS: Action sequences, shape (N_{AS}, A, a_{dim})

$AS[0 : H-1] \leftarrow AS[1 : H]$ ▷ Shift the action sequences
if Action space is discrete **then** ▷ Generate new action sequences
    $AS[H-1] \leftarrow$ randint(0,N_a)
else if Action space is continuous **then**
    $AS[H-1] \leftarrow \mathrm{U}(a_{min}, a_{max})$
end if

Shift the action sequences and replace with one sampled from the ASNN

This method is identical to the previous ones, but the new actions are sampled from the ASNN. The replacement of the vacant action can only be done at the $H-1$ time step of the

prediction time horizon in MPC, since the s_{H-1} time step is not known beforehand.

Algorithm 8 presents how to generate new action sequences after taking a step in the environment:

Algorithm 8 Shift old action sequences and replace vacant actions by sampling a uniform distribution

Require: N_{AS}: Number of particles, H: Time horizon, N_a: Number of actions (discrete action space), (a_{min}, a_{max}): Action bounds (continuous action space), a_{dim}: Action dimension, AS: Action sequences, shape (N_{AS}, H, a_{dim})

1: $AS[0 : H-1] \leftarrow AS[1 : H]$ ▷ Shift the action sequences
2: $P_{a,H-1} \leftarrow ASNN(s_{H-1}, s_{goal})$ ▷ For time step $t = H-1$ in MPC
3: $AS[H-1] \sim P_{a,H-1}$ ▷ Sample action probability distribution

Recap of the methods to modify action sequences

Figure 3.4 recaps the ways to modify action sequences after taking a step in the environment and before the following MPC optimizations iterations, which give the action for the next step in the environment:

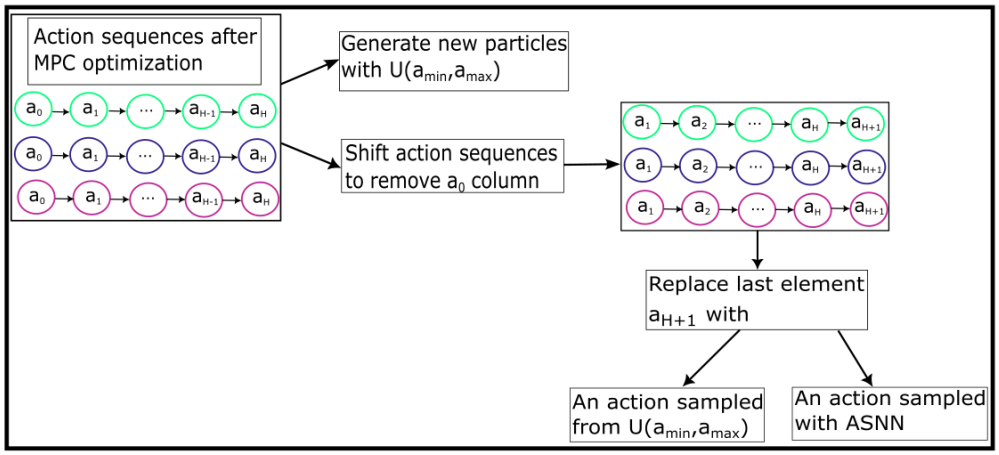


Figure 3.4 Ways to change particle after a step in the environment

3.1.4 MPC technique to optimize the action sequences

After iterating over all the particles and before performing another MPC iteration, the particles should be optimized to improve them. We consider two possible methods: particle filtering (PF) [24, 43, 44] and the cross-entropy method (CEM) [23, 38–40]. We implement versions of PF and CEM adapted for continuous and discrete action spaces. In both cases,

the new action sequences are generated based on the top N_{top} action sequences with the smallest costs. The pseudocode 9 below shows a simplified MPC code in which PF or CEM is used to replace the Update action sequences module. This function can replace the Get best and optimized action sequences from MPC module of Algorithm 1.

Algorithm 9 MPC optimization function

Require: AS: Action sequences, s_0: Current state, C: Cost function, number of action sequences N_{AS}, N_{rep}: Number of repetitions MPC, problem: RL environment to solve

for i = 0 to N_{rep} **do**
  costs = zeros(N_{AS})
  for t = 0 to H **do** ▷ Simulate action sequences and calculate their costs
    $\boldsymbol{a_t} = AS[t]$
    $\boldsymbol{s_t} = \text{clip}(\boldsymbol{s_t}, s_{min}, s_{max})$
    $\boldsymbol{a_t} = \text{clip}(\boldsymbol{a_t}, a_{\text{min}}, a_{\text{max}})$
    Predict next state $\boldsymbol{s_{t+1}}$ using model
    $\boldsymbol{s_t} = \text{clip}(\boldsymbol{s_t}, s_{\text{min}}, s_{\text{max}})$
    if problem is Panda reach, Panda reach dense, or MuJoCo reacher **then**
      costs+ = $C(\boldsymbol{s_t}, t, H, \boldsymbol{a_t}, s_{goal})$
    else
      costs+ = $C(\boldsymbol{s_t}, t, H, \boldsymbol{a_t})$
    end if
  end for
  if min(costs)<cost_{min} **then** ▷ Update best cost value and best action sequence
    $cost_{min} \leftarrow \min(costs)$
    $AS_{best} \leftarrow \text{argmin}(costs)$
  end if
  $I_{top} \leftarrow \text{argsort(costs)}[: N_{top}]$ ▷ Indexes of top particles
  $AS_{top} \leftarrow AS[I_{top}]$ ▷ Top particles
  Update action sequences
end for
Return AS_{best} and $cost_{min}$

Particle filtering

After an MPC iteration, our PF samples with replacement among the top N_{top} particles to generate a new set of N_{AS} particles. In the discrete action case, PF associates a probability of change p to each action in a sequence. The probability of change distribution is then sampled to associate a True value if an action must be changed and a False value if it must not. If the action must be changed, a random one is chosen. For continuous action space problems, a noise sampled from a Gaussian distribution $\mathcal{N}(0, \sigma)$ is added to the particles, similar to what is done in MPPI [43, 44] and POPLIN [24].

Algorithm 10 Pseudocode for particle filtering (PF)

Require: N_{AS}: Number of top particles, AS_{top}: N_{top} top particles, p: Probability of change (discrete actions), N_a: Number of possible actions (discrete actions), $\mathcal{N}(0, \sigma)$: Gaussian noise (continuous actions)

A particle is a unidimensional array of length $[H \cdot a_{dim}]$
AS_{top} is of shape $(N_{top}, H \cdot a_{dim})$
$AS \leftarrow \{a^{(1)}, a^{(2)}, ..., a^{(i)}, ..., a^{(N_{AS})}\}$, where $i \sim \mathcal{U}(0, N_{AS})$ and $a^{(i)} = AS_{top}[i]$
if discrete action space **then**
    mask $\sim$ Bern(p)
    AS[mask] $\leftarrow$ randint(0, N_a)
else ▷ Continuous action space
    $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
    $AS \leftarrow \text{clip}(AS + \varepsilon, a_{low}, a_{max})$
end if

Cross-entropy method

For discrete actions, our CEM sums the number of times each action was taken at a given time step t. It calculates the probability of occurrence of each action by dividing the number of occurrences in which an action was selected by the total number of particles N_p. To avoid the effect of an action never appearing in the count of a given time step, one is added to all action counts. This is called Laplace smoothing [56]. New particles can then be chosen by sampling the probability distribution of actions for each time step of the time horizon.

Algorithm 11 Pseudocode of the cross-entropy method (CEM) for discrete actions

Require: AS_{top}: N_{top} top particles, N_a: Number of possible actions (discrete actions)

$\alpha = 1$ ▷ Laplace alpha
new$AS \leftarrow$zeros(N_p, H)
for t = 0 to H **do**
    $c_t = \text{bincount}(AS_{top}[:, t], \text{minlength} = N_a)$ ▷ Ex: c_t =dict(0:1, 1:2, 2:0)
    $c_{t,\text{smoothed}} = c_t + \alpha$ ▷ Ex: c_t =dict(0:2, 1:3, 2:1)
    $p_t = \frac{c_{t,\text{smoothed}}}{\text{sum}(c_{t,\text{smoothed}})}$ ▷ Ex: p_t =dict(0:1/3, 1:2, 2:1/6)
    new$AS[:, t] \sim p_t$
end for

In the continuous case, a Gaussian distribution is used based on the mean and variance, or covariance, depending on the dimensionality, of the particles from the previous MPC iteration, as is done in this GitHub repository [57]. This distribution can then be sampled to generate new action sequences. It is also possible to add noise to the variance or covariance, where the noise is a single value in the variance case or an identity matrix multiplied by a noise factor in the covariance matrix case [40, 57]. The noise can be constant or decrease

with the optimization process.

Algorithm 12 Pseudocode of the cross-entropy method (CEM) for continuous actions

Require: AS_{top}: N_{top} top particles, a_{dim}: Dimension of action space
1: $\mu \leftarrow \text{mean}(AS_{top}, \text{axis=0})$
2: **if** $a_{dim} > 1$ **then**
3: $\quad \Sigma \leftarrow \text{cov}(AS_{top}, \text{rowvar=False, bias=True})$
4: $\quad$ new $AS \sim \mathcal{N}(\mu, \Sigma)$
5: **else**
6: $\quad \sigma = \text{std}(AS_{top}, \text{axis=0})$
7: $\quad$ new $AS \sim \mathcal{N}(\mu, \sigma^2)$
8: **end if**

Figure 3.5 recaps the ways to optimize action sequences after an MPC iteration for continuous and discrete action spaces:

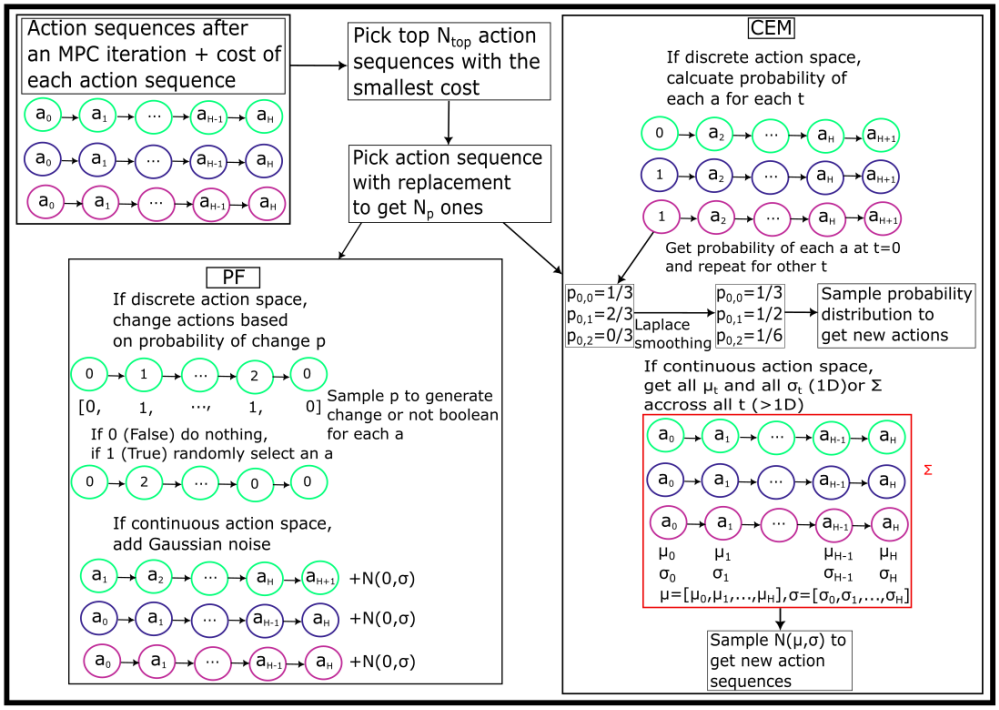


Figure 3.5 How to optimize action sequences in MPC for continuous and discrete action spaces

3.1.5 The different MPC methods

QRNN-ASNN-PF/CEM

Our proposed method is a model predictive control (MPC) approach utilizing:

- A quantile regression neural network (QRNN) as a learned stochastic model of the environment transitions,
- A neural network to generate action sequences (ASNN),
- Particle filtering (PF) or the cross-entropy method (CEM) to optimize action sequences in MPC.

The Figure 3.6 recaps the proposed QRNN-ASNN method:

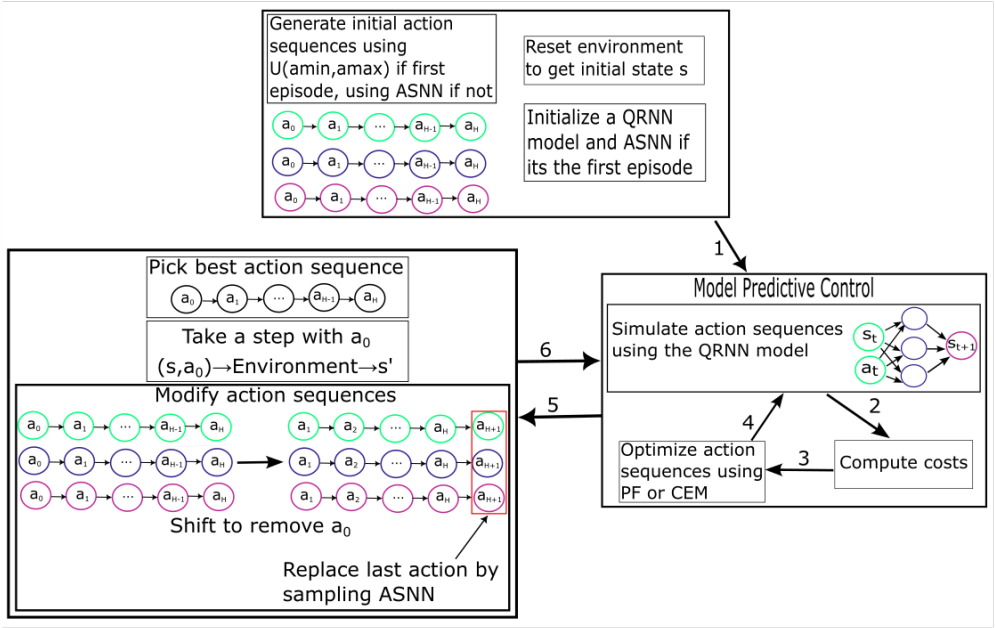


Figure 3.6 Visualization of QRNN-ASNN

Simplifications of the proposed method

We now consider the multiple simplifications of the QRNN-ASNN-PF/CEM method.

We replaced the QRNN with the ASNN and the MSENN. This led to the 50NN-ASNN-PF/CEM and MSENN-ASNN-PF/CEM methods.

The methods model-basic-PF/CEM, model-rnd-PF/CEM, and model-RS all generate their initial action sequences using a uniform distribution instead of the ASNN, where model is one of the ones defined in 3.1.2. The difference between these three methods lies in the method used to modify the action sequences after taking a step in the environment, as well as the number of MPC optimization iterations.

The model-basic-PF/CEM algorithms perform multiple MPC optimization iterations. They use a method that shifts particles after each environment step and replaces vacant actions with ones sampled from a uniform distribution.

The model-RS algorithms generate new action sequences at each environment step. They also only do one MPC iteration, so the action sequences are never optimized.

The model-rnd-PF/CEM algorithms are identical to the model-RS methods, but they do multiple MPC optimization iterations.

3.2 RL benchmark environments

The OpenAI Gymnasium Classic control, MuJoCo, and Panda Gym environments were used to test the different algorithms. Below are images and descriptions of each problem, along with the cost functions used in MPC.

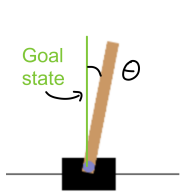
3.2.1 Images of the envs

The different algorithms mentioned were tested on the following OpenAI Gymnasium environments: Cart Pole (continuous and discrete), Acrobot, Pendulum, Lunar Lander (continuous and discrete), and Mountain Car (continuous and discrete), as well as the Reacher and Pusher environments from Panda Gym and OpenAI's MuJoCo. Images of each environment are shown in Fig. 3.7.

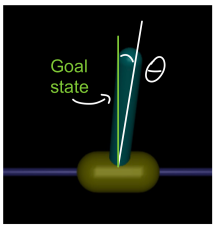
3.2.2 Environment descriptions

Cart Pole (continuous and discrete action space)

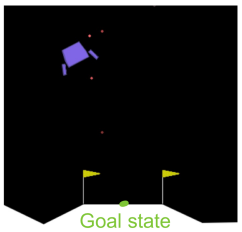
The Cart Pole environment consists of a cart with a pole attached to its center [2]. The cart can move freely from side to side, and the pole will rotate depending on the cart's movements. The goal is to balance the pole for as long as possible. The original Cart Pole environment has discrete actions and continuous states. It is also possible to adapt the environment to have continuous actions [59]. Each episode terminates if the angle of the pole to the vertical is larger than $\pm 12°$, if the position of the cart is larger than ± 2.4 horizontal units. The v1 version of the environment was used, but the truncated condition was capped at 200 to accelerate the tests. The agent is considered to have learned when it can consistently attain the 200 steps per episode mark for 100 consecutive episodes. The reward at each timestep is $+1$, which includes the last timestep. The state components are the cart's horizontal position x, horizontal velocity v, angle of the pole to the vertical θ, and angular velocity of the pole



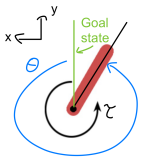
(a) Cart pole [2]



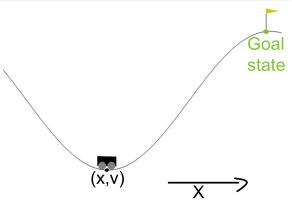
(b) Inverted Pendulum [2]



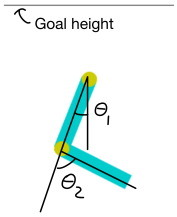
(c) Lunar lander [2]



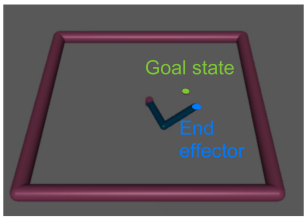
(d) Pendulum [2]



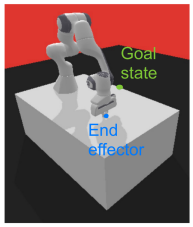
(e) Mountain car [2]



(f) Acrobot [2]



(g) MuJoCo Reacher [2]



(h) PandaGym Reach [58]

Figure 3.7 Images of the benchmark RL environments

ω. In the discrete action space case, the action a consists of either pushing the cart to the left with the action 0 or pushing it to the right with action 1. The magnitude of the applied force F is a contant value of 10 with arbitrary units. This means that for the action [2]:

- 0: The cart will be pushed to the left with a force of -10
- 1: The cart will be pushed to the right with a force of 10

In the continuous action space case, the action a is the force applied to the cart, and its value is between -1 and 1 [60].

In both cases, the [2]:

- Cart position x is bounded between -4.8 and 4.8
- Cart velocity v is unbounded
- Pole angle θ is bounded between $-24°$ and $24°$
- Pole angular velocity ω is unbounded

The table 3.1 gives the remaining values of important physics variables of the Cart Pole environment:

Table 3.1 Cart Pole environment physics variables

Variable	Value
Gravity (g)	9.8
Cart mass (m_c)	1.0
Pole mass (m_p)	0.1
Half of the pole's length (l_p)	0.5

Inverted Pendulum

The Inverted Pendulum environment follows the same idea as the Cart Pole environment, but the maximum episode length is 1000 steps, and the action space is continuous [2]. Also, the pole must stay in the angle range of ± 0.2 radians for the episode to continue. The state components are the same as for the Cart Pole environment and they are all unbounded.

Lunar Lander (continuous and discrete action space)

The Lunar Lander environment is a rocket in space that needs to land between two flags on a hill [2]. The rocket has a central main engine and two lateral engines that can be activated. For discrete actions, the agent decides not to activate an engine or to activate one of the three possible ones. For continuous actions, the agent decides the force to be applied to the main engine and one of the lateral engines. No truncation condition is defined; however, the episode terminates if the rocket exits the environment domain or the agent crashes. The reward at each time step depends on multiple factors, including the rocket's distance from the ideal landing position, its tilt, and the number of frames the engine is firing.

The state components are the 2D position (x, y), 2D velocity (v_x, v_y), velocity v, angular velocity ω, and two variables (left_contact, right_contact) that are True when the respective leg of the rocket touches the ground. Here are the bounds of each state component [2]:

- The rocket's 2D position (x, y) is bounded between -2.5 and 2.5 for each component
- The rocket's 2D velocity (v_x, v_y) is bounded between -10 and 10 for each component
- The angle θ is bounded between -6.2831855 and 6.2831855
- The angular velocity ω is bounded between -10 and 10

- The two (left_contact, right_contact) are either False (0) or True (1)

For a discrete action space, the action 0 is do nothing, action 1 is to fire the left engine, action 2 is to fire the main engine, and action 3 is to fire the right engine. For a continuous action space, the action $a = (a_1, a_2)$ is 2D and is bounded between -1 and 1. a_1 is the throttle of the center engine and a_2 is for the throttle of either the left or right engine. The following piece-wise function discribes the throttle of the center and lateral engines depending on the value of a_1 and a_2:

$$\text{engines throttle} = \begin{cases} \text{center engine is off,} & \text{if } a_1 < 0, \\ \text{center engine is on and increases with } a_1, & \text{if } 0 \leq a_1 \leq 1 \\ \text{left engine is on,} & \text{if } a_2 < -0.5, \\ \text{right engine is on,} & \text{if } a_2 < 0.5, \\ \text{left and right engines are off,} & \text{if } -0. < a_2 < 0.5, \\ \text{left engine is on and increases opposite to } a_2, & \text{if } -1 \leq a_1 \leq -0.5 \\ \text{right engine is on and increases with } a_2, & \text{if } 0.5 \leq a_1 \leq 1 \end{cases}$$

The default gravity value is -10. It is possible to add wind, define the strength of the linear and rotational wind. Wind isn't activate for our tests.

Pendulum

The Pendulum environment consists of a pole fixed at one end that is free to rotate [2]. The goal is to bring the pole vertical and then balance it upwards. There is no clear episode termination condition, and the truncation condition, which determines when the episode ends, is 200 timesteps. The agent receives a reward of -1 for each time step. The first way of defining the state components is the cosine of the pole's angle $x = \cos(\theta)$, the sine of the pole's angle $y = \sin(\theta)$, and the pole's angular velocity ω. It is also possible to access the actual state components used in the environment's dynamics, which include the pole's angle to the vertical, θ, and the pole's angular velocity, ω. The action a is the torque applied to the pole and its in N·m.

Here are the bounds of the state components and the torque [2]:

- $x = \cos(\theta)$ is bounded between -1 m and 1 m

- $x = \sin(\theta)$ is bounded between -1 m and 1 m
- ω is bounded between -8 rad/s and 8 rad/s
- a is bounded between -2 N·m and 2 N·m

The table 3.2 gives the remaining values of important physics variables of the Pendulum environment:

Table 3.2 Pendulum environment physics variables

Variable	Value
Gravity (g)	9.8
Pole mass m	1.0
Pole mass l	0.1

The reward function is given in Equation 3.8:

$$r = -(\theta^2 + 0.1\omega^2 + 0.001a^2) \tag{3.8}$$

Mountain Car (discrete and continuous action space)

The Mountain Car consists of a car that starts at the bottom of two hills [2]. The goal is to reach the top of the right side of the mountain. To do so, the agent must move up the left side of the hill to gain enough momentum to go up the right side of the mountain. The goal position is 0.45 for the continuous action space Mountain Car and 0.5 for the discrete version of the environment. By moving up the left hill, the agent moves away from the goal state, which exemplifies a classic demonstration of delayed rewards. The environment has both discrete and continuous action space versions. In both cases, the states are continuous. The reward at each timestep is -1 for the discrete version. For the continuous version, the reward at each timestep is $-0.1 \cdot a^2$, where a is the action. A reward of +100 is also given when the agent reaches the goal. The state components are the horizontal position, x, and the velocity of the car, v. The action a is the force applied to the cart. When the action space is discrete:

- Action 0 will move the cart to the left with a force $F = 0.001$

- Action 1 will do nothing
- Action 2 will move the cart to the right with a force $F = 0.001$

When the action space is continuous, the action $a = F$ will be a force with a continuous value between -1 and 1.

Here are the bounds of the state components [2]:

- The horizontal position of the car x is bounded between -1.2 and 1.2
- The horizontal velocity of the car v is bounded between -0.07 and 0.07

The gravity used is 0.0025.

Acrobot

The goal of the Acrobot environment consists of two poles connected at a joint, and the system is fixed at one end of the top pole [2]. The goal is to bring the bottom part of the lower pole up to a height of 1. The action a consists of applying a torque to the link between the two poles. The state and action spaces are continuous. The state components are the cosine of θ_1, the sine of θ_1, the cosine of θ_2, the sine of θ_2, the angular velocity of θ_1 and the angular velocity of θ_2 where θ_1 is the angle the top pole makes with the vertical axis pointed downwards and θ_2 is the angle the bottom pole makes with the top pole. The possible actions are [2]:

- 0: Torque of -1 to the joint connecting the two poles
- 1: No torque is applied
- 2: Torque of 1 to the joint connecting the two poles

The bounds of the state components are [2]:

- $\cos(\theta_1)$, $\sin(\theta_1)$, ,$\cos(\theta_2)$, and $\sin(\theta_2)$ are bounded between -11 and 1
- ω_1 and ω_2 are bounded between -4π and 4π

The table 3.3 gives the remaining values of important physics variables of the Acrobot environment:

The agent receives a reward of -1 for each time step, and the optimal reward is -100.

Table 3.3 Acrobot environment physics variables

Variable	Value
Gravity (g)	9.8 m/s^2
Top link mass m_1	1.0 kg
Bottom link mass m_2	1.0 kg
Top link length l_1	1.0 m
Bottom link length l_2	1.0 m
Center of mass position of the top link l_{c1}	0.5 m
Center of mass position of the top link l_{c2}	0.5 m

Reacher - Mujoco and Panda Gym

The MuJoCo Reacher environment is a 2D robotic arm with one end fixed on a surface and the other end free to move [2]. The goal is to bring the end of the free robotic arm to a goal position. The state space is 10d and consists of the sine, the cosine, and the angular velocity of both joints of the robotic arm, the x- and y-coordinates of the target position, and the distance in x and y between the robotic arm's tip and the goal position.

The Panda Gym Reach environment consists essentially of a 3D version of the robotic arm reaching task [58]. The state space is 6D and consists of the 3D Cartesian position and 3D velocity of the robotic arm's free end. The action is the 3D movement of the free end of the robot arm. The reward structure can either be sparse or dense. For sparse rewards, a reward is only given when the agent succeeds in the task. For dense rewards, a reward is given at each time step. In this case, the reward is the negative of the distance between the agent's current free end arm position and the goal position.

Recap of environment action space type

The table 3.4 recaps the action space type for each environment.

3.2.3 Length of an environment time step

We now give the length of taking a step in the environment using env.step(action) for the different benchmark environments. We will use the standard convention of calling it dt.

Table 3.4 Recap of action space types

Environment	**Action space** C: continuous D: discrete	N_{steps}
Acrobot	D	200
Cart pole	C + D	200
Inverted Pendulum	C	1000
Moutain car continuous	C	500
Moutain Car Discrete	D	200
Lunar Lander	C + D	1000
Pendulum	C	200
MuJoCo Reacher	C	50
Panda Reach	C	50

Cart Pole

For the Cart Pole enviroment, the dt is defined as tau and is worth 0.02 s [2].

Acrobot

The dt for the Acrobot environment is 0.2 s [2].

Pendulum

The dt for the Pendulum environment is 0.05 s [2].

Inverted Pendulum and MuJoCo Reacher

For the Inverted Pendulum, MuJoCo Reacher, and all the other MuJoCo environments, the dt is defined as in Equation 3.9:

$$\text{dt} = \frac{1}{\text{frame\_skip}} \times \text{model.opt.timestep} \tag{3.9}$$

For the Inverted Pendulum and MuJoCo Reacher environments, frame_skip = 2 s and

model.opt.timestep = 0.002 s. The lenght of a time step is therefore dt = 0.004 s [2].

Panda Reach

For the Panda Gym environments, the dt is obtained by multiplying the number of time steps run by the simulation and the duration of a simulator time step as presented in Equation 3.10 [58]:

$$\text{dt} = \text{number\_simulator\_time\_steps} \times \text{duration\_of\_time\_step} = 20 \times 2 \cdot 10^{-3} \text{ s} = 0.1 \text{ s} \tag{3.10}$$

Mountain Car, and Lunar Lander

There is no defined dt value for the Mountain Car, and Lunar Lander environments [2].

Recap of environment time step length

The table 3.5 recaps the action space type for each environment.

Table 3.5 Recap of action space types

Environment	**Action space** C: continuous D: discrete	N_{steps}	dt (s)
Acrobot	D	200	0.2
Cart pole	C + D	200	0.02
Inverted Pendulum	C	1000	0.004
Moutain car continuous	C	500	-
Moutain Car Discrete	D	200	-
Lunar Lander	C + D	1000	-
Pendulum	C	200	0.05
MuJoCo Reacher	C	50	0.004
Panda Reach	C	50	0.1

3.2.4 Dynamics of the environments

We will now go over the equations representing the dynamics of each environment.

Cart Pole and Inverted Pendulum

For the Cart Pole and Inverted Pendulum environments, as presented in Florian's technical report [61], the linear acceleration $a_x = \ddot{x}$ and the angular acceleration $\alpha = \ddot{\theta}$ are as given in Equations and [2, 61]:

$$\ddot{\theta} = \frac{g \sin(\theta) + \cos(\theta)\left(\frac{-F - m_p l \dot{\theta}^2 \sin(\theta)}{m_c + m_p}\right)}{l\left(\frac{4}{3} - \frac{m_p \cos^2(\theta)}{m_c + m_p}\right)} \tag{3.11}$$

$$\ddot{x} = \frac{F + m_p l\left(\dot{\theta}^2 \sin(\theta) - \ddot{\theta}\cos(\theta)\right)}{m_c + m_p} \tag{3.12}$$

The next state components $(x, v = \dot{x}, \theta, \omega = \dot{\theta})$ are then calculated using Euler [62] as an integration method as given in Equations 3.13, 3.14, 3.15, and 3.16 [2]:

$$x \leftarrow x + \mathrm{dt} \cdot \dot{x} \tag{3.13}$$

$$\dot{x} \leftarrow \dot{x} + \mathrm{dt} \cdot \ddot{x} \tag{3.14}$$

$$\theta \leftarrow \theta + \mathrm{dt} \cdot \dot{\theta} \tag{3.15}$$

$$\dot{\theta} \leftarrow \dot{\theta} + \mathrm{dt} \cdot \ddot{\theta} \tag{3.16}$$

Lunar Lander

The dynamics of the Lunar Lander environments are quite complex. We suggest the interested reader to look at the Gymnasium source code: `https://github.com/Farama-Foundation/Gymnasium/blob/main/gymnasium/envs/box2d/lunar_lander.py`.

Pendulum

The Pendulum task dynamics are described in Equations and .

$$\dot{\theta} \leftarrow \dot{\theta} + \left(\frac{3g}{2l}\sin(\theta) + \frac{3}{m \cdot l^2}a\right)dt \tag{3.17}$$

$$\theta \leftarrow \theta + \dot{\theta} \cdot dt \tag{3.18}$$

The new state is obtained with: $(x, y, \omega) = (\cos(\theta), \sin(\theta), \omega)$.

Mountain Car

The dynamics of the discrete action space version of the Mountain Car environment are given in Equations 3.19 and 3.20:

$$v \leftarrow v + (a-1) \cdot F + \cos(3x) \cdot (-g) \tag{3.19}$$

$$x \leftarrow x + v \tag{3.20}$$

The continuous action space Mountain Car environment dynamics are presented in Equations 3.21 and 3.22:

$$v \leftarrow v + F \cdot P - g \cdot \cos(3x) \text{ where the Power } P \text{ is worth } 0.0015 \tag{3.21}$$

$$x \leftarrow x + v \tag{3.22}$$

Acrobot

The dynamics of the Acrobot Equation based on Sutton's work [9,63] are given in Equations 3.23, 3.24 with d_1, d_2, ϕ_1, and ϕ_2 defined in Equations 3.25, 3.26, 3.27, and 3.28.

$$\ddot{\theta}_1 = -d_1^{-1}(d_2\ddot{\theta}_2 + \phi_1) \tag{3.23}$$

$$\ddot{\theta}_2 = \Big(m_2 l_{c2}^2 + I_2 - \frac{d_2^2}{d_1}\Big)^{-1}\Big(\tau + \frac{d_2}{d_1}\phi_1 - \phi_2\Big) \text{ where } \tau = a \text{ the action taken in the environment} \tag{3.24}$$

$$d_1 = m_1 l_{c1}^2 + m_2\Big(l_1^2 + l_{c2}^2 + 2 l_1 l_{c2}\cos(\theta_2)\Big) + I_1 + I_2 \tag{3.25}$$

$$d_2 = m_2\Big(l_{c2}^2 + l_1 l_{c2}\cos(\theta_2)\Big) + I_2 \tag{3.26}$$

$$\phi_1 = -m_2 l_1 l_{c2}\dot{\theta_2}^2 \sin(\theta_2) - 2 m_2 l_1 l_{c2}\dot{\theta}_2\dot{\theta}_1 \sin(\theta_2) + (m_1 l_{c1} + m_2 l_1) g\cos(\theta_1 - \pi/2) + \phi_2 \tag{3.27}$$

$$\phi_2 = m_2 l_{c2} g\cos(\theta_1 + \theta_2 - \pi/2) \tag{3.28}$$

The system of equations is solved using a 4th order Runge-Kutta method [62].

MuJoCo Reacher and Panda Reach

The MuJoCo Reacher and Panda Reach environments do not provide clear dynamics.

3.2.5 Cost function used in MPC for each environment

Below are the cost functions used for the different MPC, MBRL, and trajectory optimization methods used later on. It is worth noting that we chose to clip the state components to the acceptable range before being passed to the cost function as a hard constraint.

Cart Pole and Inverted Pendulum

For the Cart Pole and the Inverted Pendulum environments, the cost function is given in Equation 3.29:

$$C = \sum_{t=0}^{H-1}(\theta_t^2 + 0.1 \cdot x_t^2 + 0.1 \cdot v_t^2) \tag{3.29}$$

The goal is to balance the pole and not stray too far from the center, which explains the first two terms of the cost function that we will want to minimize. The third term represents the idea that smaller velocities will allow the agent to balance the pole more easily.

Acrobot

For the Acrobot environment, the distance of the height of the free end of the bottom pole from the goal height of 1 is important, so the cost function is described in Equation 3.30:

$$C = \sum_{t=0}^{H-1} \left[1 + \cos\left(\theta_{1,t}\right) + \cos\left(\theta_{1,t} + \theta_{2,t}\right)\right]^2 \tag{3.30}$$

When $\theta_{1,t} = 0$, the top pole will be oriented downwards, and when $\theta_{2,t} = 0$, the bottom pole will align with the top pole. The smallest possible cost is -1 which is when $\theta_{1,t} = \pi$ and $\theta_{2,t} = 0$.

Mountain Car

For the mountain car environment, the goal is to reach the goal state located at the top right of the hill, which is achieved when the agent's horizontal position exceeds x_g. We will then want to minimize the agent's distance from the goal, but we will add an inverse discount factor to encourage the agent to get as close as possible to the goal near the end of the episode. This allows the agent to move away from the goal early on and eventually approach it more closely as the episode progresses without penalizing it for doing so prematurely. The cost function used is the following Equation 3.31:

$$C = \sum_{t=0}^{H-1} \gamma^{H-t-1} \cdot (x_t - x_g)^2 \tag{3.31}$$

For the discrete environment, $x_g = 0.45$ and $x_g = 0.5$ for the continuous version. Depending on the chosen γ value, we can consider more or less the importance of the different steps along the prediction horizon. For example, when $\gamma = 0$, only the cost of the last step is considered. Multiple γ values were tested (0, 0.5, 0.99). The value $\gamma = 0.5$ was chosen based on my tests.

For the continuous Mountain Car environment, the GP-MPC method repeated the action found using MPC 4 times. This led to excellent performance. In section , we compare the performance of our MPC methods, CEM, iCEM, and MPPI, when we take four steps with the optimized action and when we only take one.

Lunar Lander

To successfully solve the task, the rocket must minimize its distance from the center position of the landing pad, situated at (0, 0), land with as little angle as possible, and turn off its thrusters once it has landed. The cost function for the discrete action space version of the environment is as described in Equation 3.32:

$$C = \sum_{t=0}^{H-1} x_t^2 + y_t^2 + 0.1(v_{x,t}^2 + v_{y,t}^2) + 0.3(\theta_t^2 + \omega_t^2) - 10(\text{LL} + \text{RL}) \tag{3.32}$$

For the continuous action space version of the environment, we add the actions in the cost function of Equation 3.33:

$$C = \sum_{t=0}^{H-1} x_t^2 + y_t^2 + 0.1(v_{x,t}^2 + v_{y,t}^2) + 0.3(\theta_t^2 + \omega_t^2) + 0.001(a_1^2 + a_2^2) - 10(\text{LL} + \text{RL}) \tag{3.33}$$

where a_1 is the thrust applied to the main engine and a_2 is the one associated with the side engines.

The specific value of the weights used is arbitrary; what matters in our cost functions is the relative importance of the variables based on the associated weights. The LL and RL parameters represent the booleans for the left and right legs of the rocket being in contact with the ground.

Here's the chosen importance of the different quantities in decreasing order:

- The contact of the legs with the ground
- The distance to the center
- The angle and angular velocity
- The linear velocity
- The action applied

Pendulum

For the pendulum env, minimizing θ, ω, and τ is important, so the cost function used is defined in Equation 3.34:

$$C = \sum_{t=0}^{H-1} \gamma^{(H-t-1)}(\theta_t^2 + 0.1 \cdot \omega_t^2 + 0.01\tau^2) \tag{3.34}$$

The inverse discounting $\gamma^{(H-t-1)}$ gives more importance to the later costs than the early ones in the prediction horizon. This is important since the agent should be closer to the goal position of being vertical later on in the time horizon. The parameter $\gamma = 0.99$ was chosen because it yielded promising results.

MuJuCo Reacher and Panda Gym Reach

For the MuJoCo Reeacher and Panda Gym Reach problems, we aim to minimize the distance between the current position of the free end of the robot arm, $P_{e,t}$, and the goal position, d_g. For MuJoCo Reacher, we also want to especially minimize excessive and large movements. The MPC cost functions used for each environment are given in Equations 3.35 and 3.36:

$$C = \sum_{t=0}^{H} \|P_{e,t} - P_g\| + 0.1(a_{x,t}^2 + a_{y,t}^2) = \sum_{t=0}^{H} \sqrt{(x_{e,t} - x_g)^2 + (y_{e,t} - y_g)^2} + 0.1(a_{x,t}^2 + a_{y,t}^2) \quad (3.35)$$

$$C = \sum_{t=0}^{H-1} \|P_{e,t} - P_g\| = \sum_{t=0}^{H} \sqrt{(x_{e,t} - x_g)^2 + (y_{e,t} - y_g)^2 + (z_{e,t} - z_g^2)} \quad (3.36)$$

Recap of cost functions used in MPC

The table 3.6 recaps the cost functions used in MPC for each environment.

3.3 Description of the tests

3.3.1 Test of the validity of QRNN model's quantile predictions

To test the quantile predictions of the QRNN, 20,000 steps were taken using a randomly sampled action in each environment. At each step, the prediction provided by the QRNN for each state component was compared to the actual next state prediction of the environment. When the true state component value was below that of a given quantile, a counter was augmented by 1, and then the value of the counter at this step was divided by the total number of steps up to that point. This calculation aims to determine the percentage of time the different quantile predictions exceed those of the environment. Pseudocode 13 explains this.

Table 3.6 Recap of cost functions

Environment	**Cost function** $C = \sum_{t=0}^{H-1}$
Acrobot	$\left[1 + \cos(\theta_{1,t}) + \cos(\theta_{1,t} + \theta_{2,t})\right]^2$
Cart Pole	$\theta_t^2 + 0.1 \cdot x_t^2 + 0.1 \cdot v_t^2$
Inverted Pendulum	$\theta_t^2 + 0.1 \cdot x_t^2 + 0.1 \cdot v_t^2$
Mountain Car	$\gamma^{(H-t-1)}(x_t - x_g)^2$
Lunar Lander discrete	$x_t^2 + y_t^2 + 0.1(v_{x,t}^2 + v_{y,t}^2)$ $+ 0.3(\theta_t^2 + \omega_t^2) - 10(\mathrm{LL} + \mathrm{RL})$
Lunar Lander continuous	$x_t^2 + y_t^2 + 0.1(v_{x,t}^2 + v_{y,t}^2)$ $+ 0.3(\theta_t^2 + \omega_t^2) + 0.001(a_1^2 + a_2^2)$ $- 10(\mathrm{LL} + \mathrm{RL})$
Pendulum	$\gamma^{(H-t-1)}(\theta_t^2 + 0.1 \cdot \omega_t^2 + 0.01\tau_t^2)$
MuJoCo Reacher	$\|P_{e,t} - P_g\| + 0.1(a_{x,t}^2 + a_{y,t}^2)$
Panda Reach	$\|P_{e,t} - P_g\|$

Algorithm 13 Pseudocode of the QRNN next state prediction compared to the environment's

Require: i: Number of state components, j: Number quantiles, s_{env}: Environment true next state, $s_{quantile}$: QRNN predicted next state

counter$_0$=0, counter$_1$=0, ..., counter$_j$=0 ▷ Initialize a counter for each quantile
for each state component i **do**
    for each quantile j **do**
        if $s_{env,i} < s_{quantile_j,i}$ **then** ▷ Take the ith state component of the jth quantile
            counter$_j$ += 1
        end if
        Save counter$_j/(i+1)$
    end for
end for

3.3.2 Hyperparameter testing of noise levels in MPC particle filtering

Our different MPC-based methods were run on the benchmark environments for seeds 0, 8, and 15. Different probabilities of change (p) were tested for the different MPC methods applied to problems with discrete action spaces, and the one that seemed the most effective was chosen. Similarly, these tests were repeated with the different standard deviations σ for the MPC methods applied to continuous action spaces.

3.3.3 Comparison of methods

The methods that are compared in the results are now given and we explain why certain methods mentioned previously are not included.

QRNN-ASNN-PF/CEM and its ablations

Our different MPC methods utilize three models of the environment (QRNN, 50NN, or MSENN) and either particle filtering (PF) or the cross-entropy method (CEM) to modify the action sequences in MPC. The general groups of methods are: model-ASNN-PF, model-basic-PF, model-rnd-PF, model-ASNN-CEM, model-basic-CEM, model-rnd-CEM, and model-RS, where the possible choices for model are defined in Section 3.1.2. This results in 21 different MPC methods that we test, comprising the QRNN-ASNN method and its various ablations. For discrete action space problems, the probability of change p used in particle filtering is set to 0.1. For continuous action space problems, the standard deviation σ as part of the Gaussian noise added to the particles is 0.3.

MFRL

The standard MFRL algorithms used for comparison later on are DQN [17], IV-DQN [30], QR-DQN [29], SAC [18], TD3 [20], DDPG [19], TQC [31], PPO [21], and A2C [64].

DQN [17] and IV-DQN [30] are the latter author's implementations [65]. They only run their code for the discrete versions of the Mountain Car and the Lunar Lander environment. They specify DQN and IV-DQN hyperparameters for both, but we ultimately used the default hyperparameters for Lunar Lander. This is because the performance did not change for Mountain Car and was worse when using their chosen hyperparameters. Similarly, since the authors did not run DQN and IV-DQN for Acrobot and Cart Pole, Iwe decided to use the default hyperparameter values. We made this choice since the hyperparameter tuning method described by the authors is quite exhaustive, and they did not provide any code for their hyperparameter tuning method. Here's an excerpt of the authors' hyperparameter tuning method [30]:

> For every result presented in this paper, the hyperparameters for each algorithm were tuned using grid search. Each combination of hyperparameters was run five times, with different seeds on both initialization and environment. The best 3 or 4 configurations were then selected to be run 25 times - this time, the combinations of 5 environment and 5 initialization seeds. The configuration selected to be shown in the paper is the best of these configurations based on the 25 runs

QR-DQN [29] is from an unofficial reimplementation of the method [66]. We ran their code as is, besides changing the environment.

The other RL methods are from the stable-baseline3 (sb3) package [67]. The hyperparameters used for our tests are from sb3's Hugging Face page [68]. Appendix A lists the different hyperparameters used for clarity. Some of the given hyperparameters are no longer used by sb3, such as normalize, noise_std, noise_type, normalize_kwargs, learning_starts, action_noise, and env_wrapper. The hyperparameters were provided for the Acrobot, Cart Pole, Lunar Lander, Mountain Car, Pendulum, and the sparse reward version of Panda Reach. The default algorithm hyperparameters were assumed for the other environments. It is worth noting that only the TQC hyperparameters were provided for the Panda Reach environment. For most environments for which hyperparameters were provided, we were able to use the same hyperparameters as those used in our tests for the corresponding environment version. However, for the Lunar Lander environments, the hyperparameters were

specified for the v2 version of the environment, not the v3 version. We ran the algorithms on the v2 version of the discrete Lunar Lander environment and on the v3 version of the continuous Lunar Lander environment, as v2 is deprecated. We therefore assumed that the hyperparameters for the v2 version of the continuous Lunar Lander were acceptable to use for the v3 version. The hyperparameters for the continuous Cart Pole were assumed to be the same as those for the discrete version of the environment, as they were not provided, and the environment is essentially the same, except that it has continuous instead of discrete actions. This means that only A2C and PPO have the sb3-defined hyperparameters for the continuous Cart Pole environment. The other algorithms use the default ones.

MPC-based and trajectory sampling

The MPC and trajectory sampling methods used as benchmarks are PETS-CEM [23], MPPI [43, 44], GP-MPC [33, 35], CEM [38, 39], and MPC-iCEM [41].

PETS-CEM [23] is from the Facebook archived MB-RL library [59]. The authors implemented their code for the continuous Cart Pole environment. We use the code as is, besides modifying the environment and the cost function used. An important note is that the cost functions used for the Pendulum and Mountain Car continuous environments for PETS-CEM are the ones given earlier but without the γ discount factor term, since the code did not allow to simply integrate the time step t into the cost function.

MPPI [43, 44] is from the Pytorch-MPPI package [54], CEM [39] is from the Pytorch-CEM repository [42], and iCEM [41] is from the Pytorch-iCEM package [42]. These three algorithms are from the same author, who implemented them for the Pendulum environment. They specify that the important hyperparameters are the terminal_state_cost, lambda_, num_samples, and noise_mu. We do not use a different cost function or a bonus for the final state therefore we do not use the terminal_state_cost parameter. The authors suggest to use 1e-2 for lambda_, so that is what we used. Num_samples is the number of action sequences to test. We use the same amount as we used for QRNN-ASNN and its ablations. We kept the default value of 0 for noise_mu.

GP-MPC [33, 35] is from an unofficial implementation [34]. The authors implemented their code for the Pendulum and Mountain Car environments. We use their code as is, besides modifying specific parameters and the cost function based on the environment.

Algorithms not included

POPLIN-A is omitted, as we were unable to get the author's code working. Their code [60] was programmed with such specific requirements that the only way to get their code to work would be to use their computer. Even with a lot of help from Réjean Lepage in the Electrical Engineering department at l'École Polytechnique Montréal, and even running the code on a computer with a Linux version and hardware from 2015-2016, the code still did not work. The authors mentioned when they published their work that it is an unfinished product, and there does not seem to have been any advances since then, which might explain the extreme difficulty in running their code.

iLQR is omitted [47] since none of the other methods we propose or consider as benchmarks assume that the environment dynamics are known. We would use Neural iLQR if an implementation were available, but there is none. If we wanted to use iLQR using a known model of the environment, we could use this implementation [69].

Tests run

To compare the different methods, they were run on various benchmark problems with environment initialization seeds of 0, 8, and 15. The episodic returns were averaged over the seeds. Then, a moving average was applied to the last N_{mov} values to smooth out the curves, where N_{mov} varies depending on the desired level of smoothness in the collected data. The number of steps per episode and the number of episodes run depend on the problem. The table 3.7 shows the different number of steps per episode N_{steps} and the number of episodes N_{ep} run for each environment.

3.3.4 Other tests

Sampling method for QRNN next state prediction

An idea to consider the whole quantile distribution in the QRNN next state prediction would be to do the following steps, as suggested by my co-supervisor Dr. Prémont-Schwarz:

1. For a given state s_t and a chosen state a_t, we will predict a distribution over the next state s_{t+1} using the QRNN
2. The steps below will be repeated over N_{samples} and we will iterate of j:
    - Randomly choose a real number i sampled from $\mathcal{U}(0, 10($ for each particles and each state component

Table 3.7 Recap of environment properties

Environment	**Action** C: continuous D: discrete	**Cost function** $C = \sum_{t=0}^{H-1}$	N_{steps}	N_{ep}	dt (s)
Acrobot	D	$[1 + \cos\theta_{1,t} + \cos(\theta_{1,t} + \theta_{2,t})]^2$	200	300	0.2
Cart Pole	C+D	$\theta_t^2 + 0.1x_t^2 + 0.1v_t^2$	200	300	0.02
Inverted Pendulum	C	$\theta_t^2 + 0.1x_t^2 + 0.1v_t^2$	200	300	0.004
Mountain Car	C+D	$\gamma^{(H-t-1)}(x_t - x_g)^2$	200	300	-
Lunar Lander discrete	D	$x_t^2 + y_t^2 + 0.1(v_{x,t}^2 + v_{y,t}^2)$ $+ 0.3(\theta_t^2 + \omega_t^2) - 10(\mathrm{LL} + \mathrm{RL})$	1000	300	-
Lunar Lander continuous	C	$x_t^2 + y_t^2 + 0.1(v_{x,t}^2 + v_{y,t}^2)$ $+ 0.3(\theta_t^2 + \omega_t^2) + 0.001(a_1^2 + a_2^2)$ $- 10(\mathrm{LL} + \mathrm{RL})$	1000	300	-
Pendulum	C	$\gamma^{(H-t-1)}(\theta_t^2 + 0.1\omega_t^2 + 0.01\tau_t^2)$	200	300	0.05
MuJoCo Reacher	C	$\|P_{e,t} - P_g\| + 0.1(a_{x,t}^2 + a_{y,t}^2)$	50	300	0.004
Panda Reach	C	$\|P_{e,t} - P_g\|$	50	300	0.1

- Get a next state prediction $s_{t+1}^{j} = (1 - i + \lfloor i \rfloor) q_{\lfloor i \rfloor} + (i - \lfloor i \rfloor) q_{\lfloor i \rfloor + 1}$ and save it (For a given state component and particle as mentioned at 1)

3. Calculate the j costs, the next state prediction s_{t+1} is associated with smallest cost

Algorithm 14 presents the above in pseudocode form:

Algorithm 14 QRNN next state prediction by sampling the quantiles

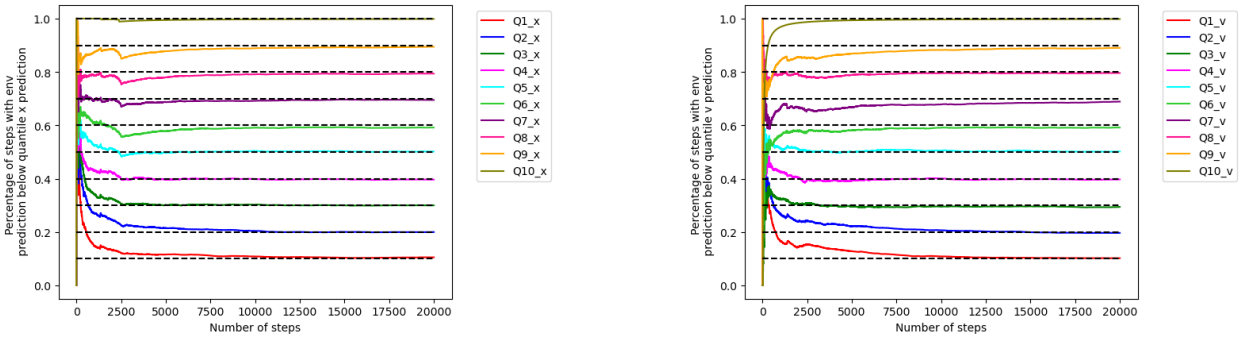
Require: QRNN: Quantile regression neural network, s: Current state, a: Action, N_Q: Number of quantiles

$s'_Q = \text{QRNN}(s, a)$ ▷ Next state quantiles
for j = 0 to N_{samples} **do**
    $s_{t+1}^{j} = [\,]$ ▷ Create tensor to store possible next states
    for $p = 0$ to N_p **do**
        for $c = 0$ to $N_{s,c}$ **do**
            $i \sim \mathcal{U}(0, N_Q)$ ▷ Select a quantile
            i_{floor}=floor(i)
            $i_{\text{ceil}} = i_{\text{floor}}$+1
            $q_{\text{low}} = s'_Q[p, c, i_{\text{floor}}]$
            $q_{\text{high}} = s'_Q[p, c, i_{\text{ceil}}]$
            $s_c = (1 + i + i_{\text{floor}}) q_{\text{low}} + (i - i_{\text{floor}}) q_{\text{high}}$
            Add s_c to element of s_{t+1}^{j} associated to the component c and the particle p
        end for
    end for
end for
Calculate costs associated with the different next states in the s_{t+1}^{j} tensor
The next state is the one with the smallest cost

CHAPTER 4 RESULTS

4.1 Quantile regression neural network predictions

The following graphs in Figure 4.1 show the quantile predictions for the x and v next state components for the discrete Mountain Car environment with these cumulative counter values, though the results shouldn't change for the continuous version. The next state prediction comparison graphs for the other environments are shown in Annexe B.



(a) x (b) v

Figure 4.1 Different quantile predictions for x and v next state components of the Mountain Car environment

The tendencies of Fig. 4.1 show a quick learning of the different quantile predictions by the QRNN model. While most quantiles only converge after around 7500 steps, the quantile tendencies learn by the model are the correct ones only after a couple of hundred steps or less. For the QRNN next-state predictions in the other environments shown in Annex B, the correct quantile prediction tendencies are learned over a varying number of episode steps.

4.2 Mean episodic return graphs

Figures 4.2, 4.3, 4.4, and 4.5 present the mean episodic reward graphs of the different algorithms applied to the benchmark environments. Fig. 4.2 shows the results for the discrete Cart Pole, Acrobot, and discrete Lunar Lander problems. Fig. 4.3 is for the discrete Mountain Car, continuous Cart Pole, and continuous Mountain Car environments. Fig. 4.4 is for the continuous Lunar Lander, Inverted Pendulum, and Pendulum problems. Fig. 4.5 shows the results for the MuJoCo Reacher, in addition to the sparse and dense reward versions of

the Panda Reach environment.

Each figure organizes data into three columns:

- MPC-PF methods
- MPC-CEM methods
- Top three algorithms from the MPC-PF and MPC-CEM methods, as well as the other benchmark algorithms.

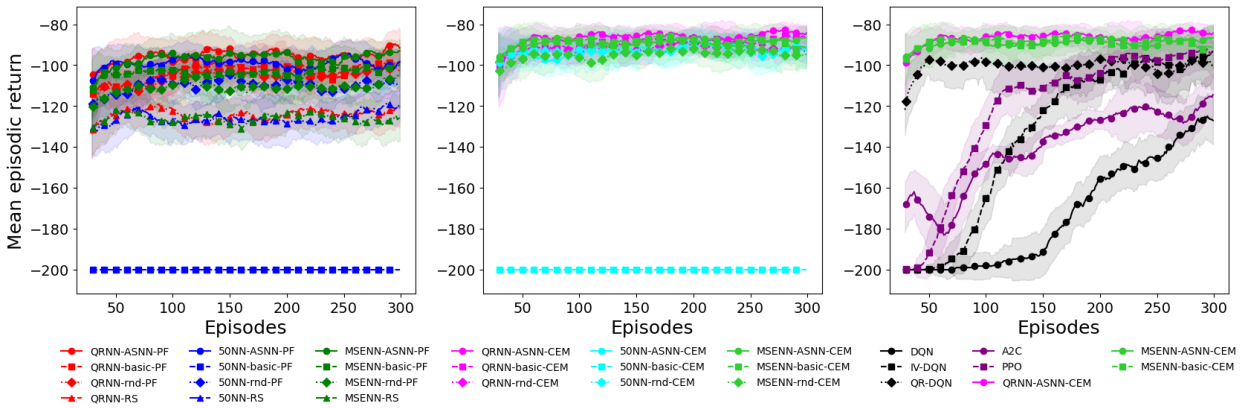
4.3 Area under the curve of the episodic return tables

Table 4.1 shows the normalized mean and standard deviation of the area under the curve (AUC) of the mean episodic return for the different algorithms applied to the benchmark environments with a discrete action space.

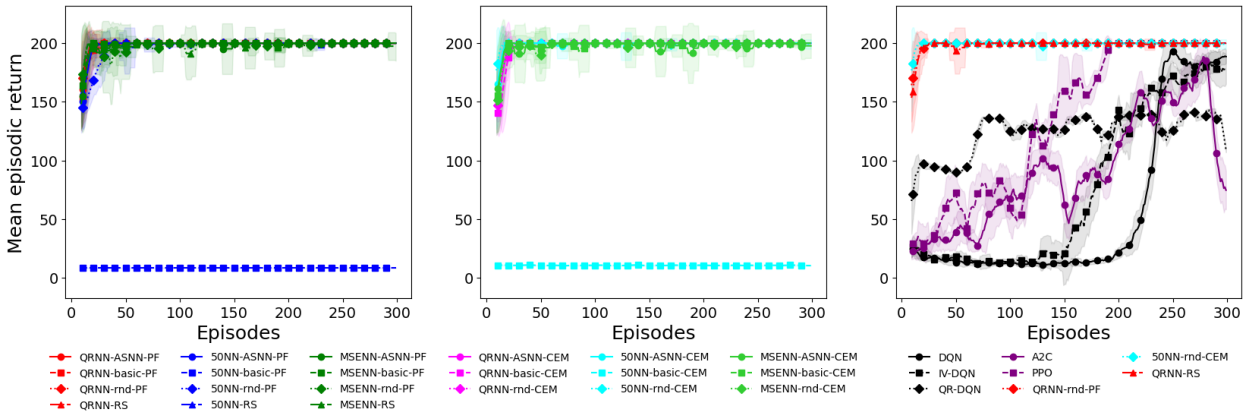
Table 4.1 Area under the curve of the episodic return for discrete action space problems

	Acrobot	Cart Pole	Lunar Lander	Mountain Car
QRNN-ASNN-PF	−28689.00 ± 1296.94	59305.33 ± 699.56	−48587.72 ± 17147.57	−59799.33 ± 0.94
QRNN-basic-PF	−31032.67 ± 1241.48	59243.67 ± 786.77	−51911.79 ± 22503.81	−59781.00 ± 19.61
QRNN-rnd-PF	−31767.00 ± 1423.24	**59424.00 ± 531.74**	−51796.42 ± 20282.50	−59800.00 ± 0.00
QRNN-RS	−37135.83 ± 798.06	**59418.67 ± 539.29**	−84800.03 ± 48710.93	−59757.67 ± 25.00
50NN-ASNN-PF	−29807.50 ± 866.15	59216.17 ± 794.06	−45510.71 ± 13709.11	−59800.00 ± 0.00
50NN-basic-PF	−59800.00 ± 0.00	2591.33 ± 140.95	−237153.22 ± 98016.51	−59800.00 ± 0.00
50NN-rnd-PF	−33033.33 ± 1182.28	58768.67 ± 1458.53	−50685.63 ± 24348.51	−59762.33 ± 45.05
50NN-RS	−37649.33 ± 760.09	59235.17 ± 798.79	−79663.60 ± 45101.08	−59790.33 ± 8.18
MSENN-ASNN-PF	−29086.00 ± 1516.73	59120.67 ± 829.56	−68139.01 ± 30516.77	−59357.17 ± 294.15
MSENN-basic-PF	−31076.17 ± 2093.78	59295.00 ± 714.18	−59295.08 ± 25582.97	−59765.00 ± 25.15
MSENN-rnd-PF	−33232.67 ± 1052.73	59091.83 ± 841.48	−59780.33 ± 26732.23	−59778.00 ± 31.11
MSENN-RS	−37765.67 ± 890.60	59104.00 ± 954.75	−121314.94 ± 57540.93	−59728 ± 47.42
QRNN-ASNN-CEM	**-26047.00 ± 599.26**	59216.00 ± 825.90	−42705.73 ± 11515.52	−59800.00 ± 0.00
QRNN-basic-CEM	−26957.67 ± 1256.68	59053.50 ± 1055.71	−46083.07 ± 14078.49	−59671.00 ± 45.99
QRNN-rnd-CEM	−27614.00 ± 573.17	59197.00 ± 852.77	−39717.68 ± 10104.95	−59760.00 ± 56.57
50NN-ASNN-CEM	−27760.50 ± 366.96	59399.83 ± 565.92	−44834.81 ± 11295.70	−59782.00 ± 10.98
50NN-basic-CEM	−59800.00 ± 0.00	3192.00 ± 146.33	−107406.96 ± 28496.87	−59721.33 ± 14.82
50NN-rnd-CEM	−27966.17 ± 48.24	**59543.00± 363.45**	−46379.54 ± 8400.29	−59800.00 ± 0.00
MSENN-ASNN-CEM	**-26731.00 ± 1156.87**	58903.00 ± 838.72	−59763.88 ± 20532.07	−59133.67 ± 244.19
MSENN-basic-CEM	**-26728.33 ± 1303.24**	59156.00 ± 841.36	−59323.06 ± 22940.22	−59066.00 ± 705.91
MSENN-rnd-CEM	−28454.17 ± 180.72	58964.67 ± 1137.78	−57459.30 ± 19774.73	−59780.67 ± 13.77
A2C	−41984.33 ± 7789.17	26731.00 ± 4120.32	−15338.93 ± 6400.97	−59800.00 ± 0.00
PPO	−36909.67 ± 1895.65	40018.17 ± 3158.19	−42493.16 ± 5499.64	−59800.00 ± 0.00
DQN	−51376.33 ± 2239.63	17194.33 ± 218.63	**10366.16 ± 6716.96**	−55685.67 ± 1896.94
QR-DQN	−30390.83 ± 2202.62	36698.83 ± 22378.26	**24594.92 ± 16502.78**	**-47409.17 ± 3416.06**
IV-DQN	−40393.00 ± 297.23	22942.17 ± 1869.13	**14651.49 ± 5757.50**	−57488.50 ± 1268.37

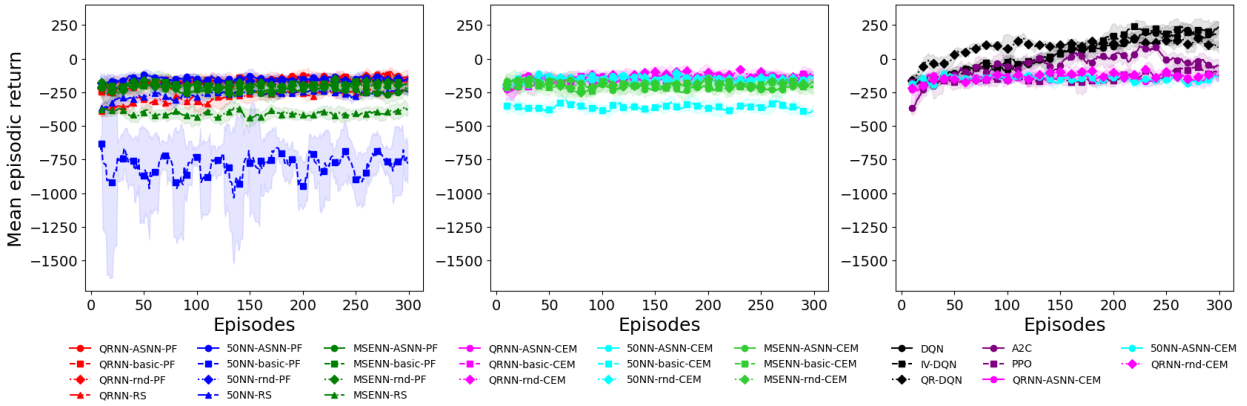
Tables 4.2 and 4.3 show the mean and standard deviation of the AUC of the mean episodic return for the different algorithms applied to the benchmark environments with a continuous action space.



(a) Acrobot

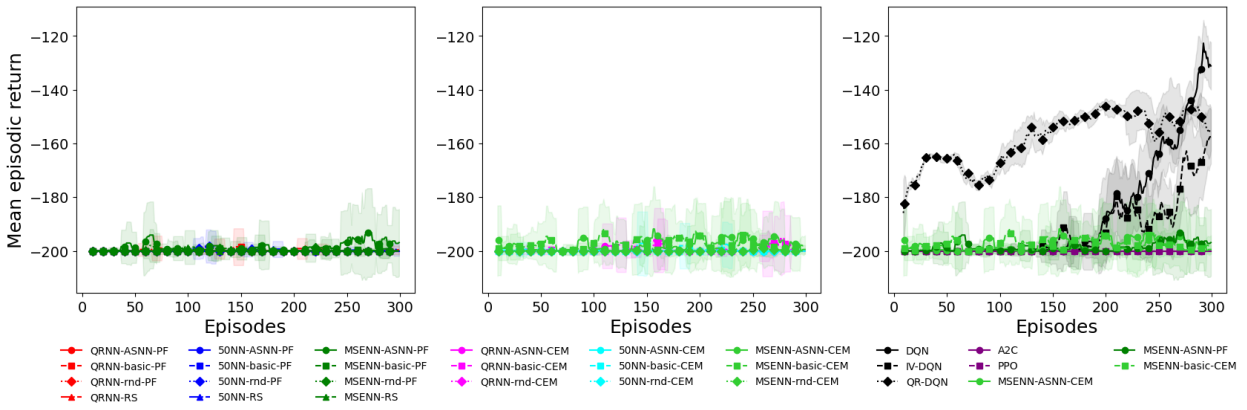


(b) Discrete Cart Pole

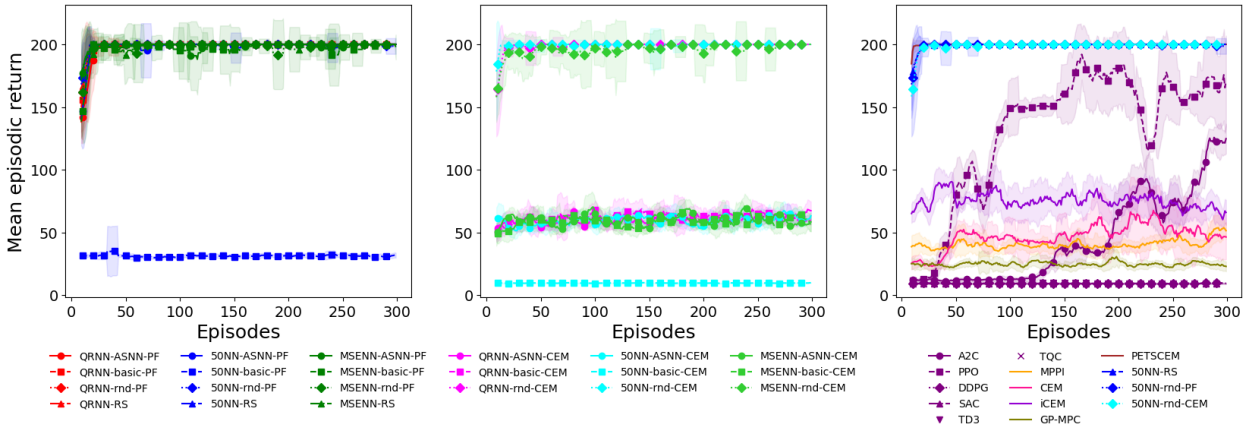


(c) Discrete Lunar Lander

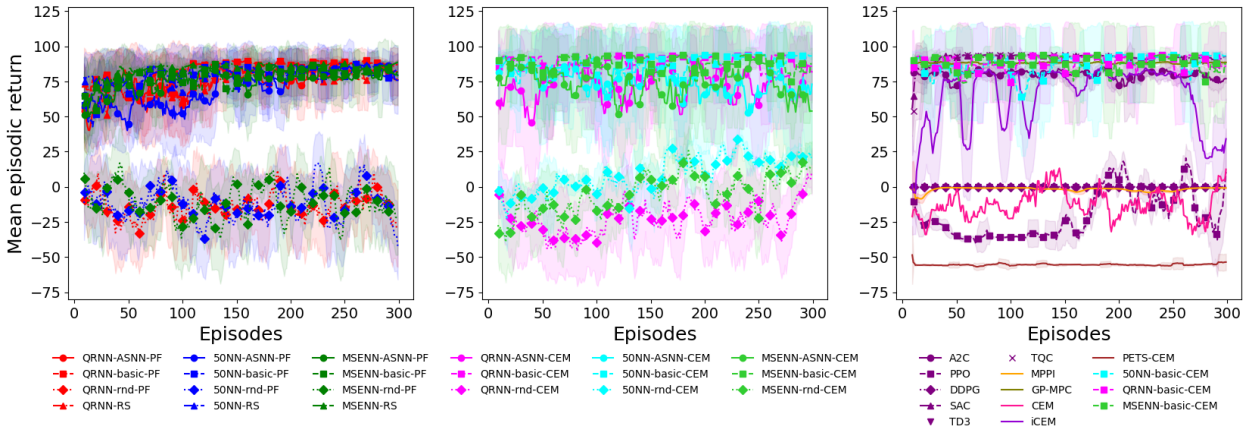
Figure 4.2 Episodic return averaged over seeds 0, 8, and 15 for the Cart Pole, Acrobot, Mountain Car, and Lunar Lander environments.



(a) Discrete Mountain Car

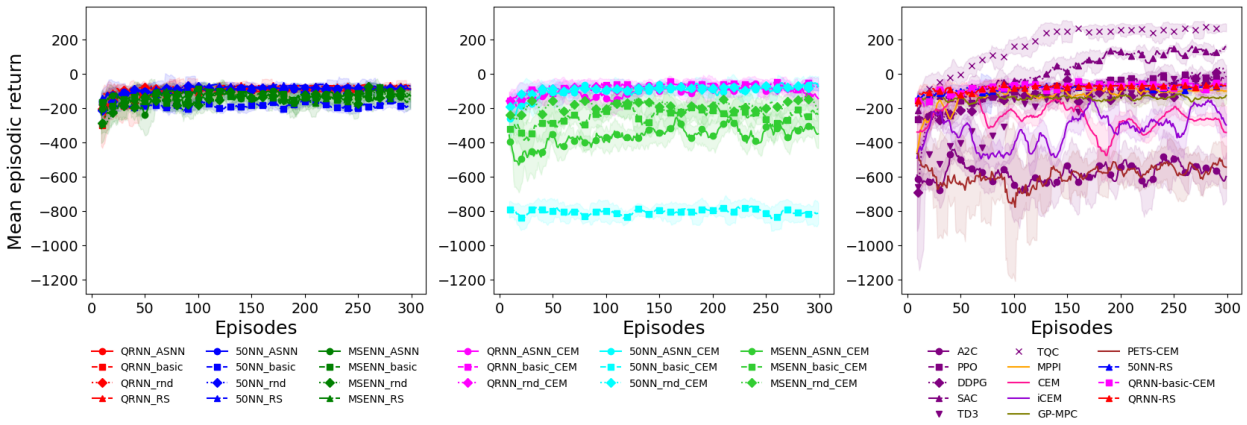


(b) Continuous Cart Pole

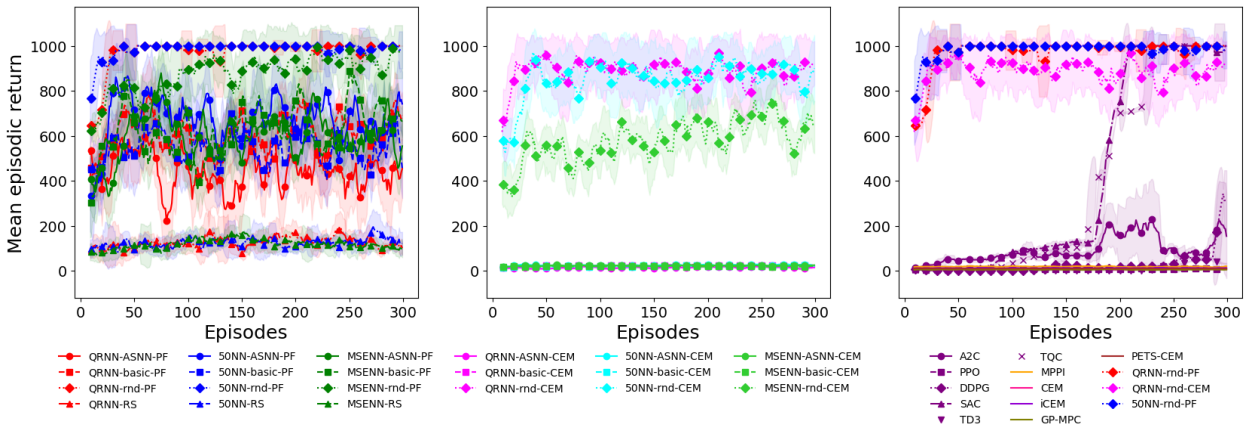


(c) Continuous Mountain car

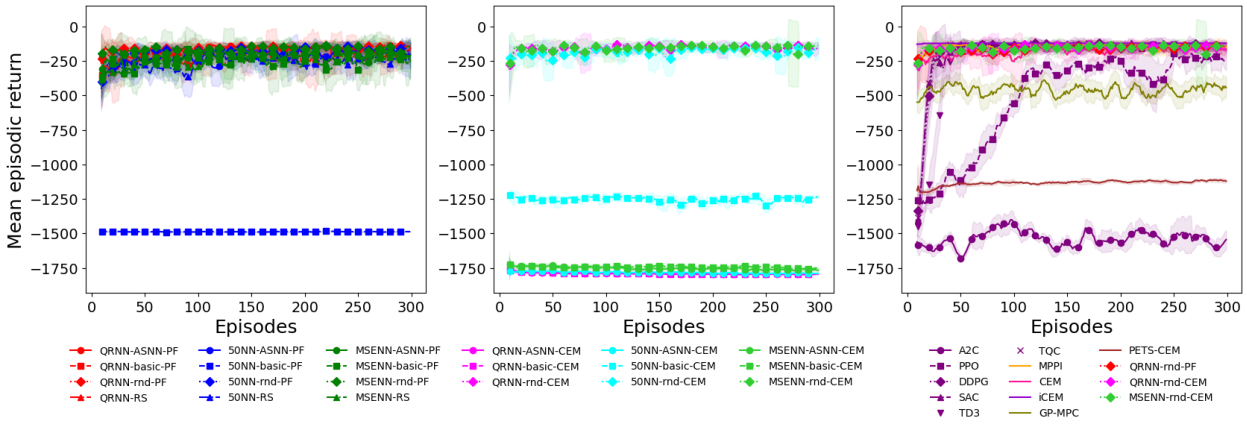
Figure 4.3 Episodic return averaged over seeds 0, 8, and 15 for the Mountain Car, continuous Cart Pole, and continuous Lunar Lander environments.



(a) Continuous Lunar Lander

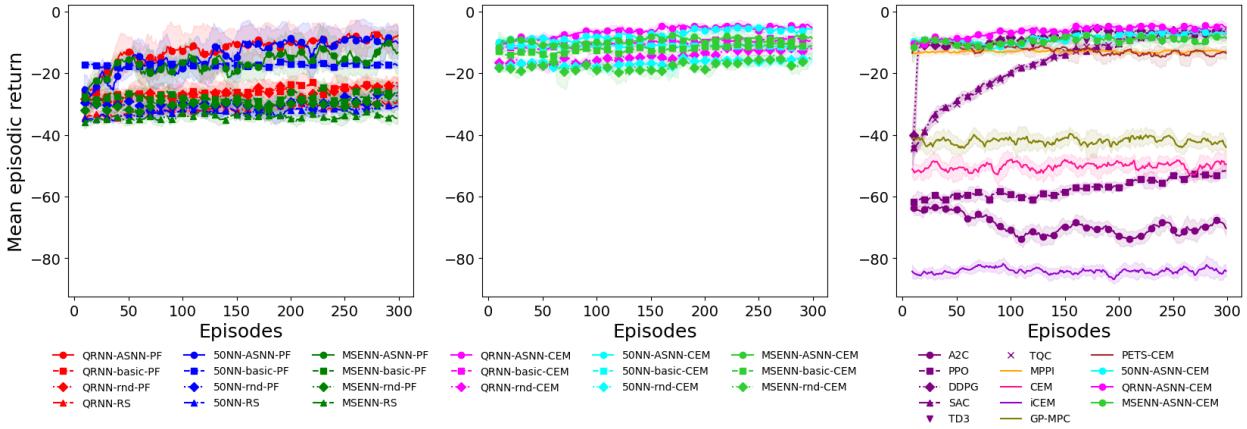


(b) Inverted Pendulum

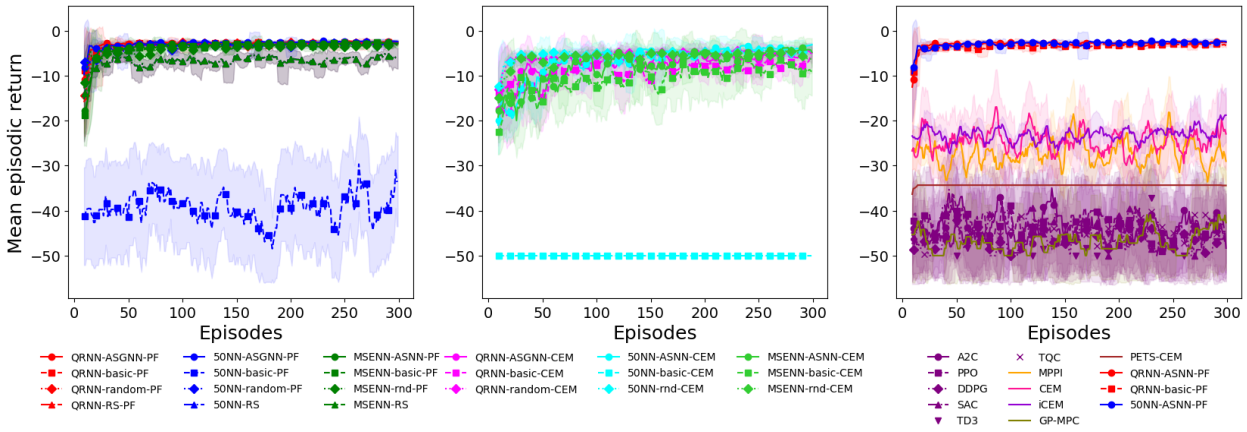


(c) Pendulum

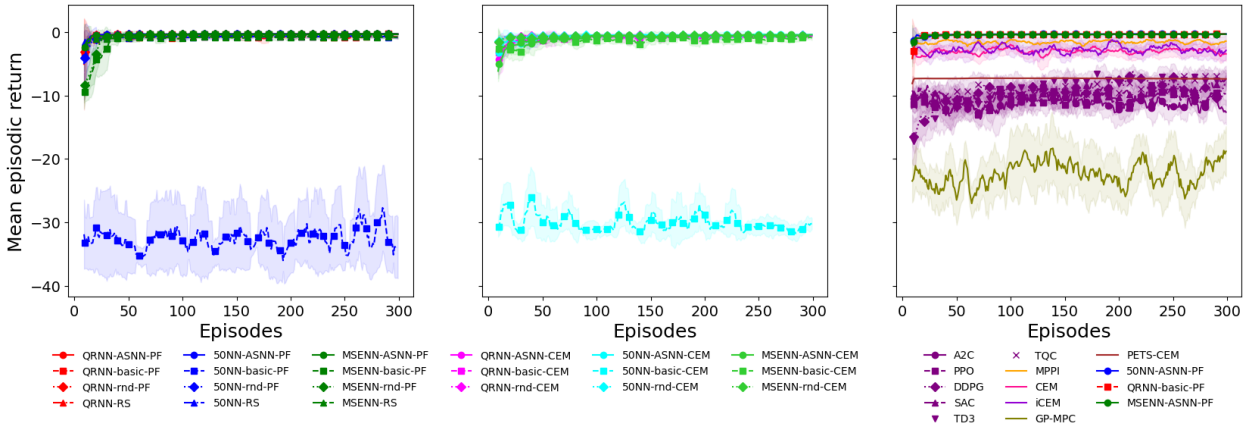
Figure 4.4 Episodic return averaged over seeds 0, 8, and 15 for the continuous Lunar Lander, Pendulum, MuJoCo Reacher environments.



(a) MuJoCo Reacher



(b) Panda Reach sparse rewards



(c) Panda Reach dense rewards

Figure 4.5 Episodic return averaged over seeds 0, 8, and 15 for the MuJoCo Reacher and Panda Reach with sparse and dense reward environments.

Table 4.2 Area under the curve of the episodic return for the continuous action space problems Cart Pole continuous, Mountain Car continuous, Lunar Lander continuous, and Inverted Pendulum

	Cart Pole Continuous	Mountain Car Continuous	Lunar Lander Continuous	Inverted Pendulum
QRNN-ASNN-PF	59045.17 ± 924.33	22510.25 ± 2687.41	−31113.95 ± 1316.16	133146.00 ± 15642.87
QRNN-basic-PF	59271.67 ± 728.87	24463.34 ± 2600.91	−32717.46 ± 7097.44	181218.00 ± 7624.33
QRNN-rnd-PF	59376.33 ± 594.92	−4031.02 ± 3416.00	−28573.99 ± 4056.70	**290150.67 ± 9457.55**
QRNN-RS	59399.33 ± 566.63	22571.00 ± 2022.86	−24697.70 ± 2868.93	37620.83 ± 3684.21
50NN-ASNN-PF	59102.00 ± 745.32	21156.89 ± 5282.43	−28158.63 ± 2417.63	200374.50 ± 6575.47
50NN-basic-PF	9397.00 ± 1679.53	23620.66 ± 2716.22	−55294.50 ± 16318.88	162573.33 ± 7780.51
50NN-rnd-PF	59481.50 ± 331.53	−3668.27 ± 5845.17	−29004.51 ± 3049.95	**293743.17 ± 5455.89**
50NN-RS	**59548.67 ± 301.97**	23475.75 ± 2487.08	−25529.63 ± 4834.29	37455.67 ± 2708.10
MSENN-ASNN-PF	59268.17 ± 416.50	22646.30 ± 1621.73	−40777.65 ± 842.84	178113.00 ± 4146.38
MSENN-basic-PF	58758.33 ± 844.52	23406.81 ± 996.06	−39033.16 ± 1928.96	185951.17 ± 36114.23
MSENN-rnd-PF	59156.33 ± 805.20	−3186.53 ± 2272.49	−45197.36 ± 6558.42	261737.00 ± 29926.80
MSENN-RS	59256.67 ± 607.20	24779.16 ± 377.49	−43005.02 ± 10168.95	35424.50 ± 4711.04
QRNN-ASNN-CEM	18070.67 ± 3188.99	21774.89 ± 73.42	−31760.22 ± 4483.96	4290.33 ± 795.54
QRNN-basic-CEM	18510.83 ± 3068.17	26366.36 ± 973.40	−24053.43 ± 3280.14	6889.83 ± 637.67
QRNN-rnd-CEM	59334.83 ± 624.21	−6563.22 ± 4608.83	−26484.40 ± 4853.13	**265183.83 ± 6636.04**
50NN-ASNN-CEM	17865.33 ± 2542.95	22553.39 ± 3850.98	−30082.03 ± 11660.14	7374.17 ± 663.79
50NN-basic-CEM	2915.00 ± 149.59	25931.50 ± 1544.34	−241166.18 ± 104020.89	6260.83 ± 866.03
50NN-rnd-CEM	**59533.50 ± 246.57**	2700.03 ± 13084.25	−26906.45 ± 3534.27	254636.50 ± 15829.31
MSENN-ASNN-CEM	17909.50 ± 3009.13	22376.78 ± 3499.66	−108650.17 ± 30791.40	6960.67 ± 409.10
MSENN-basic-CEM	17630.67 ± 2727.13	26346.90 ± 809.08	−74216.88 ± 34473.95	5995.83 ± 503.50
MSENN-rnd-CEM	58522.67 ± 1453.37	−1451.56 ± 11856.69	−56682.71 ± 28269.55	172981.17 ± 17415.03
A2C	13240.33 ± 2010.27	23869.03 ± 279.92	−173736.97 ± 59740.15	27405.50 ± 1831.70
PPO	39278.17 ± 11254.01	−5961.02 ± 6138.63	−29289.58 ± 7843.26	2305.50 ± 109.28
DDPG	2816.50 ± 19.30	−4.97 ± 0.53	−43258.09 ± 5443.25	8890.83 ± 9974.04
SAC	2804.83 ± 15.74	**27223.34 ± 168.75**	**5774.24 ± 28151.24**	123590.33 ± 7457.22
TD3	2927.50 ± 180.34	−7.28 ± 0.70	−61906.61 ± 23650.56	3149.00 ± 2200.29
TQC	2808.67 ± 20.83	**27001.77 ± 70.68**	**50448.08 ± 4889.07**	116316.33 ± 19573.50
MPPI	12173.67 ± 3176.01	−531.36 ± 27.06	−39199.82 ± 8640.33	5678.17 ± 963.55
CEM	13979.00 ± 9007.91	−3724.58 ± 1155.38	−76178.52 ± 34314.74	2240.50 ± 497.68
iCEM	22492.17 ± 1382.98	18885.45 ± 2124.18	−97883.40 ± 14614.77	3676.83 ± 886.85
GP-MPC	7440.17 ± 209.10	**26375.91 ± 7.77**	−41935.81 ± 2479.64	1557.33 ± 21.06
PETS-CEM	**59690.33 ± 46.46**	−16522.36 ± 106.63	−176626.20 ± 60431.69	3042.50 ± 276.05

Table 4.3 Area under the curve of the episodic return for the continuous action space problems Pendulum, MuJoCo Reacher, Panda Reach sparse, and Panda Reach dense

	Pendulum	MuJoCo Reacher	Panda Reach sparse	Panda Reach dense
QRNN-ASNN-PF	-50566.89 ± 1893.46	-3768.78 ± 1095.11	**-931.33 ± 280.03**	-149.71 ± 81.78
QRNN-basic-PF	-62423.38 ± 5255.86	-7735.50 ± 1740.14	**-956.33 ± 299.73**	**-141.42 ± 72.11**
QRNN-rnd-PF	-48334.66 ± 2051.91	-7878.76 ± 1144.03	-1045.00 ± 418.72	-144.88 ± 62.97
QRNN-RS	-67757.61 ± 2708.83	-9420.25 ± 392.20	-2067.33 ± 464.32	-254.06 ± 57.74
50NN-ASNN-PF	-68423.97 ± 4940.63	-4209.75 ± 1935.39	**-900.17 ± 273.59**	**-141.23 ± 83.08**
50NN-basic-PF	-444781.26 ± 4779.90	-5183.74 ± 540.23	-11728.83 ± 1599.37	-9736.58 ± 964.17
50NN-rnd-PF	-54983.23 ± 8619.08	-8898.68 ± 1412.97	-1055.50 ± 296.61	-158.73 ± 85.41
50NN-RS	-78347.40 ± 3892.22	-9704.82 ± 624.40	-2067.33 ± 464.32	-240.36 ± 60.94
MSENN-ASNN-PF	-65285.52 ± 1875.90	-5008.59 ± 2236.41	-1170.33 ± 456.83	**-131.02 ± 58.94**
MSENN-basic-PF	-75304.65 ± 13082.64	-8340.40 ± 1325.34	-1173.67 ± 417.02	-281.64 ± 258.12
MSENN-rnd-PF	-49745.17 ± 2413.38	-8996.69 ± 1095.72	-1221.67 ± 326.96	-248.87 ± 187.75
MSENN-RS	-73355.04 ± 2889.18	-10216.22 ± 946.98	-2067.33 ± 464.32	-267.65 ± 68.38
QRNN-ASNN-CEM	-534814.86 ± 9114.03	**-1926.03 ± 336.28**	-2178.67 ± 757.71	-269.83 ± 99.43
QRNN-basic-CEM	-535268.56 ± 9116.50	-3075.66 ± 195.50	-2670.50 ± 928.40	-286.73 ± 117.84
QRNN-rnd-CEM	**-47382.41 ± 2580.26**	-4367.64 ± 934.68	-1653.17 ± 377.96	-188.39 ± 52.07
50NN-ASNN-CEM	-532741.88 ± 10011.77	**-2228.83 ± 497.77**	-2018.33 ± 899.01	-215.54 ± 89.63
50NN-basic-CEM	-374486.07 ± 87569.26	-3343.29 ± 268.90	-14950.00 ± 0.00	-8985.52 ± 1614.02
50NN-rnd-CEM	-55279.43 ± 3055.49	-4978.76 ± 1246.09	-1596.17 ± 376.11	-187.91 ± 55.77
MSENN-ASNN-CEM	-523445.21 ± 12021.27	-2819.56 ± 241.31	-2379.17 ± 947.13	-279.53 ± 134.55
MSENN-basic-CEM	-521272.86 ± 11508.91	-3629.88 ± 293.76	-3170.17 ± 1246.26	-391.97 ± 149.99
MSENN-rnd-CEM	-47665.20 ± 4816.93	-5253.12 ± 1120.96	-1794.00 ± 405.26	-207.02 ± 63.73
A2C	-460152.51 ± 7622.30	-20708.68 ± 2890.80	-13189.00 ± 112.33	-3338.66 ± 197.77
PPO	-157429.34 ± 3942.44	-17156.65 ± 106.14	-13127.00 ± 229.52	-3191.12 ± 115.24
DDPG	-59428.56 ± 1276.20	**-2722.29 ± 91.03**	-13856.50 ± 123.44	-2870.66 ± 305.19
SAC	-59894.24 ± 2428.59	-5031.53 ± 61.38	-13240.00 ± 26.17	-2927.82 ± 59.77
TD3	-75016.92 ± 1414.30	-3133.22 ± 43.77	-13921.00 ± 185.48	-2783.64 ± 125.49
TQC	-59958.40 ± 1136.59	-5096.24 ± 5.07	-13176.67 ± 100.67	-2710.74 ± 34.49
MPPI	**-43317.12 ± 1452.93**	-3871.10 ± 571.40	-8154.17 ± 4243.01	-475.91 ± 310.16
CEM	-54577.26 ± 12386.34	-15022.86 ± 684.61	-7290.67 ± 2951.20	-907.98 ± 545.07
iCEM	**-36832.39 ± 940.64**	-25154.36 ± 497.51	-6900.33 ± 5601.44	-838.75 ± 189.60
GP-MPC	-138375.38 ± 9584.98	-12552.08 ± 163.47	-13971.33 ± 356.84	-6678.02 ± 263.29
PETS-CEM	-339150.29 ± 4772.69	-3715.01 ± 1150.74	-10280.33 ± 6603.91	-2194.12 ± 1496.71

4.4 Normalized mean area under the curve of the episodic return tables

Table 4.4 shows the normalized mean of the area under the curve (AUC) of the mean episodic return for the different algorithms applied to the benchmark environments with a discrete action space.

Tables 4.5 and 4.6 show the normalized mean and standard deviation of the AUC of the mean episodic return for the different algorithms applied to the benchmark environments with a continuous action space.

In both cases, the normalization for the AUCs of a **given problem** was done by comparing algorithm j's average and standard deviation AUC with the best and worst mean AUC of all the methods as described by equation 4.1:

Table 4.4 Normalized area under the curve of the episodic return for discrete action space problems

	Acrobot	Cart Pole	Lunar Lander	Mountain Car
QRNN-ASNN-PF	0.9217 ± 0.0384	0.9847 ± 0.0373	0.7204 ± 0.0655	0.0001 ± 0.0001
QRNN-basic-PF	0.8523 ± 0.0368	0.9814 ± 0.0420	0.7077 ± 0.0860	0.00015 ± 0.0016
QRNN-rnd-PF	0.8305 ± 0.0422	$\mathbf{0.9910 \pm 0.0284}$	0.7081 ± 0.0775	0.0000 ± 0.0000
QRNN-RS	0.6715 ± 0.0236	$\mathbf{0.9908 \pm 0.0288}$	0.5821 ± 0.1861	0.0034 ± 0.0020
50NN-ASNN-PF	0.8886 ± 0.0257	0.9801 ± 0.0425	0.7322 ± 0.0524	0.0000 ± 0.0000
50NN-basic-PF	0.0000 ± 0.0000	0.0000 ± 0.0025	0.0000 ± 0.3745	0.0000 ± 0.0000
50NN-rnd-PF	0.7930 ± 0.0350	0.9563 ± 0.0775	0.7124 ± 0.0930	0.0030 ± 0.0036
50NN-RS	0.6563 ± 0.0225	0.9815 ± 0.0418	0.6017 ± 0.1723	0.0008 ± 0.0007
MSENN-ASNN-PF	0.9100 ± 0.0449	0.9782 ± 0.0466	0.6457 ± 0.1166	0.0357 ± 0.0237
MSENN-basic-PF	0.8510 ± 0.0620	0.9880 ± 0.0328	0.6795 ± 0.0977	0.0028 ± 0.0020
MSENN-rnd-PF	0.7871 ± 0.0312	0.9781 ± 0.0467	0.6776 ± 0.1021	0.0018 ± 0.0025
MSENN-RS	0.6528 ± 0.0264	0.9838 ± 0.0375	0.4426 ± 0.2198	0.0058 ± 0.0038
QRNN-ASNN-CEM	$\mathbf{1.0000 \pm 0.0178}$	0.9799 ± 0.0441	0.7429 ± 0.0440	0.0058 ± 0.0057
QRNN-basic-CEM	0.9730 ± 0.0372	0.9713 ± 0.0564	0.7300 ± 0.0538	0.0104 ± 0.0037
QRNN-rnd-CEM	0.9536 ± 0.0170	0.9789 ± 0.0455	0.7543 ± 0.0386	0.0032 ± 0.0046
50NN-ASNN-CEM	0.9492 ± 0.0109	0.9898 ± 0.0302	0.7347 ± 0.0432	0.0015 ± 0.0009
50NN-basic-CEM	0.00 ± 0.00	0.0107 ± 0.0027	0.4957 ± 0.1089	0.0063 ± 0.0012
50NN-rnd-CEM	0.9431 ± 0.0014	$\mathbf{1.0000 \pm 0.0157}$	0.7288 ± 0.0321	0.0000 ± 0.0000
MSENN-ASNN-CEM	$\mathbf{0.9797 \pm 0.0343}$	0.9771 ± 0.0444	0.6777 ± 0.0784	0.0538 ± 0.0197
MSENN-basic-CEM	$\mathbf{0.9798 \pm 0.0386}$	0.9793 ± 0.0427	0.6794 ± 0.0876	0.0592 ± 0.0570
MSENN-rnd-CEM	0.9287 ± 0.0054	0.9713 ± 0.0557	0.6865 ± 0.0755	0.0016 ± 0.0011
A2C	0.5278 ± 0.2308	0.1654 ± 0.1248	0.8474 ± 0.0245	0.0000 ± 0.0000
PPO	0.6782 ± 0.0562	0.2483 ± 0.0151	0.7437 ± 0.0210	0.0000 ± 0.0000
DQN	0.2496 ± 0.0664	0.0292 ± 0.0033	$\mathbf{0.9565 \pm 0.0460}$	0.3320 ± 0.1531
QR-DQN	0.8713 ± 0.0653	0.5172 ± 0.3418	$\mathbf{1.0000 \pm 0.0630}$	$\mathbf{1.0000 \pm 0.2757}$
IV-DQN	0.5750 ± 0.0088	0.0366 ± 0.0037	$\mathbf{0.9620 \pm 0.0220}$	0.1865 ± 0.1024

$$\text{AUC}_{\text{norm},j} = \mu_{\text{norm AUC},j} \pm \sigma_{\text{norm AUC},j} \tag{4.1}$$

where $\mu_{\text{norm AUC},j}$ and $\sigma_{\text{norm AUC},j}$ are described by Equations 4.2 and 4.3:

$$\mu_{\text{norm AUC},j} = \frac{\mu_{\text{AUC}_j} - \max(\mu_{AUC})}{\max(\mu_{AUC}) - \min(\mu_{AUC})} \tag{4.2}$$

$$\sigma_{\text{norm AUC},j} = \frac{\sigma_{\text{AUC}_j}}{\max(\mu_{AUC}) - \min(\mu_{AUC})} \tag{4.3}$$

where AUC_{norm} is the normalized AUC, μ_{AUC} contains the mean AUC of all the algorithms for a given problem, $\mu_{AUC,j}$ is the mean AUC of an algorithm, and $\sigma_{AUC,j}$ is the standard deviation AUC of an algorithm.

Table 4.5 Normalized area under the curve of the episodic return for the continuous action space problems Cart Pole Continuous, Mountain Car Continuous, Lunar Lander Continuous, and Inverted Pendulum

	Cart Pole Continuous	Mountain Car Continuous	Lunar Lander Continuous	Inverted Pendulum
QRNN-ASNN-PF	0.9887 ± 0.0162	0.8923 ± 0.0614	0.7203 ± 0.0045	0.4504 ± 0.0535
QRNN-basic-PF	0.9926 ± 0.0128	0.9369 ± 0.0595	0.7148 ± 0.0243	0.6149 ± 0.0261
QRNN-rnd-PF	0.9945 ± 0.0105	0.2855 ± 0.0781	0.7290 ± 0.0139	$\mathbf{0.9877 \pm 0.0324}$
QRNN-RS	0.9949 ± 0.0100	0.8937 ± 0.0462	0.7423 ± 0.0098	0.1234 ± 0.0126
50NN-ASNN-PF	0.9897 ± 0.0131	0.8613 ± 0.1208	0.7304 ± 0.0083	0.6804 ± 0.0225
50NN-basic-PF	0.1159 ± 0.0295	0.9176 ± 0.0621	0.6374 ± 0.0560	0.5511 ± 0.0266
50NN-rnd-PF	0.9963 ± 0.0058	0.2938 ± 0.1336	0.7275 ± 0.0105	$\mathbf{1.0000 \pm 0.0187}$
50NN-RS	$\mathbf{0.9975 \pm 0.0053}$	0.9143 ± 0.0569	0.7395 ± 0.0166	0.1229 ± 0.0093
MSENN-ASNN-PF	0.9926 ± 0.0073	0.8954 ± 0.0371	0.6872 ± 0.0029	0.6043 ± 0.0142
MSENN-basic-PF	0.9836 ± 0.0148	0.9128 ± 0.0228	0.6932 ± 0.0066	0.6311 ± 0.1236
MSENN-rnd-PF	0.9906 ± 0.0142	0.3048 ± 0.0519	0.6720 ± 0.0225	0.8905 ± 0.1024
MSENN-RS	0.9924 ± 0.0107	0.9441 ± 0.0086	0.6795 ± 0.0349	0.1159 ± 0.0161
QRNN-ASNN-CEM	0.2684 ± 0.0561	0.8755 ± 0.0017	0.7181 ± 0.0154	0.0094 ± 0.0027
QRNN-basic-CEM	0.2761 ± 0.0539	0.9804 ± 0.0223	0.7445 ± 0.0112	0.0183 ± 0.0022
QRNN-rnd-CEM	0.9938 ± 0.0110	0.2277 ± 0.1054	0.7362 ± 0.0166	$\mathbf{0.9023 \pm 0.0227}$
50NN-ASNN-CEM	0.2648 ± 0.0447	0.8932 ± 0.0880	0.7238 ± 0.0400	0.0199 ± 0.0023
50NN-basic-CEM	0.0019 ± 0.0026	0.9705 ± 0.0353	0.0000 ± 0.3567	0.0161 ± 0.0030
50NN-rnd-CEM	$\mathbf{0.9972 \pm 0.0043}$	0.4394 ± 0.2991	0.7347 ± 0.0121	0.8662 ± 0.0542
MSENN-ASNN-CEM	0.2655 ± 0.0529	0.8892 ± 0.0800	0.4544 ± 0.1056	0.0185 ± 0.0014
MSENN-basic-CEM	0.2606 ± 0.0479	0.9800 ± 0.0185	0.5725 ± 0.1182	0.0152 ± 0.0017
MSENN-rnd-CEM	0.9795 ± 0.0255	0.3445 ± 0.2710	0.6326 ± 0.0969	0.5867 ± 0.0596
A2C	0.1834 ± 0.0353	0.9233 ± 0.0064	0.2312 ± 0.2049	0.0885 ± 0.0063
PPO	0.6412 ± 0.1978	0.2414 ± 0.1403	0.7266 ± 0.0269	0.0026 ± 0.0004
DDPG	0.0002 ± 0.0003	0.3776 ± 0.0000	0.6787 ± 0.0187	0.0251 ± 0.0341
SAC	0.0000 ± 0.0003	$\mathbf{1.0000 \pm 0.0039}$	$\mathbf{0.8468 \pm 0.0965}$	0.4177 ± 0.0255
TD3	0.0022 ± 0.0032	0.3775 ± 0.0000	0.6147 ± 0.0811	0.0054 ± 0.0075
TQC	0.0001 ± 0.0004	$\mathbf{0.9949 \pm 0.0016}$	$\mathbf{1.0000 \pm 0.0168}$	0.3928 ± 0.0670
MPPI	0.1647 ± 0.0558	0.3655 ± 0.0006	0.6926 ± 0.0296	0.0141 ± 0.0033
CEM	0.1964 ± 0.1584	0.2925 ± 0.0264	0.5658 ± 0.1177	0.0023 ± 0.0017
iCEM	0.3461 ± 0.0243	0.8094 ± 0.0486	0.4913 ± 0.0501	0.0073 ± 0.0030
GP-MPC	0.0815 ± 0.0037	$\mathbf{0.9806 \pm 0.0002}$	0.6832 ± 0.0085	0.0000 ± 0.0001
PETS-CEM	$\mathbf{1.0000 \pm 0.0008}$	0.0000 ± 0.0024	0.2213 ± 0.2072	0.0051 ± 0.0009

Table 4.6 Normalized area under the curve of the episodic return for the continuous action space problems Pendulum, MuJoCo Reacher, Panda Reach Sparse, and Panda Reach Dense

	Pendulum	MuJoCo Reacher	Panda Reach Sparse	Panda Reach Dense
QRNN-ASNN-PF	0.9724 ± 0.0038	0.9207 ± 0.0471	$\mathbf{0.9978 \pm 0.0199}$	0.9981 ± 0.0085
QRNN-basic-PF	0.9487 ± 0.0105	0.7499 ± 0.0749	$\mathbf{0.9960 \pm 0.0213}$	$\mathbf{0.9989 \pm 0.0075}$
QRNN-rnd-PF	0.9769 ± 0.0041	0.7437 ± 0.0493	0.9897 ± 0.0298	0.9986 ± 0.0066
QRNN-RS	0.9380 ± 0.0054	0.6774 ± 0.0169	0.9169 ± 0.0330	0.9872 ± 0.0060
50NN-ASNN-PF	0.9366 ± 0.0099	0.9017 ± 0.0833	$\mathbf{1.0000 \pm 0.0195}$	$\mathbf{0.9989 \pm 0.0086}$
50NN-basic-PF	0.1815 ± 0.0096	0.8598 ± 0.0233	0.2293 ± 0.1138	0.0000 ± 0.1004
50NN-rnd-PF	0.9636 ± 0.0173	0.6998 ± 0.0608	0.9889 ± 0.0211	0.9971 ± 0.0089
50NN-RS	0.9167 ± 0.0078	0.6651 ± 0.0269	0.9169 ± 0.0330	0.9886 ± 0.0063
MSENN-ASNN-PF	0.9429 ± 0.0038	0.8673 ± 0.0963	0.9808 ± 0.0325	$\mathbf{1.0000 \pm 0.0061}$
MSENN-basic-PF	0.9228 ± 0.0262	0.7239 ± 0.0571	0.9805 ± 0.0297	0.9843 ± 0.0269
MSENN-rnd-PF	0.9741 ± 0.0048	0.6956 ± 0.0472	0.9771 ± 0.0233	0.9877 ± 0.0195
MSENN-RS	0.9267 ± 0.0058	0.6431 ± 0.0408	0.9169 ± 0.0330	0.9858 ± 0.0071
QRNN-ASNN-CEM	0.0009 ± 0.0183	$\mathbf{1.0000 \pm 0.0145}$	0.9090 ± 0.0539	0.9855 ± 0.0104
QRNN-basic-CEM	0.0000 ± 0.0183	0.9505 ± 0.0084	0.8740 ± 0.0661	0.9838 ± 0.0123
QRNN-rnd-CEM	$\mathbf{0.9788 \pm 0.0052}$	0.8949 ± 0.0402	0.9464 ± 0.0269	0.9940 ± 0.0054
50NN-ASNN-CEM	0.0051 ± 0.0201	$\mathbf{0.9870 \pm 0.0214}$	0.9204 ± 0.0640	0.9912 ± 0.0093
50NN-basic-CEM	0.3226 ± 0.1757	0.9390 ± 0.0116	0.0000 ± 0.0000	0.0782 ± 0.1680
50NN-rnd-CEM	0.9630 ± 0.0061	0.8686 ± 0.0536	0.9505 ± 0.0268	0.9941 ± 0.0058
MSENN-ASNN-CEM	0.0237 ± 0.0241	0.9615 ± 0.0104	0.8947 ± 0.0674	0.9845 ± 0.0140
MSENN-basic-CEM	0.0281 ± 0.0231	0.9266 ± 0.0126	0.8384 ± 0.0887	0.9728 ± 0.0156
MSENN-rnd-CEM	0.9783 ± 0.0097	0.8568 ± 0.0483	0.9364 ± 0.0288	0.9921 ± 0.0066
A2C	0.1507 ± 0.0153	0.1914 ± 0.1245	0.1253 ± 0.0080	0.6661 ± 0.0206
PPO	0.7580 ± 0.0079	0.3443 ± 0.0046	0.1298 ± 0.0163	0.6814 ± 0.0120
DDPG	0.9547 ± 0.0026	$\mathbf{0.9657 \pm 0.0039}$	0.0778 ± 0.0088	0.7148 ± 0.0318
SAC	0.9537 ± 0.0049	0.8663 ± 0.0026	0.1217 ± 0.0019	0.7088 ± 0.0062
TD3	0.9234 ± 0.0028	0.9480 ± 0.0019	0.0732 ± 0.0132	0.7238 ± 0.0131
TQC	0.9536 ± 0.0023	0.8635 ± 0.0002	0.1262 ± 0.0072	0.7314 ± 0.0036
MPPI	$\mathbf{0.9870 \pm 0.0029}$	0.9163 ± 0.0246	0.4837 ± 0.3020	0.9641 ± 0.0323
CEM	0.9644 ± 0.0249	0.4362 ± 0.0295	0.5452 ± 0.2101	0.9191 ± 0.0567
iCEM	$\mathbf{1.0000 \pm 0.0019}$	0.0000 ± 0.0214	0.5729 ± 0.3987	0.9263 ± 0.0197
GP-MPC	0.7963 ± 0.0192	0.5425 ± 0.0070	0.0697 ± 0.0254	0.3184 ± 0.0274
PETS-CEM	0.3935 ± 0.0096	0.9230 ± 0.0495	0.3324 ± 0.4700	0.7852 ± 0.1558

4.5 Overall algorithm comparison

To obtain an overall comparison of all the algorithms tested on the benchmark problems, the average normalized mean area under the curve is computed by summing the normalized mean AUCs for a given algorithm across all discrete and continuous action space problems. The root-mean squared (rms) normalized normalized AUC std is also computed. To get an algorithm's average normalized AUC over the N_{problems} different problems with its standard deviation, Equation 4.4 is used:

$$\overline{\text{AUC}_{\text{norm},j}} = \left(\frac{1}{N_{\text{problems}}} \sum_{i=0}^{N_{\text{problems}}} \mu_{\text{norm AUC},j} \right) \pm \sqrt{\frac{\sum_{j=0}^{N_{\text{problems}}} \sigma^2_{\text{norm AUC},j}}{N_{\text{problems}}}} \tag{4.4}$$

Tables 4.7 and 4.8 provide these values in the discrete and continuous action space cases, respectively.

Table 4.7: Average normalized area under the curve of the episodic return for discrete space environments in decreasing order

Model	Average normalized AUC
QR-DQN	0.8675 ± 0.2443
QRNN-ASNN-CEM	0.6857 ± 0.0250
MSENN-basic-CEM	0.6779 ± 0.0562
QRNN-rnd-CEM	0.6763 ± 0.0225
QRNN-basic-CEM	0.6762 ± 0.0340
MSENN-ASNN-CEM	0.6750 ± 0.0445
50NN-ASNN-CEM	0.6707 ± 0.0228
50NN-rnd-CEM	0.6680 ± 0.0164
QRNN-ASNN-PF	0.6595 ± 0.0385
50NN-ASNN-PF	0.6538 ± 0.0300
MSENN-rnd-CEM	0.6517 ± 0.0392
MSENN-ASNN-PF	0.6460 ± 0.0640
QRNN-basic-PF	0.6391 ± 0.0473
QRNN-rnd-PF	0.6341 ± 0.0444
MSENN-basic-PF	0.6322 ± 0.0582
50NN-rnd-PF	0.6237 ± 0.0514
MSENN-rnd-PF	0.6147 ± 0.0539
QRNN-RS	0.5637 ± 0.0939
50NN-RS	0.5633 ± 0.0872
IV-DQN	0.5202 ± 0.0550
PPO	0.5198 ± 0.0408
MSENN-RS	0.5234 ± 0.1110
A2C	0.4498 ± 0.1215
DQN	0.4459 ± 0.0844
50NN-basic-CEM	0.1281 ± 0.0545
50NN-basic-PF	0.0000 ± 0.1872

Table 4.8: Average normalized area under the curve of the episodic return for continuous space environments

Model	Average normalized AUC
50NN-ASNN-PF	0.8874 ± 0.0534
MSENN-ASNN-PF	0.8713 ± 0.0388
QRNN-basic-PF	0.8691 ± 0.0374
QRNN-ASNN-PF	0.8676 ± 0.0347
MSENN-basic-PF	0.8540 ± 0.0520
50NN-rnd-CEM	0.8517 ± 0.1097
QRNN-rnd-PF	0.8382 ± 0.0368
QRNN-rnd-CEM	0.8343 ± 0.0424
50NN-rnd-PF	0.8334 ± 0.0535
MSENN-rnd-PF	0.8116 ± 0.0462
MSENN-rnd-CEM	0.7884 ± 0.1063
QRNN-RS	0.7842 ± 0.0222
50NN-RS	0.7827 ± 0.0263
MSENN-RS	0.7756 ± 0.0237
TQC	0.6328 ± 0.0246
SAC	0.6144 ± 0.0355
QRNN-basic-CEM	0.6034 ± 0.0325
50NN-ASNN-CEM	0.6007 ± 0.0453
QRNN-ASNN-CEM	0.5958 ± 0.0295
MSENN-basic-CEM	0.5743 ± 0.0564
MPPI	0.5735 ± 0.1100
MSENN-ASNN-CEM	0.5615 ± 0.0568
iCEM	0.5192 ± 0.1437
CEM	0.4902 ± 0.1051
DDPG	0.4743 ± 0.0181
TD3	0.4585 ± 0.0296
PETS-CEM	0.4576 ± 0.1906
PPO	0.4407 ± 0.0866
50NN-basic-PF	0.4366 ± 0.0635
GP-MPC	0.4340 ± 0.0154
A2C	0.3200 ± 0.0862
50NN-basic-CEM	0.2910 ± 0.1532

4.6 Interpretation of the Results on the OpenAI Gymnasium and Panda Gym environments

Acrobot

Figure 4.2a shows that the MPC methods are overall better. This is true except for the 50NN-basic-PF and 50NN-basic-CEM methods. The left figure allows us to see that RS is worse than the other MPC methods. This indicates that the use of PF or CEM to optimize action sequences in MPC has a positive effect. Overall, the MPC-CEM are better than the MPC-PF ones.

Among the RL methods, QR-DQN performs best initially, and IV-DQN achieves the best convergence. The latter one attains a similar convergence value to the best MPC-methods, which are QRNN-ASNN-CEM, 50NN-ASNN-CEM, and MSENN-basic-CEM, as the area under the curve (AUC) tables 4.1 and 4.4 show. The tables also clearly show that MPC methods using CEM are generally superior to those using PF.

Discrete Cart Pole

Figure 4.2b shows that the MPC-PF and MPC-CEM methods give slight variations early on but can quickly converge and solve the task. This is true, except for the 50-basic-PF and 50NN-basic-CEM, which are similar to the results for the Acrobot environment. Overall, the MFRL methods yield poor results, although QR-DQN is the best among them. These performances are based on the fact that we only ran the algorithms on 100 episodes, which is quite minimal.

Tables 4.1 and 4.4 show that the top three methods are QRNN-rnd-PF, 50NN-rnd-CEM, and QRNN-RS. To solve the task, the agent must attain an episodic reward of 200. Most MPC methods have achieved this goal, whereas none of the RL methods have. Overall, there doesn't seem to be a significant difference between using PF or CEM.

Discrete Lunar lander

Figure 4.2c shows that the MPC-PF and MPC-CEM methods give similar results. They are also generally better than RS. This is true, except for 50NN-basic-PF and 50NN-basic-CEM, which are the worst of the MPC methods, and 50NN-basic-CEM, which performs slightly better than MSENN-RS.

Tables 4.1 and 4.4 show that the best MPC methods are QRNN-ASNN-CEM, QRNN-rnd-CEM, and 50NN-ASNN-CEM. These methods exhibit similar behaviour to the PPO, which

is considered the worst of the RL algorithms. The best algorithms overall are three DQN methods. They achieve much better results than the best MPC methods. To solve the task, the agent must attain an episodic reward of 200. None of the methods have achieved this goal, but the DQN methods are the closest and have been able to solve the task properly sometimes. Overall, the MPC-CEM methods are slightly better than the MPC-PF algorithms.

Discrete Mountain Car

Figure 4.3a shows that A2C, PPO and the MPC methods aren't able to solve the task successfully. This is clear since the agents constantly receive an episodic reward of -200, which is the result of the -1 reward for each time step. The DQN methods have more success, with QR-DQN performing the best initially and DQN performing better later on. Surprisingly, IV-DQN does very poorly, but it is better than the MPC methods.

Tables 4.1 and 4.4 support the conclusions from the graphs that the DQN and QR-DQN are the two best methods, and the rest are never able to solve the task or do so significantly less than them. In this case, there isn't a difference between the MPC-PF and MPC-CEM algorithms.

Continous Cart Pole

Figure 4.3b shows that most of the MPC-PF algorithms and the different MPC-rnd-CEM methods are the best overall and converge quickly, solving the task. Once again, the two 50NN-basic methods are the worst MPC methods. The non-rnd MPC-CEM methods behave much worse compared to their PF equivalents. The different MFRL, trajectory optimization and benchmark MBRL algorithms yield poor performance. Though, A2C and PPO are the best of them.

Tables 4.2 and 4.5 show that the top three methods are 50NN-RS, 50NN-rnd-PF, and 50NN-rnd-CEM. Overall, there's a slight advantage in using PF over CEM in an MPC algorithm.

Continuous Mountain Car

Figure 4.3c shows that the three basic-CEM methods (50NN-basic-CEM, QRNN-basic-CEM, MSENN-basic-CEM) outperform the other MPC methods. The rnd-PF and rnd-CEM methods perform the worst of our MPC methods. Overall, using PF in MPC seems to be better than CEM, but the top MPC methods use CEM. Among the trajectory optimization methods, iCEM performs the best, but it still struggles to achieve performance comparable to the

top algorithms. For the RL methods, SAC, TQC, and A2C are the top ones. As tables 4.2 and 4.5, the top three algorithms are TQC, SAC, and GP-MPC. As mentioned earlier, the MPC-PF methods overall outperform the MPC-CEM ones. PETS-CEM is the worst of all. For the continuous Mountain Car environment, there is no defined expected episodic return to state that the task has been learned confidently. However, a larger reward is better since it indicates that the agent attained the goal in a few steps and received the +100 bonus for reaching the goal state.

Continuous Lunar Lander

Figure 4.4a shows that the spread in the performance in the MPC-PF methods is slimmer than for the MPC-CEM algorithms.

Tables 4.2 and 4.6 show that the top three algorithms are TQC, SAC, and QRNN-basic-CEM. Overall, the MPC-PF methods outperform the MPC-CEM ones. As for the discrete Lunar Lander environment, the task is considered learned by an algorithm when the episodic return exceeds 200. TQC is the only method that attains this goal.

Inverted Pendulum

Figure 4.4b shows that the random versions of the MPC-PF and MPC-CEM methods are the best. The top MFRL methods are TQC and SAC. To successfully solve the task, the agent must achieve a return of 1000 as an episodic reward. 50NN-rnd-PF, QRNN-rnd-PF, TQC, and SAC are the four methods that attain this goal at convergence. 50NN-rnd-PF and QRNN-rnd-PF converge significantly faster than TQC and SAC. The other algorithms perform poorly and do not converge to an episodic return of 1000. The poor performance of the MPC-ASNN methods is surprising, considering they performed well on similar control tasks, such as the continuous Cart Pole and Pendulum. This may be caused by the fact that the environment's state variables are unbounded, which leads to a potentially huge discrepancy in orders of magnitude between the different state components. This can significantly affect the ASNN's predictions.

Tables 4.2 and 4.5 show that the top three algorithms are 50NN-rnd-PF, QRNN-rnd-PF, and QRNN-rnd-CEM. Overall, the MPC-PF algorithms outperform the MPC-CEM ones.

Pendulum

Figure 4.4c shows that all MPC methods using PF give similar performances except for 50NN-basic-PF. For the MPC methods using CEM, most aren't effective except for the

three rnd-CEM methods: QRNN-rnd-CEM, 50NN-rnd-CEM, and MSENN-rnd-CEM. The trajectory optimization methods iCEM and MPPI perform extremely well and converge virtually instantly. The best MFRL method is DDPG, which starts poorly early on but quickly attains similar performance to the best MPC and trajectory optimization algorithms. SAC and TQC do slightly worse.

Tables 4.3 and 4.6 show that the best methods are iCEM, MPPI, and QRNN-rnd-CEM. Overall, the PF methods are much better than the CEM ones.

MuJoCo Reacher

Figure 4.5a shows two groupings in the results for the MPC-PF. The ASNN methods have a clear advantage compared to the others. This distinction remains present for the MPC-CEM algorithms, but it is less pronounced since the non-ASNN methods perform better. For the MFRL, MBRL, and control methods, DDPG and TD3 converge the quickest.

Tables 4.3 and 4.6 show that the top three methods are QRNN-ASNN-CEM, 50NN-ASNN-CEM, and DDPG. Overall, there is a clear advantage in using CEM over PF for the MuJoCo reacher environment. Additionally, the ASNN is a valuable addition to the MPC methods.

Panda 3D sparse reward Reach

Figure 4.5b shows a clear distinction again between RS and the other MPC-PF methods. The latter all give similar performance, except 50NN-basic-PF. This separation isn't present for the MPC-CEM methods, except for 50NN-basic-CEM, which is once again inferior to the rest. The control methods all do very poorly. The benchmark MBRL and RL methods do even worse. These results are expected and logical, given the sparse rewards. This means that the agents do not receive any useful rewards unless they happen to randomly attain the goal. The agent must perform this task sufficiently often to learn an effective policy, which is very challenging in only 300 episodes. The overall better performance of the MPC methods demonstrates their strength, as they do not depend on the given problem's reward structure.

Tables 4.3 and 4.6 show that the best MPC methods are QRNN-ASNN-PF, QRNN-basic-PF, and 50NN-ASNN-PF. Using PF in MPC is generally better than CEM for the sparse reward version of the Panda Reach environment.

Panda 3D dense reward Reach

Figure 4.5c shows that there is very little difference between the MPC-PF and MPC-CEM algorithms, though the PF ones are slightly better. The two 50NN-basic methods are once again much worse than the rest. The trajectory optimization methods outperform the MBRL and MFRL methods. The MFRL, trajectory optimization, and MBRL algorithms still perform significantly worse than the MPC methods; however, their performance has improved substantially compared to that obtained for the sparse reward version of the Panda Reach environment. This highlights the crucial role of reward distribution in algorithms that learn based on the rewards.

Tables 4.3 and 4.6 show that the best MPC methods are 50NN-ASNN-PF, QRNN-basic-PF, and MSENN-ASNN-PF. Once again, using PF has a clear advantage over CEM in this environment. Adding an ASNN to an MPC algorithm also appears to improve performance generally.

4.7 Recap of results

The table 4.9 recaps the key trends in the results and analysis presented above. The colours used here are to mimic the ones used in the graphs.

Table 4.9 Recap of Results Across Environments

Environment	**PF/CEM**	**Top 3 Algorithms**
Acrobot	CEM	1. QRNN-ASNN-CEM, 2. MSENN-basic-CEM, 3. MSENN-ASNN-CEM
Discrete Cart Pole	Equivalent	1. 50NN-rnd-CEM, 2. QRNN-rnd-PF, 3. QRNN-RS
Discrete Lunar Lander	CEM	1. QR-DQN, 2. IV-DQN, 3. DQN
Discrete Mountain Car	PF	1. QR-DQN, 2. DQN, 3. IV-DQN
Continuous Cart Pole	PF	1. PETS-CEM, 2. 50NN-RS, 3. 50NN-rnd-CEM
Inverted Pendulum	PF	1. 50NN-rnd-PF, 2. QRNN-rnd-PF, 3. QRNN-rnd-CEM
Continuous Mountain Car	PF	1. SAC, 2. TQC, 3. GP-MPC
Continuous Lunar Lander	CEM	1. TQC, 2. SAC, 3. QRNN-basic-CEM
Pendulum	PF	1. iCEM, 2. MPPI, 3. QRNN-rnd-CEM
MuJoCo Reacher	CEM	1. QRNN-ASNN-CEM, 2. 50NN-ASNN-CEM, 3. DDPG
Panda Reach (sparse)	PF	1. 50NN-ASNN-PF, 2. QRNN-ASNN-PF, 3. QRNN-basic-PF
Panda Reach (dense)	PF	1. MSENN-ASNN-PF, 2. 50NN-ASNN-PF, 3. QRNN-basic-PF

4.8 Other tests

4.8.1 Sampling method for QRNN next state prediction

The Figure 4.6 compares the results for the QRNN algorithms using the mid quantile and the sampling method described in 3.3.4 for next state prediction:

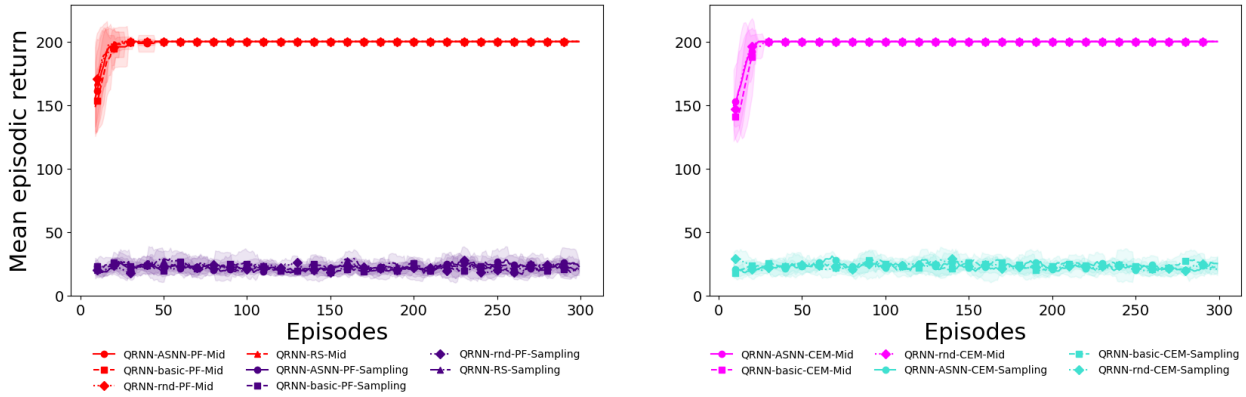


Figure 4.6 Visualization of QRNN-ASNN

Table 4.10 Area under the curve of the episodic return on the discrete Cart Pole environment for our MPC methods using the mid quantile or sampling the quantiles for next state prediction

	Using the mid quantile	Sampling the generated quantiles
QRNN-ASNN-PF	59305.33 ± 699.56	6682.83 ± 153.14
QRNN-basic-PF	59243.67 ± 786.77	6607.67 ± 128.83
QRNN-rnd-PF	$\mathbf{59424.00 \pm 531.74}$	6615.33 ± 216.13
QRNN-RS	$\mathbf{59418.67 \pm 539.29}$	6822.00 ± 177.29
QRNN-ASNN-CEM	59216.00 ± 825.90	6909.67 ± 104.44
QRNN-basic-CEM	59053.50 ± 1055.71	7051.00 ± 241.24
QRNN-rnd-CEM	59197.00 ± 852.77	6890.67 ± 207.93

Figure 4.6 and Table 4.10 clearly show that for next state prediction using the QRNN, it is much better to use the mid quantile than sampling the quantiles using the described method.

4.8.2 Compare repeating 4 times the optimized action to not doing so for the Mountain Car environments

As described above in section 3.2.5, the GP-MPC method repeats the action found for four steps in the environment and this idea was applied to our MPC methods, CEM, iCEM and MPPI. We now compare the performance of these methods when repeating the action four

times and only using the action found for a single step in the environment. This comparison was done for the continuous and discrete versions of the Mountain Car environment. Figures 4.7 and 4.8 show the episodic rewards of our MPC methods and the trajectory optimization algorithms when taking the action found for a single step and repeating it for four steps in the environment. Table 4.11 presents the area under the curve of the episodic return for the continuous and discrete versions of the Mountain Car environment in both cases.

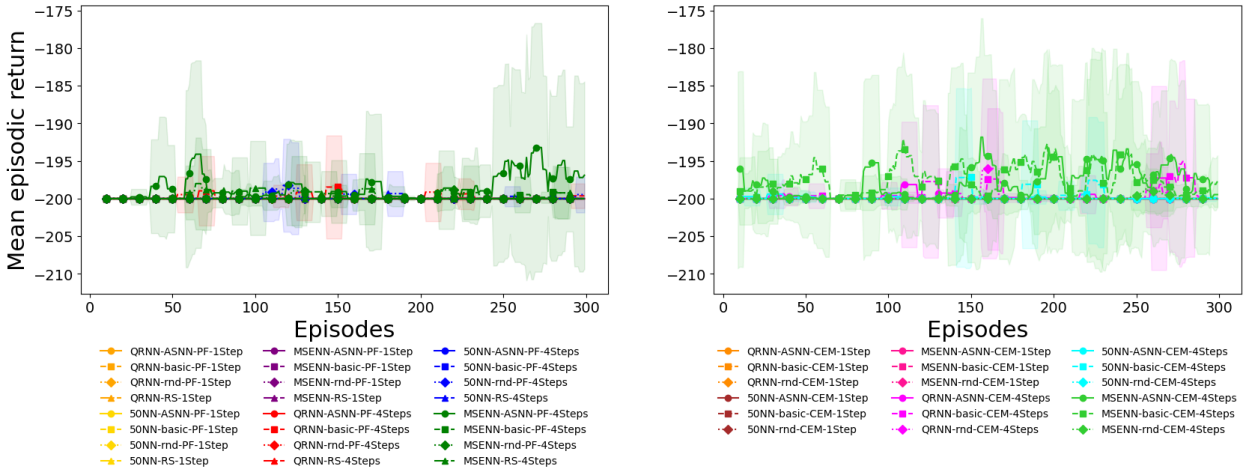


Figure 4.7 Comparing the performance of our MPC methods on the discrete Mountain Car when taking the optimized action once or repeating it four times in the environment.

Figure 4.7 and Table 4.11 show that repeating the same action for four environment steps is very slightly advantageous compared to only taking it once in our MPC methods when applied to the discrete Mountain Car environment. Using this idea for the MPC methods and the three trajectory optimization methods (CEM, iCEM, and MPPI) generally leads to huge improvements for the continuous Mountain Car environment as Figure 4.8 and Table 4.11 show. There is no data for MPPI, CEM, and iCEM for the discrete Mountain Car task, since these methods are made for continuous action spaces.

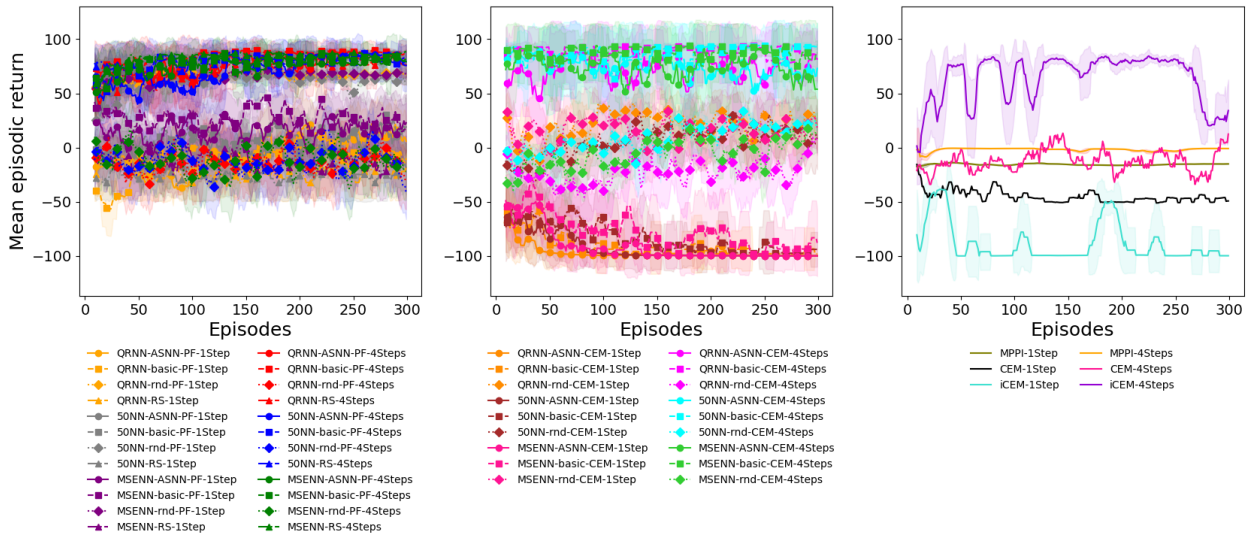


Figure 4.8 Comparing the performance of our MPC methods on the continuous Mountain Car when taking the optimized action once or repeating it four times in the environment.

Table 4.11 Area under the curve of the episodic return for the continuous and discrete versions of the Mountain Car environment when taking one or four steps with the optimized action

	Mountain Car Discrete repeated 1 step	Mountain Car Discrete repeated 4 steps	Mountain Car Continuous 1 step	Mountain Car Continuous repeated 4 steps
QRNN-ASNN-PF	-59800.00 ± 0.00	-59799.33 ± 0.94	-1251.85 ± 9755.01	22510.25 ± 2687.41
QRNN-basic-PF	-59800.00 ± 0.00	-59781.00 ± 19.61	-3259.55 ± 15340.32	24463.34 ± 2600.91
QRNN-rnd-PF	-59800.00 ± 0.00	-59800.00 ± 0.00	19671.26 ± 1699.08	-4031.02 ± 3416.00
QRNN-RS	-59800.00 ± 0.00	-59757.67 ± 25.00	-6741.55 ± 1859.36	22571.00 ± 2022.86
50NN-ASNN-PF	-59800.00 ± 0.00	-59800.00 ± 0.00	-107.67 ± 9222.49	21156.89 ± 5282.43
50NN-basic-PF	-59800.00 ± 0.00	-59800.00 ± 0.00	3009.88 ± 9906.88	23620.66 ± 2716.22
50NN-rnd-PF	-59800.00 ± 0.00	-59762.33 ± 45.05	17755.07 ± 3139.43	-3668.27 ± 5845.17
50NN-RS	-59800.00 ± 0.00	-59790.33 ± 8.18	-7477.44 ± 1262.09	23475.75 ± 2487.08
MSENN-ASNN-PF	-59800.00 ± 0.00	-59357.17 ± 294.15	4624.14 ± 7014.74	22646.30 ± 1621.73
MSENN-basic-PF	-59800.00 ± 0.00	-59765.00 ± 25.15	8272.69 ± 5056.53	23406.81 ± 996.06
MSENN-rnd-PF	-59800.00 ± 0.00	-59778.00 ± 31.11	19758.46 ± 1647.30	-3186.53 ± 2272.49
MSENN-RS	-59800.00 ± 0.00	-59728.00 ± 47.42	-3657.45 ± 1215.75	24779.16 ± 377.49
QRNN-ASNN-CEM	-59800.00 ± 0.00	-59728.33 ± 70.77	-28828.97 ± 1507.05	21774.89 ± 73.42
QRNN-basic-CEM	-59800.00 ± 0.00	-59671.00 ± 45.99	-27104.62 ± 2251.05	26366.36 ± 973.40
QRNN-rnd-CEM	-59800.00 ± 0.00	-59760.00 ± 56.57	6364.54 ± 5107.60	-6563.22 ± 4608.83
50NN-ASNN-CEM	-59800.00 ± 0.00	-59782.00 ± 10.98	-27745.67 ± 3017.28	22553.39 ± 3850.98
50NN-basic-CEM	-59800.00 ± 0.00	-59721.33 ± 14.82	-25068.01 ± 5765.54	25931.50 ± 1544.34
50NN-rnd-CEM	-59800.00 ± 0.00	-59800.00 ± 0.00	1882.73 ± 8688.69	2700.03 ± 13084.25
MSENN-ASNN-CEM	-59800.00 ± 0.00	-59133.67 ± 244.19	-27160.30 ± 3854.85	22376.78 ± 3499.66
MSENN-basic-CEM	-59800.00 ± 0.00	-59066.00 ± 705.91	-23263.88 ± 5918.65	26346.90 ± 809.08
MSENN-rnd-CEM	-59800.00 ± 0.00	-59780.67 ± 13.77	4761.95 ± 5127.12	-1451.56 ± 11856.69
MPPI	NA	NA	-4647.97 ± 517.69	-531.36 ± 27.06
CEM	NA	NA	-13377.27 ± 929.09	-3724.58 ± 1155.38
iCEM	NA	NA	-26390.42 ± 4195.58	18885.45 ± 2124.18

CHAPTER 5 DISCUSSION - OVERALL ALGORITHM COMPARISON

5.1 Discrete action space

As table 4.7 shows, the top three methods are QR-DQN, QRNN-ASNN-CEM, and QRNN-basic-CEM. The use of CEM appears to have a clear advantage compared to PF for environments with discrete action spaces. The QRNN-ASNN-CEM and the other top MPC-CEM methods were able to solve the Cart Pole and Acrobot environments virtually immediately; however, they were unable to solve the Mountain Car and Lunar Lander environments. The QR-DQN was the overall best method and initially performed worse than the MPC methods, but it quickly converged to similar values or overpassed them. Overall, QR-DQN appears to be the most reliable method for discrete action space problems. For the MPC methods, using the QRNN as a model of the environment seems to be advantageous compared to the 50NN or the MSENN. Similarly, using the ASNN over generating action sequences by randomly uniformly picking among the possible actions seems slightly better.

The general trends in the results for problems with discrete action spaces are the following:

- CEM > PF > RS
- ASNN > basic > rnd
- QRNN > MSENN > 50NN

5.2 Continuous action space

As table 4.8 shows, the top three algorithms are 50NN-ASNN-PF, MSENN-ASNN-PF, and QRNN-basic-PF. Overall, using PF or CEM in MPC yields similar results, but PF appears to be slightly more advantageous for continuous action space tasks. More precisely, the MPC-rnd algorithms perform exceptionally well on control environments, such as Cart Pole continuous, Inverted Pendulum, and continuous Lunar Lander. Similarly, the MPC-basic-CEM methods had excellent performance on the continuous Mountain Car problem. Using the QRNN seems to be somewhat beneficial compared to the 50NN and MSENN models of the environment. This is especially true when it is combined with PF on problems such as continuous Cart Pole, Pendulum, MuJoCo Reacher, and Panda Reach. The QRNN and the other models weren't suitable for environments where it was challenging to learn a model of the environment, such as the continuous Lunar Lander environment. This is because the

performance relies heavily on the quality of the model, as it is used to predict the next state when generating action sequences. In comparison, the MPC-ASNN algorithms do better on robotics environments, such as MuJoCo Reacher and Panda Reach. For these environments, a model of the environment appears to be more intuitive for learning. In contrast, MPC-rnd algorithms do not rely on the model of the environment to generate action sequences and therefore outperform the other algorithms.

The general trends in the results for problems with continuous action spaces are the following:

- CEM $\leftrightarrow$ PF, but PF is generally slightly better
- MPC methods using PF or CEM are better than RS, besides the 50NN-basic algorithms
- basic $>$ ASNN $>$ rnd on control tasks (Cart Pole, Lunar Lander, Mountain Car, Inverted Pendulum, Pendulum). See Table B.1 for the in-depth data.
- ASNN $>$ rnd $>$ basic on robotics tasks (MuJoCo Reacher, Panda Reach). See Table B.2 for the in-depth data.
- QRNN $>$ 50NN $>$ MSENN

CHAPTER 6 CONCLUSION

Before giving a summary, limitations and future work, let's recap the different components of the MPC methods I have presented.

Models of the environment:

- QRNN: The neural network (NN) predicts 11 quantiles for the next state but only uses the 50% quantile. Trained with quantile regression loss. We also compare only using the mid quantile for next state prediction or sampling the predicted quantiles.
- 50NN: Predicts a single next state, trained with quantile regression loss using only the 50% quantile.
- MSENN: Predicts a single next state, trained with mean squared error (MSE).

Ways to generate initial particles:

- Sample a neural network: Used in the discrete and continuous versions (though the sampling method differs) of the MPC algorithms using an action sequence neural network (ASNN).
- Uniformly randomly (discrete) pick among the possible actions: Used in the discrete action space environment versions of the MPC algorithms called basic, rnd, and RS.
- Sample a uniform distribution: Used in the continuous action space environment versions of the MPC algorithms called basic, rnd, and RS.

How to modify the action sequences after taking a step in the environment:

- Sample a neural network: Shift the previous step's action sequences and replace the vacant column with actions sampled from an NN, used in the MPC methods using an ASNN.
- Shift the action sequences to replace the action taken and uniformly at random pick among the possible actions: Used in the discrete action space environment versions of the MPC algorithms called basic.
- Shift the action sequences to replace the action taken and sample a uniform distribution: Used in the continuous action space environment versions of the MPC methods called rnd and RS.

MPC techniques used to improve the action sequences:

- Particle filtering (PF)
- Cross-entropy method (CEM)

See Section 3.1.4 for more details on them and our specific implementation.

6.1 Summary

This thesis presented an in-depth study of the impact of the different components of an MPC algorithm. More precisely, we studied the use of different models of the environment, methods to generate action sequences at the start of an episode, MPC techniques to improve the action sequences, and strategies to modify them after taking a step in the environment with the first action of the best action sequences found by MPC. This lead to the comparison of 21 different algorithms (QRNN-ASNN-PF, QRNN-basic-PF, QRNN-rnd-PF, QRNN-RS, 50NN-ASNN-PF, 50NN-basic-PF, 50NN-rnd-PF, 50NN-RS, MSENN-ASNN-PF, MSENN-basic-PF, MSENN-rnd-PF, MSENN-RS, QRNN-ASNN-CEM, QRNN-basic-CEM, QRNN-rnd-CEM, 50NN-ASNN-CEM, 50NN-basic-CEM, 50NN-rnd-CEM, MSENN-ASNN-CEM, MSENN-basic-CEM, MSENN-rnd-CEM). We compared these methods to multiple algorithms from the literature, including those from the MFRL (DQN, IV-DQN, QR-DQN, A2C, PPO, DDPG, SAC, TD3, TQC), MBRL (GP-MPC, PETS-CEM), and trajectory optimization (MPPI, CEM, iCEM). Twelve standard RL benchmark environments with a clear goal state, drawn from the OpenAI Gymnasium [2] and Panda Gym [58] packages, were used. The comparisons among the algorithms for a given problem were made by examining the area under the curve (AUC) of the episodic return learning curves. To obtain an overall comparison of the algorithms, the AUC for each problem was averaged over each algorithm for each action space problem type. Ultimately, the top three methods for discrete action space problems were QR-DQN, QRNN-ASNN-CEM, and MSENN-basic-CEM. For continuous action space problems, they were 50NN-ASNN-PF, MSENN-ASNN-PF, and QRNN-basic-PF. It appears that PF is slightly preferable to CEM for continuous action space problems, while CEM is better suited for discrete action space problems. However, the most effective method found depends on the specific environment. Similarly, the QRNN seems to be generally more advantageous than the 50NN and MSENN models of the environment. This is clearer for the discrete action space environments than for the continuous action space ones. The MPC-basic methods are generally more suitable for continuous control problems, while the MPC-ASNN algorithms are preferable for robotic environments.

6.2 Limitations

The project's main limitation is that the QRNN model only uses the mid or 50% quantile for its prediction of the next state instead of utilizing the full quantile distribution. While we present a method that samples the generated quantiles, it assumes that they are independent, which isn't the case. Another limitation is that only the standard noiseless versions of the benchmark problems were used. It isn't possible to say with certainty that each algorithm would perform similarly in a noisy version of the environment or a real-world control task.

6.3 Future Research

Future research should focus on integrating multiple quantiles into the QRNN model's next state prediction process, as well as comparing different algorithms on more complex control and robotics tasks, such as the MuJoCo Pusher [2], the Panda Gym Push, and the Panda Gym slide environments [58].

The QRNN next state prediction sampling method described in section 3.3.4 of the methodology produces poor results compared to the mid/50% quantile method as shown in section 4.8.1. This is because it assumes that the state components are independent. A method that would not make this assumption would be to have a neural network that predicts the principal component analysis (PCA) vectors of the covariance matrix of the next state predictions. For each vector, we can predict the quantiles, which can be sampled independently to get a next state prediction.

REFERENCES

[1] Piscine26, "Reinforcement learning is type of basic machine learning paradigms, alongside supervised learning and unsupervised learning," Adobe Stock. [Online]. Available: https://stock.adobe.com/ca/images/reinforcement-learning-is-type-of-basic-machine-learning-paradigms-alongside-

[2] "Gymnasium," Farama Foundation, 2025. [Online]. Available: https://gymnasium.farama.org/

[3] S. L. C. E. M. L. C. U. M. S. L. P. S. S. Q. G. L. M. M. N. D. Marco Bonizzato, Rose Guay Hottin, "Autonomous optimization of neuroprosthetic stimulation parameters that drive the motor cortex and spinal cord outputs in rats and monkeys," *Cell Reports Medicine*, vol. 4, no. 4, Apr. 2023. [Online]. Available: https://doi.org/10.1016/j.xcrm.2023.101008

[4] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv*, Jul. 2018. [Online]. Available: https://arxiv.org/abs/1807.02811

[5] L. C. . C. Muller, "Benchmarking the performance of bayesian optimization across multiple experimental materials science domains," *npj Computational Materials*, vol. 7, Nov. 2021. [Online]. Available: https://doi.org/10.1038/s41524-021-00656-9

[6] ——, "Bayesian optimization in drug discovery," *Methods in Molecular Biology*, vol. 2716, Sep. 2023. [Online]. Available: https://link.springer.com/protocol/10.1007/978-1-0716-3449-3_5

[7] R. P. A. Jasper Snoek, Hugo Larochelle, "Practical bayesian optimization of machine learning algorithms," *arXiv*, Jun. 2012. [Online]. Available: https://arxiv.org/abs/1206.2944

[8] K. S. D. M. D. P. K. S. Pawan Sharma, Yixuan Du, "Novel comprehensive analysis of skilled reaching and grasping behavior in adult rats," *Journal of Neuroscience Methods*, vol. 411, Apr. 2024. [Online]. Available: https://doi.org/10.1016/j.jneumeth.2024.110271

[9] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018. [Online]. Available: http://www.incompleteideas.net/book/the-book-2nd.html

[10] V. Jain, S. Liu, and G. Iyer, "Coping with sample inefficiency of deep-reinforcement learning (drl) for embodied ai," *CML WiML*, Jul. 2020. [Online]. Available: https://www.ri.cmu.edu/publications/coping-with-sample-inefficiency-of-deep-reinforcement-learning-drl-for-embodied-ai/

[11] A. J. S. Ibrahim, M. Mostafa and P. Osinenko, "Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications," *arXiv*, Jul. 2024. [Online]. Available: https://arxiv.org/html/2408.10215v1

[12] I. C. J. H. Y. W. E. L. S. Z. G. Z. P. A. J. B. Tingwu Wang, Xuchan Bao, "Benchmarking model-based reinforcement learning," *arXiv*, Jul. 2019. [Online]. Available: https://arxiv.org/abs/1907.02057

[13] T. B. M. Schwenzer, M. Ay and D. Abel, "Review on model predictive control: an engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, pp. 1327–1349, Nov. 2021. [Online]. Available: https://link.springer.com/article/10.1007/s00170-021-07682-3#citeas

[14] H. S. Spong, M.W. and M. Vidyasagar, *Robot Modeling and Control*, 1st ed. New York, NY: Wiley, 2006. [Online]. Available: https://www.researchgate.net/profile/Mohamed_Mourad_Lafifi/post/How_to_avoid_singular_configurations/attachment/59d6361b79197b807799389a/AS%3A386996594855942%401469278586939/download/Spong+-+Robot+modeling+and+Control.pdf

[15] S. Q. Oussama Khatib and D. Williams, "Robot planning and control," *Robotics and Autonomous Systems*, vol. 21, no. 3, Sep. 1997. [Online]. Available: https://doi.org/10.1016/S0921-8890(96)00078-4

[16] "Part 1: Key concepts in rl," Open AI Spinning Up, 2025. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

[17] V. Mnhih and al., "Playing atari with deep reinforcement learning," *arXiv*, Dec. 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[18] P. A. S. L. Tuomas Haarnoja, Aurick Zhou, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv*, Jan. 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1801.01290

[19] A. P. N. H. T. E. Y. T. D. S. D. W. Timothy P. Lillicrap, Jonathan J. Hunt, "Continuous control with deep reinforcement learning," *arXiv*, Sep. 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1509.02971

[20] D. M. Scott Fujimoto, Herke van Hoof, "Addressing function approximation error in actor-critic methods," *arXiv*, May 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1802.09477

[21] P. D. A. R. O. K. John Schulman, Filip Wolski, "Proximal policy optimization algorithms," *arXiv*, Jul. 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1707.06347

[22] C. E. R. Marc Peter Deisenroth, "Pilco: a model-based and data-efficient approach to policy search," *ICML*, 2011. [Online]. Available: https://dl.acm.org/doi/10.5555/3104482.3104541

[23] R. M. S. L. Kurtland Chua, Roberto Calandra, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *ArXiv*, May 2018. [Online]. Available: https://arxiv.org/abs/1805.12114

[24] J. B. Tingwu Wang, "Exploring model-based planning with policy networks," *arXiv*, Jun. 2019. [Online]. Available: https://arxiv.org/abs/1906.08649

[25] A. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, Jan. 2010. [Online]. Available: https://arxiv.org/html/2408.10215v1

[26] R. I. M. G. B. Vincent François-Lavet, Peter Henderson and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends in Machine Learning*, vol. 11, no. 3-4, Apr. 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1811.12560

[27] M. S. Jim Holdsworth, "what is deep learning?" IBM, june 2024. [Online]. Available: https://www.ibm.com/think/topics/deep-learning

[28] M. H. H. v. H. M. L. Z. Wang, T. Schaul and N. de Freitas., "Dueling network architectures for deep reinforcement learning," *arXiv*, Nov. 2015. [Online]. Available: https://arxiv.org/abs/1511.06581

[29] M. G. B. W. Dabney, M. Rowland and R. Munos, "Distributional reinforcement learning with quantile regression," *arXiv*, Oct. 2017. [Online]. Available: https://arxiv.org/abs/1710.10044

[30] K. M. V. Mai and L. Paull, "Sample efficient deep reinforcement learning via uncertainty estimation," *arXiv*, Jan. 2022. [Online]. Available: https://arxiv.org/abs/2201.01666

[31] A. G. D. V. Arsenii Kuznetsov, Pavel Shvechikov, "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics," *arXiv*, May 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2005.04269

[32] "Nllloss," PyTorch, 2025. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html#torch.nn.NLLLoss

[33] M. P. D. Sanket Kamthe, "Data-efficient reinforcement learning with probabilistic model predictive control," *ArXiv*, Feb. 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1706.06491

[34] "Data-efficient-reinforcement-learning-with-probabilistic-model-predictive-control," GitHub, 2021. [Online]. Available: https://github.com/SimonRennotte/Data-Efficient-Reinforcement-Learning-with-Probabilistic-Model-Predictive-Control

[35] M. P. Deisenroth, "E cient reinforcement learning using gaussian processes," doctoral thesis, Faculty of Informatics Institute for Anthropomatics Intelligent Sensor-Actuator-Systems Laboratory (ISAS), Seattle, WA, USA, 2010. [Online]. Available: https://deisenroth.cc/pdf/thesis.pdf

[36] "The l-bfgs-b algorithm," Stanford Exploration Project, 2004. [Online]. Available: https://sepwww.stanford.edu/data/media/public/docs/sep117/antoine1/paper_html/node6.html

[37] N. d. F. Eric Brochu, Vlad M. Cora, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv*, Dec. 2010. [Online]. Available: https://arxiv.org/abs/1012.2599

[38] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, May 2012. [Online]. Available: https://doi.org/10.1177/0278364912444543

[39] R. R. . Y. G. Shie Mannor, "Review on model predictive control: an engineering perspective," *The International Journal of Advanced Manufacturing Technology*, 2003. [Online]. Available: https://link.springer.com/article/10.1007/s00170-021-07682-3

[40] I. Szita and A. Lörincz, "Learning tetris using the noisy cross-entropy method," *Neural Computation*, vol. 18, no. 12, 2006. [Online]. Available: https://doi.org/10.1162/neco.2006.18.12.2936

[41] S. B. J. A. J. S. M. R. G. M. C. Pinneri, S. Sawant, "Sample-efficient cross-entropy method for real-time planning," *arXiv*, Aug. 2020. [Online]. Available: https://arxiv.org/abs/2008.06389

[42] "Pytorch cem implementation," GitHub, 2019. [Online]. Available: https://github.com/LemonPi/pytorch_cem/tree/master

[43] B. G. J. M. R. Grady Williams, Paul Drews and E. A. Theodorou, "Aggressive driving with model predictive path integral control," *IEEE International Conference on Robotics and Automation*, May 2016. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7487277

[44] B. G. P. D. J. M. R. B. B. E. A. T. Grady Williams, Nolan Wagener, "Information theoretic mpc for model-based reinforcement learning," *IEEE International Conference on Robotics and Automation*, Feb. 2017. [Online]. Available: https://doi.org/10.1109/ICRA.2017.7989202

[45] J. Noble, "What is llm temperature?" IBM, Dec. 2004. [Online]. Available: https://www.ibm.com/think/topics/llm-temperature#:~:text=Different%20temperature%20settings%20essentially%20introduce,world%20applications%20of%20text%20generation.

[46] "Boltzmann weight," Encyclopedia of Mathematics, Jul. 2020. [Online]. Available: https://encyclopediaofmath.org/wiki/Boltzmann_weight

[47] E. T. Weiwei Li, "Iterative linear quadratic regulator design for nonlinear biological movement systems," *Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics*, Jan. 2004. [Online]. Available: http://maeresearch.ucsd.edu/skelton/publications/weiwei_ilqg_biological.pdf

[48] K. C. J. M. T. L. Z. Cheng, Y. Li, "Neural-ilqr: A learning-aided shooting method for trajectory optimization," *arXiv*, Nov. 2020. [Online]. Available: https://arxiv.org/abs/2011.10737

[49] "Stable-baselines3 contrib docs!" Stable-Baselines3 Contrib, 2025. [Online]. Available: https://sb3-contrib.readthedocs.io/en/master/index.html

[50] D. A. Nix and A. S. Weigend, "Estimating the mean and variance of the target probability distribution," *arXiv*, vol. 1, Jun. 1994. [Online]. Available: https://ieeexplore.ieee.org/document/374138

[51] S. Turney, "Quartiles quantiles | calculation, definition interpretation," may 2022. [Online]. Available: https://www.scribbr.com/statistics/quartiles-quantiles/

[52] V. Efimov, "Quantile loss quantile regression," towards data science, Jan. 2023. [Online]. Available: https://towardsdatascience.com/quantile-loss-and-quantile-regression-b0689c13f54d/

[53] "Pytorch icem implementation," GitHub, 2023. [Online]. Available: https://github.com/UM-ARM-Lab/pytorch_icem

[54] "Pytorch mppi implementation," GitHub, 2023. [Online]. Available: https://github.com/UM-ARM-Lab/pytorch_mppi/tree/master

[55] Britannica, "Mean squarred error," june 2025. [Online]. Available: https://www.britannica.com/science/mean-squared-error

[56] V. Jayaswal, "Laplace smoothing in naive bayes algorithm," towards data science, Nov. 2020. [Online]. Available: https://towardsdatascience.com/laplace-smoothing-in-naive-bayes-algorithm-9c237a8bdece/

[57] "Learning tetris using the noisy cross entropy method," GitHub, 2023. [Online]. Available: https://github.com/corentinpla/Learning-Tetris-Using-the-Noisy-Cross-Entropy-Method/tree/main

[58] E. D. Q. Gallouédec, N. Cazin and L. Chen, "panda-gym: Open-source goal-conditioned environments for robotic learning," *arXiv*, Jun. 2021. [Online]. Available: https://arxiv.org/abs/2106.13687

[59] "Mbrl-lib," GitHub, 2021. [Online]. Available: https://github.com/facebookresearch/mbrl-lib

[60] "mbbl," GitHub, 2019. [Online]. Available: https://github.com/WilsonWangTHU/mbbl

[61] R. V. Florian, "Correct equations for the dynamics of the cart-pole system," Center for Cognitive and Neural Studies (Coneural), Cluj-Napoca, Romania, Tech. Rep., Jul 2005, updated Feb. 10, 2007. [Online]. Available: http://coneural.org/florian/papers/05_cart_pole.pdf.

[62] A. Fortin, *Analyse numérique pour ingénieurs*, 5th ed. Presses Intern. Polytechnique, 2016.

[63] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems*, D. Touretzky, M. C. Mozer, and M. Hasselmo, Eds., vol. 8. MIT Press, 1996. [Online]. Available: https://proceedings.neurips.cc/paper/1995/file/8f1d43620bc6bb580df6e80b0dc05c48-Paper.pdf

[64] M. M. A. G. T. P. L. T. H. D. S. K. K. Volodymyr Mnih, Adrià Puigdomènech Badia, "Asynchronous methods for deep reinforcement learning," *arXiv*, Feb. 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1602.01783

[65] "Iv-rl: Sample efficient deep reinforcement learning via uncertainty estimation," GitHub, 2021. [Online]. Available: https://github.com/montrealrobotics/iv_rl

[66] "Quantile regression dqn," GitHub, 2018. [Online]. Available: https://github.com/senya-ashukha/quantile-regression-dqn-pytorch/tree/master

[67] "Stable-baselines3 docs - reliable reinforcement learning implementations," Stable-Baselines3, 2025. [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/

[68] "Stable-baslines3," Hugging Face, 2022. [Online]. Available: https://huggingface.co/sb3

[69] "ilqr," GitHub, 2021. [Online]. Available: https://github.com/Bharath2/iLQR/tree/main

APPENDIX A STABLE BASELINES3 RL ALGORITHM HYPERPARAMETERS

Let's list out all the hyperpameters used by each algorithm for the different benchmark environments. These are taken from Stable baselines3 Hugging face page [68].

A.1 Acrobot

A.1.1 A2C

Table A.1 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	0.0

Table A.1 A2C hyperparameters used on the continuous and discrete Cart Pole environments

A.1.2 PPO

Table A.2 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	0.0
gae_lambda	0.94
gamma	0.99
n_epochs	4
n_steps	256

Table A.2 PPO hyperparameters used on the Acrobot environment

A.2 Discrete and continuous Cart Pole

As mentionned earlier on, the hyperparameters were given for the discrete version of the environment but I assumed it was fair to reuse them for the continuous version of the environment. This leads to only A2C and PPO having precise hyperparameters. The other algorithms use the default ones.

A.2.1 A2C

Table A.3 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	256
clip_range	'lin_0.2'
ent_coeff	0.0
gae_lambda	0.8
gamma	0.8
learning_rate	'lin_0.001'
n_epochs	20
n_steps	32

Table A.3 A2C hyperparameters used on the continuous and discrete Cart Pole environments

A.2.2 PPO

Table A.4 gives the hyperparameters used.

A.3 Discrete Lunar Lander

A.3.1 A2C

Table A.5 gives the hyperparameters used.

A.3.2 PPO

Table A.6 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	256
ent_coeff	0.0
gae_lambda	0.8
gamma	0.8
learning_rate	'lin_0.001'
n_epochs	20
n_steps	32

Table A.4 PPO hyperparameters used on the continuous and discrete Cart Pole environments

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	1e-05
gamma	0.995
learning_rate	'lin_0.00083'
n_steps	5

Table A.5 PPO hyperparameters used on the discrete Lunar Lander environment

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	64
ent_coeff	0.01
gae_lambda	0.98
gamma	0.999
n_epochs	4
n_steps	1024

Table A.6 PPO hyperparameters used on the discrete Lunar Lander environments

A.4 Discrete Mountain Car

A.4.1 A2C

Table A.7 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	0.0

Table A.7 PPO hyperparameters used on the discrete Mountain Car environment

A.4.2 PPO

Table A.8 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	0.0
gae_lambda	0.98
gamma	0.99
n_epochs	4
n_steps	16

Table A.8 PPO hyperparameters used on the discrete Mountain Car environment

A.5 Pendulum

A.5.1 A2C

Table A.9 gives the hyperparameters used.

A.5.2 DDPG

Table A.10 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	0.0
gae_lambda	0.9
gamma	0.99
learning_rate	7e-4
max_grad_norm	0.5
n_steps	8
normalize_advantage	False
policy_kwargs	dict(log_std_init=-2, ortho_init=False)
use_rms_prop	True
use_sde	True
vf_coef	0.4

Table A.9 A2C hyperparameters used on the Pendulum environment

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
buffer_size	200000
gamma	0.98
gradients_steps	-1
learning_rate	0.001
policy_kwargs	dict(net_arch=[400,300])
train_freq	(1,'episode')

Table A.10 DDPG hyperparameters used on the Pendulum environment

A.5.3 PPO

Table A.11 gives the hyperparameters used.

A.5.4 SAC

Table A.12 gives the hyperparameters used.

A.5.5 TD3

Table A.13 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
clip_range	0.2
ent_coeff	0.0
gae_lambda	0.95
gamma	0.9
learning_rate	0.001
n_epochs	10
n_steps	1024
sde_sample_freq	4
use_sde	True

Table A.11 PPO hyperparameters used on the Pendulum environment

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
learning_rate	0.001

Table A.12 SAC hyperparameters used on the Pendulum environment

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
buffer_size	200000
gamma	0.98
gradients_steps	-1
learning_rate	0.001
policy_kwargs	dict(net_arch=[400,300])
train_freq	(1,'episode')

Table A.13 TD3 hyperparameters used on the Pendulum environment

A.5.6 TQC

Table A.14 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
learning_rate	0.001

Table A.14 TQC hyperparameters used on the Pendulum environment

A.6 Lunar Lander continuous

A.6.1 A2C

Table A.15 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	256
clip_range	'lin_0.2'
ent_coeff	0.0
gae_lambda	0.9
gamma	0.99
learning_rate	7e-4
max_grad_norm	0.5
n_steps	8
normalize_advantage	False
policy_kwargs	dict(log_std_init=-2, ortho_init=False)
use_rms_prop	True
use_sde	True
vf_coef	0.4

Table A.15 PPO hyperparameters used on the continuous and discrete Cart Pole environments

A.6.2 DDPG

Table A.16 gives the hyperparameters used.

A.6.3 PPO

Table A.17 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
buffer_size	200000
gamma	0.98
gradients_steps	-1
learning_rate	0.001
policy_kwargs	dict(net_arch=[400,300])
train_freq	(1, 'episode')

Table A.16 PPO hyperparameters used on the continuous and discrete Cart Pole environments

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	64
ent_coeff	0.0
gae_lambda	0.98
gamma	0.999
n_epochs	4
n_steps	1024

Table A.17 PPO hyperparameters used on the continuous and discrete Cart Pole environments

A.6.4 SAC

Table A.18 gives the hyperparameters used.

A.6.5 TD3

Table A.19 gives the hyperparameters used.

A.6.6 TQC

Table A.20 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	256
buffer_size	1000000
ent_coeff	'auto'
gae_lambda	0.98
gradient_steps	1
learning_rate	'lin_7.3e-4'
policy$_k wargs$	dict(net$_a rch = [400, 300]$)
tau	0.01
train_freq	1

Table A.18 SAC hyperparameters used on the continuous and discrete Cart Pole environments

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
buffer_size	200000
gamma	0.98
gradient_steps	-1
learning_rate	0.001
policy_kwargs	dict(net_arch=[400,300])
train_freq	(1,'episode')

Table A.19 TD3 hyperparameters used on the continuous and discrete Cart Pole environments

A.7 Mountain Car continuous

A.7.1 A2C

Table A.21 gives the hyperparameters used.

A.7.2 DDPG

Table A.22 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	256
buffer_size	1000000
ent_coeff	'auto'
gae_lambda	0.98
gradient_steps	1
learning_rate	'lin_7.3e-4'
policy_kwargs	dict(net_arch=[400, 300])
tau	0.01
train_freq	1

Table A.20 TQC hyperparameters used on the continuous and discrete Cart Pole environments

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	0.0
n_steps	100
policy_kwargs	dict(log_std_init=0.0, ortho_init=False)
sde_sample_freq	16
use_sde	True

Table A.21 PPO hyperparameters used on the continuous Mountain Car environments

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	0.0
n_steps	100
policy_kwargs	dict(log_std_init=0.0, ortho_init=False)
sde_sample_freq	16
use_sde	True

Table A.22 PPO hyperparameters used on the continuous Mountain Car environments

A.7.3 PPO

Table A.23 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	256
clip_range	0.1
ent_coeff	0.00429
gae_lambda	0.9
gamma	0.9999
learning_rate	'lin_7.77e-05'
n_epochs	10
n_steps	8
policy_kwargs	dict(log_std_init=-3.29, ortho_init=False)
use_sde	True
vf_coeff	0.19

Table A.23 PPO hyperparameters used on the continuous Mountain Car environments

A.7.4 SAC

Table A.24 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	512
buffer_size	50000
enf_coeff	0.1
gamma	0.9999
gradient_steps	32
learning_rate	0.0003
policy_kwargs	dict(log_std_init=-3.67, net_arch=[64, 64])
tau	0.01
train_freq	32
use_sde	True

Table A.24 SAC hyperparameters used on the continuous Mountain Car environments

A.7.5 TD3

Table A.25 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
ent_coeff	0.0
n_steps	100
policy_kwargs	dict(log_std_init=0.0, ortho_init=False)
sde_sample_freq	16
use_sde	True

Table A.25 TD3 hyperparameters used on the continuous Mountain Car environments

A.7.6 TQC

Table A.26 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	512
buffer_size	50000
enf_coeff	0.1
gamma	0.9999
gradient_steps	32
learning_rate	0.0003
policy_kwargs	dict(log_std_init=-3.67, net_arch=[64, 64])
tau	0.01
train_freq	32
use_sde	True

Table A.26 TQC hyperparameters used on the continuous Mountain Car environments

A.8 Panda Reach Sparse

A.8.1 TQC

Table A.27 gives the hyperparameters used.

Hyperparameter	Value
policy	"MlpPolicy"
env	"CartPole-v0"
device	'cpu'
batch_size	256
buffer_size	1000000
enf_coeff	'auto'
gamma	0.95
learning_rate	0.001
policy_kwargs	dict(net_arch=[64,64], n_critics=1)
replay_buffer_class	'HerReplayBuffer'
replay_buffer_kwargs	dict(online_sampling=True, goal_selection_strategy='future', n_sampled_goal=4)

Table A.27 TQC hyperparameters used on the Panda Reach sparse environments

A.8.2 A2C, DDPG, PPO, SAC, TD3

No hyperparameters are defined for these problems, so the default ones are used.

A.9 MuJoCo Reacher, Inverted Pendulum, Panda Reach Dense, MuJoCo Pusher, and Panda Push

Sb3 doesn't define any hyperparameters for any algorithm applied to the MuJoCo Reacher, Inverted Pendulum, Panda Reach Dense, MuJoCo Pusher, and Panda Push (dense and sparse reward versions) environments. The default hyperparameters are used for all methods on these problems.

APPENDIX B QUANTILE REGRESSION NEURAL NETWORK NEXT STATE PREDICTIONS COMPARISON WITH THOSE OF THE ENVIRONMENT

Appendix B presents the comparison of the next state prediction between the quantile neural network and the environment using the cumulative counter values for the other benchmark environments.

B.1 Quantile regression neural network predictions

B.1.1 Acrobot

The following graphs in Figure B.1 show the quantile predictions for the different state components of the Acrobot environment using the cumulative counter values.

B.1.2 Cart Pole

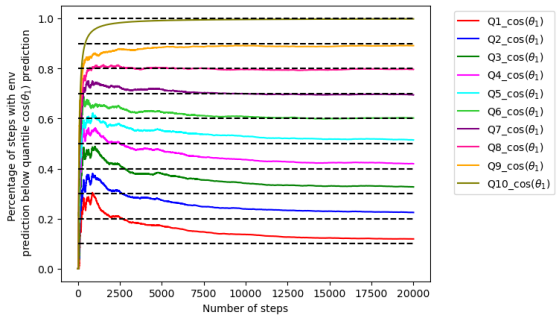
The following graph, Figure B.2, shows a comparison of the quantile predictions with the ground truth environment predictions for the different state components of the Cart Pole environment, using the cumulative counter values. The results are for the discrete version of the environment, but the results shouldn't change for the continuous version.

B.1.3 Lunar Lander

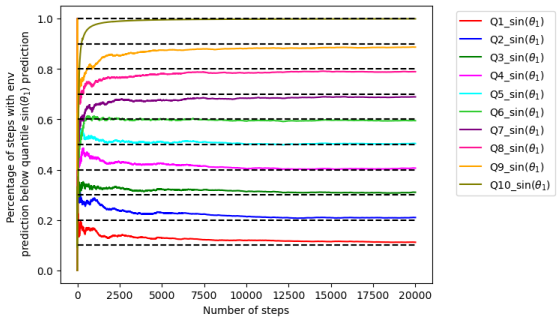
The following graphs in Figures B.3 and B.4 show a comparison of the quantile predictions with the true environment predictions for the different state components of the MuJoCo Reacher environment, using the cumulative counter values. The results are for the discrete version of the environment, but the results shouldn't change for the continuous version.

B.1.4 MuJoCo Reacher

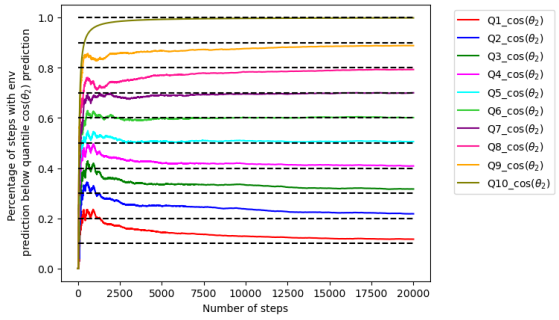
The following graphs in Figures B.5 and B.6 show a comparison of the quantile predictions with the true environment predictions for the different state components of the MuJoCo Reacher environment, using cumulative counter values.



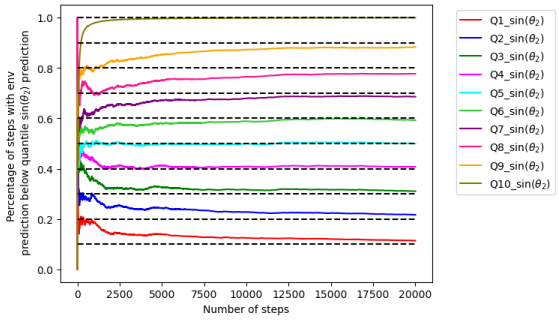
(a) $\cos(\theta_1)$



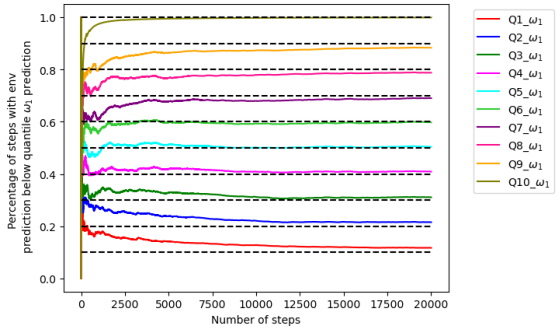
(b) $\sin(\theta_1)$



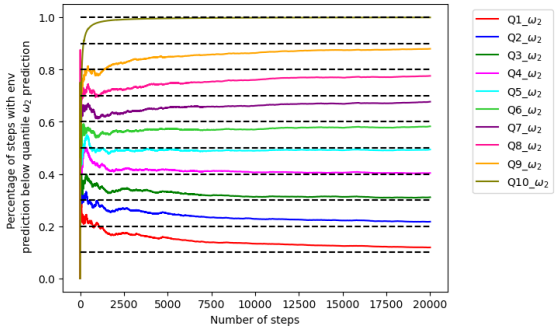
(c) $\cos(\theta_2)$



(d) $\sin(\theta_2)$

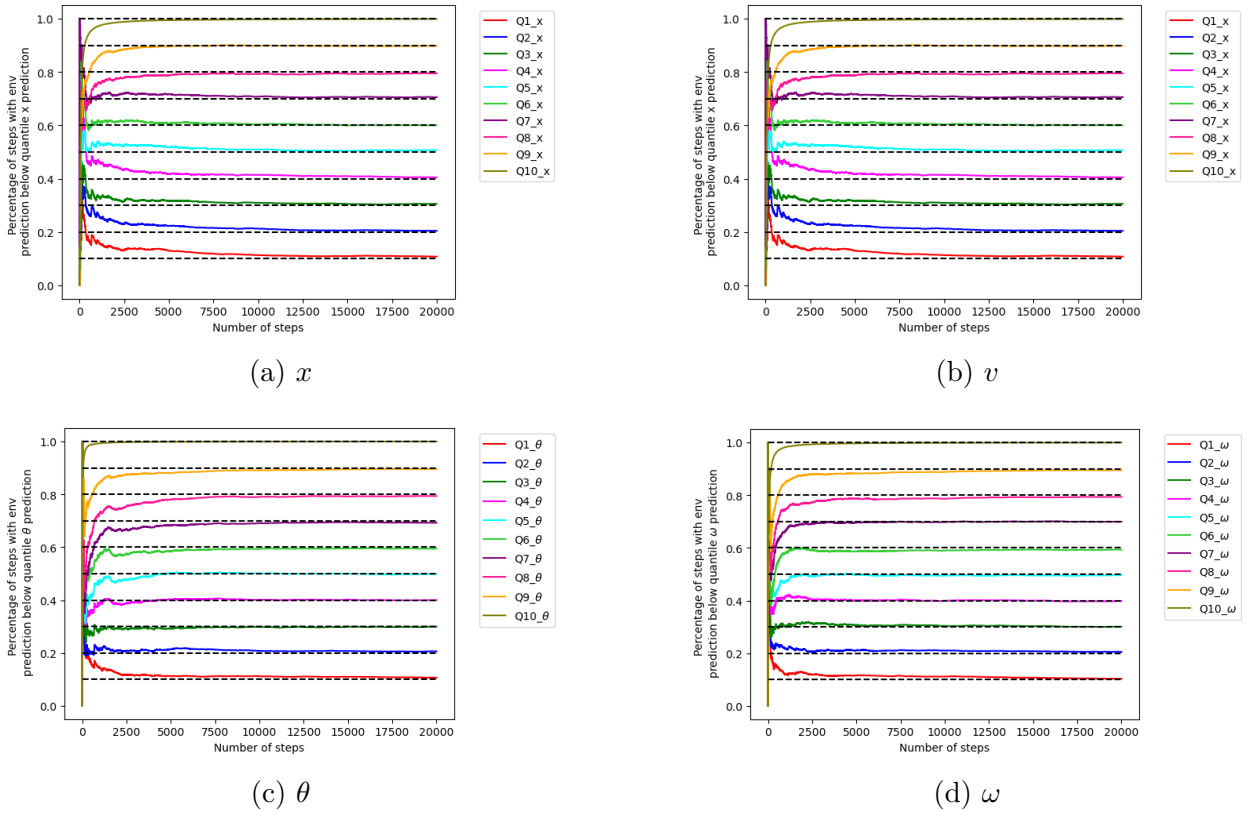


(e) ω_1



(f) ω_2

Figure B.1 Different quantile predictions for the next state components of the Acrobot environment



(a) x (b) v

(c) θ (d) ω

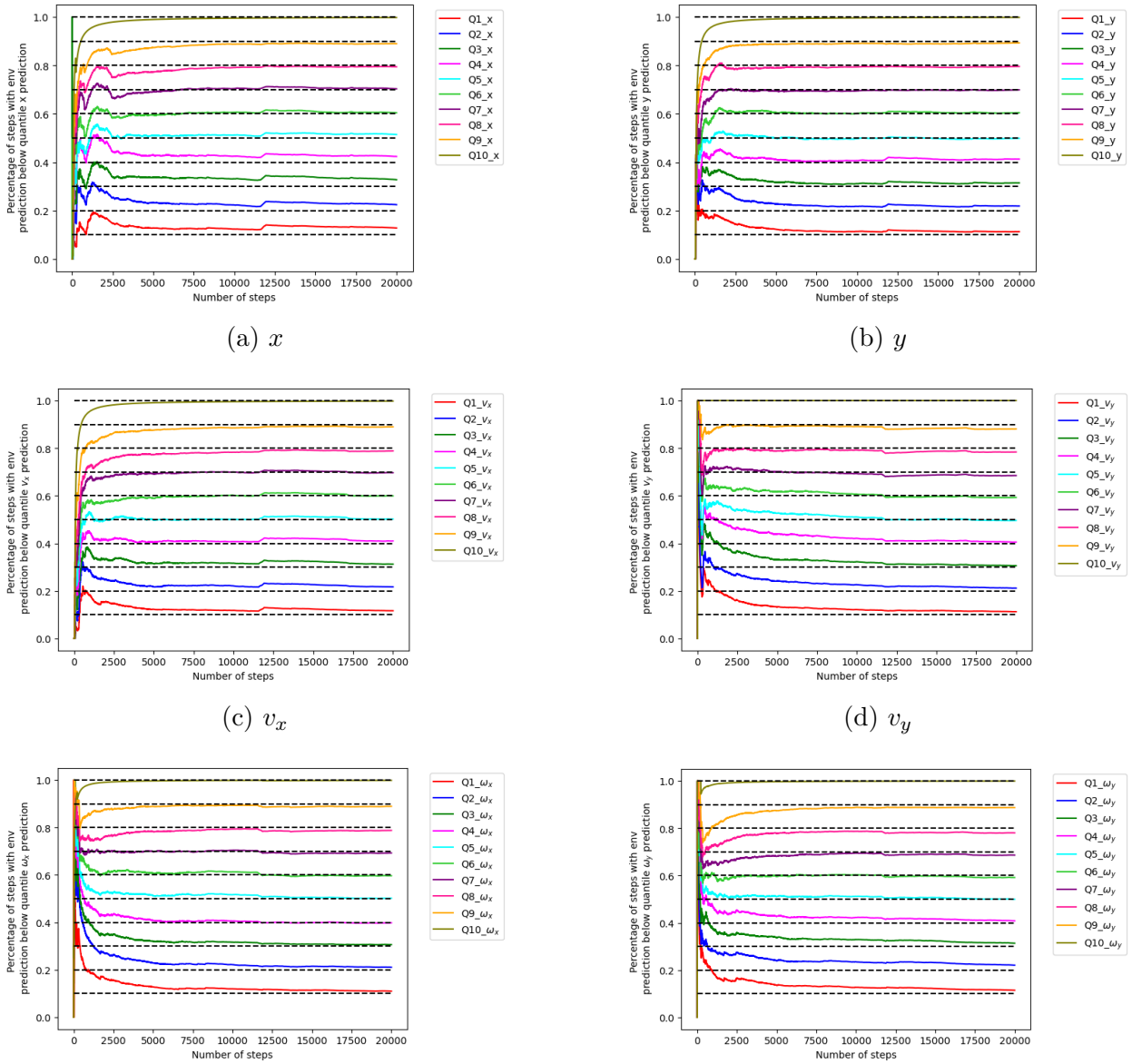
Figure B.2 Different quantile predictions for the next state components x, v, θ, and ω of the Cart Pole environment

B.1.5 Panda Reach

The following graphs in Figure B.7 show a comparison of the quantile predictions with the true environment predictions for the different state components of the Panda Reach environment, using cumulative counter values. The results are for the sparse reward version of the environment, but they should remain unchanged for the dense reward version.

B.1.6 Pendulum

The following graphs in Figure B.8 show a comparison of the quantile predictions with the true environment predictions for the different state components of the Pendulum environment, using cumulative counter values.



(a) x (b) y

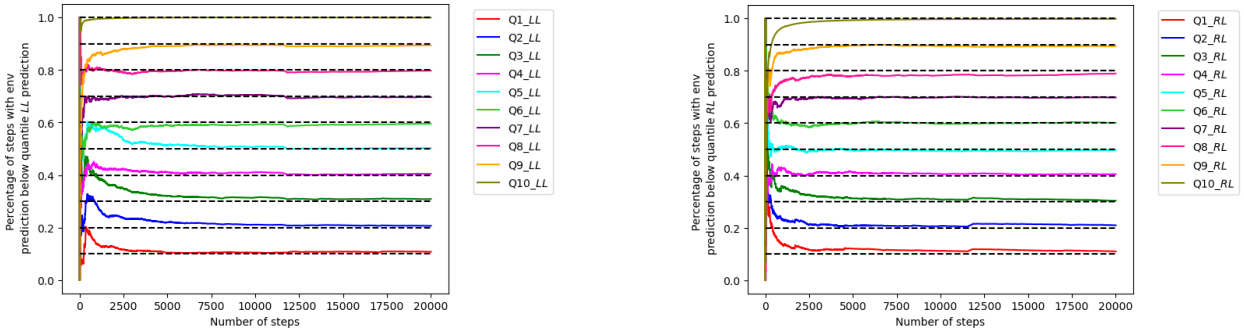
(c) v_x (d) v_y

(e) ω_x (f) ω_y

Figure B.3 Different quantile predictions for the next state components x, y, v_x, v_y, ω_x, and ω_y of the Lunar Lander environment

B.1.7 Inverted Pendulum

The following graph Figure B.9 presents a comparison of the quantile predictions with the true environment predictions for the different state components of the Inverted Pendulum environment, using cumulative counter values.



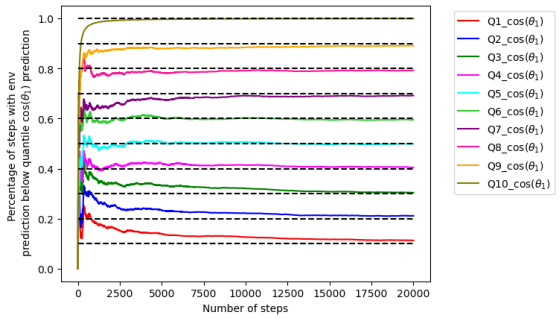
(a) Left leg is in contact variable

(b) Right leg is in contact variable

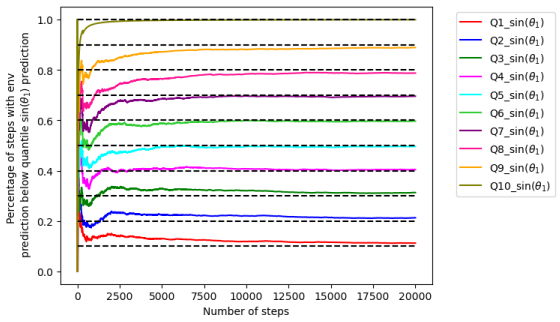
Figure B.4 Different quantile predictions for the left leg in contact and the right leg in contact next state components of the Lunar Lander environment

B.2 Deeper data analysis of control and robotics environments with continuous action spaces

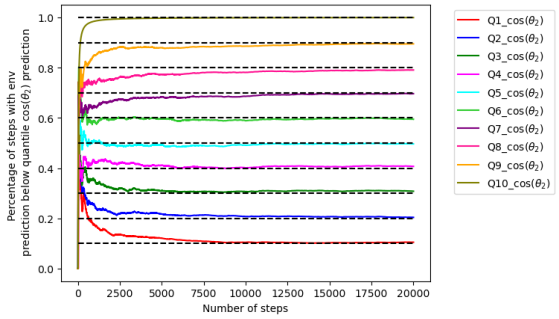
Tables B.1 and B.2 present the AUC data for the control and robotics environments with continuous action spaces.



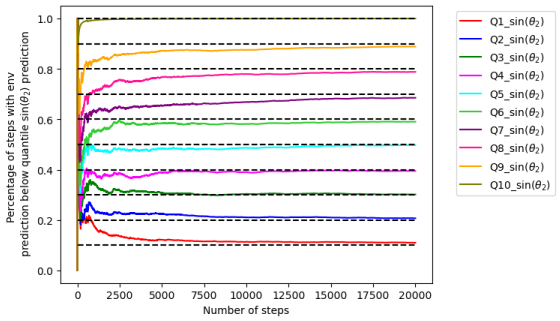
(a) $\cos(\theta_1)$



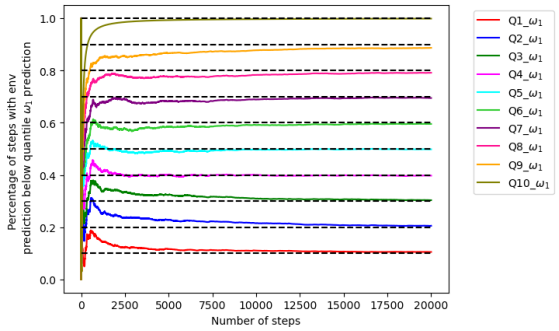
(b) $\sin(\theta_1)$



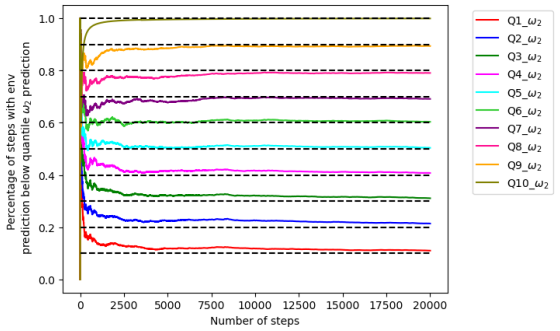
(c) $\cos(\theta_2)$



(d) $\sin(\theta_2)$

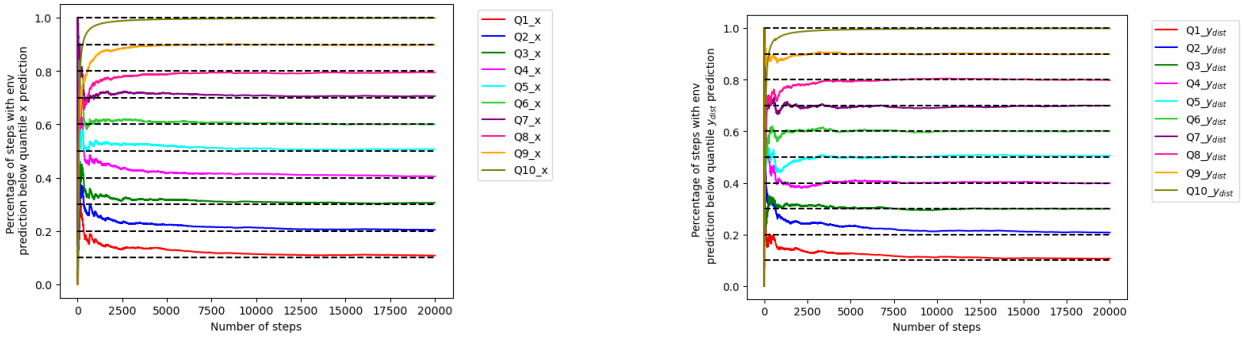


(e) ω_1



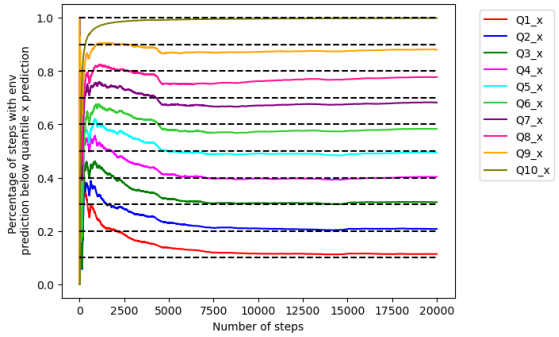
(f) ω_2

Figure B.5 Different quantile predictions for the next state components $\cos(\theta_1)$, $\sin(\theta_1)$, $\cos(\theta_2)$, $\sin(\theta_2)$, ω_1, and ω_2 of the MujoCo Reacher environment

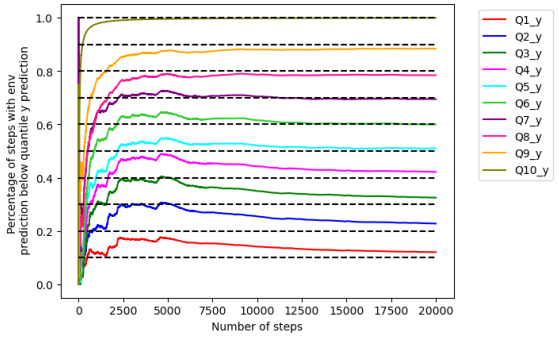


(a) x_{dist} (b) y_{dist}

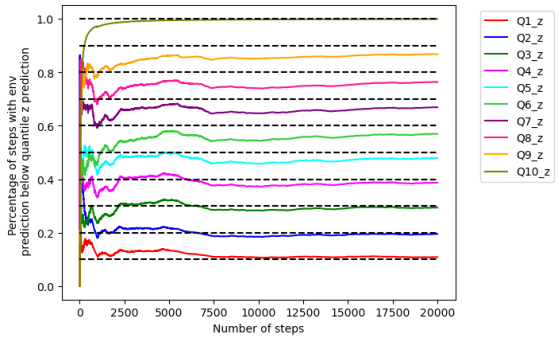
Figure B.6 Different quantile predictions for $\cos(\theta_1)$ and $\sin(\theta_1)$ next state components of the MuJoCo Reacher environment



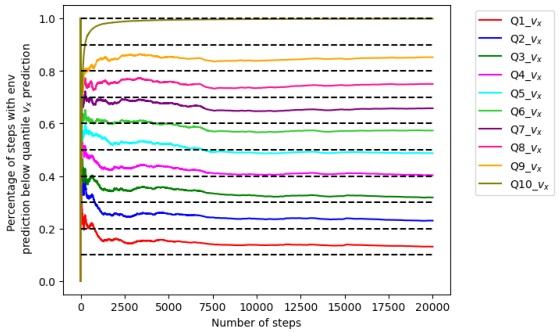
(a) x



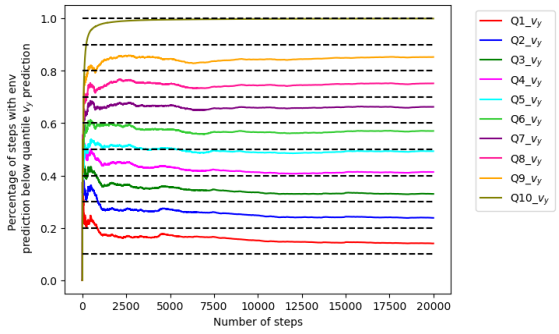
(b) y



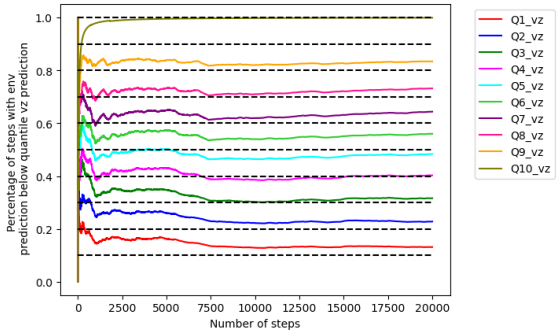
(c) z



(d) v_x

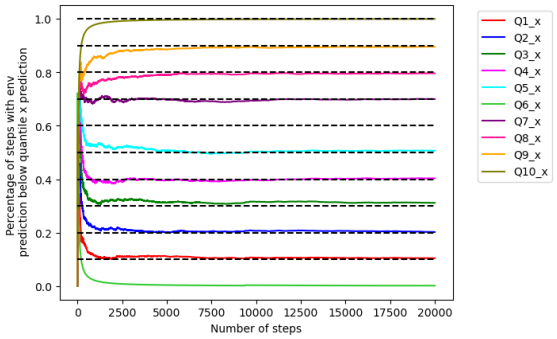


(e) v_y

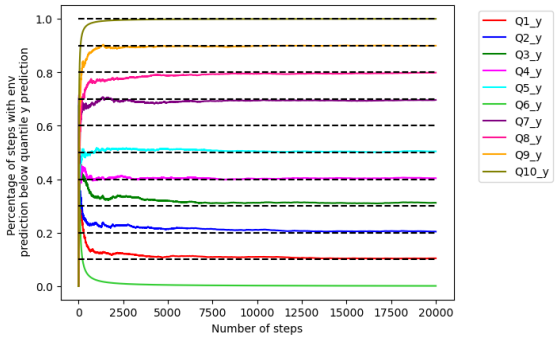


(f) v_z

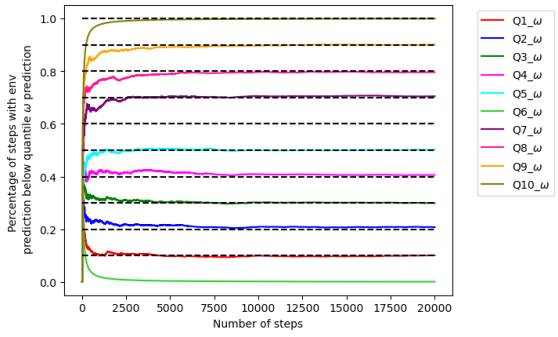
Figure B.7 Different quantile predictions for the next state components x, v, z, v_x, v_y, and v_z of the Panda Reach environment



(a) $x = \cos(\theta)$

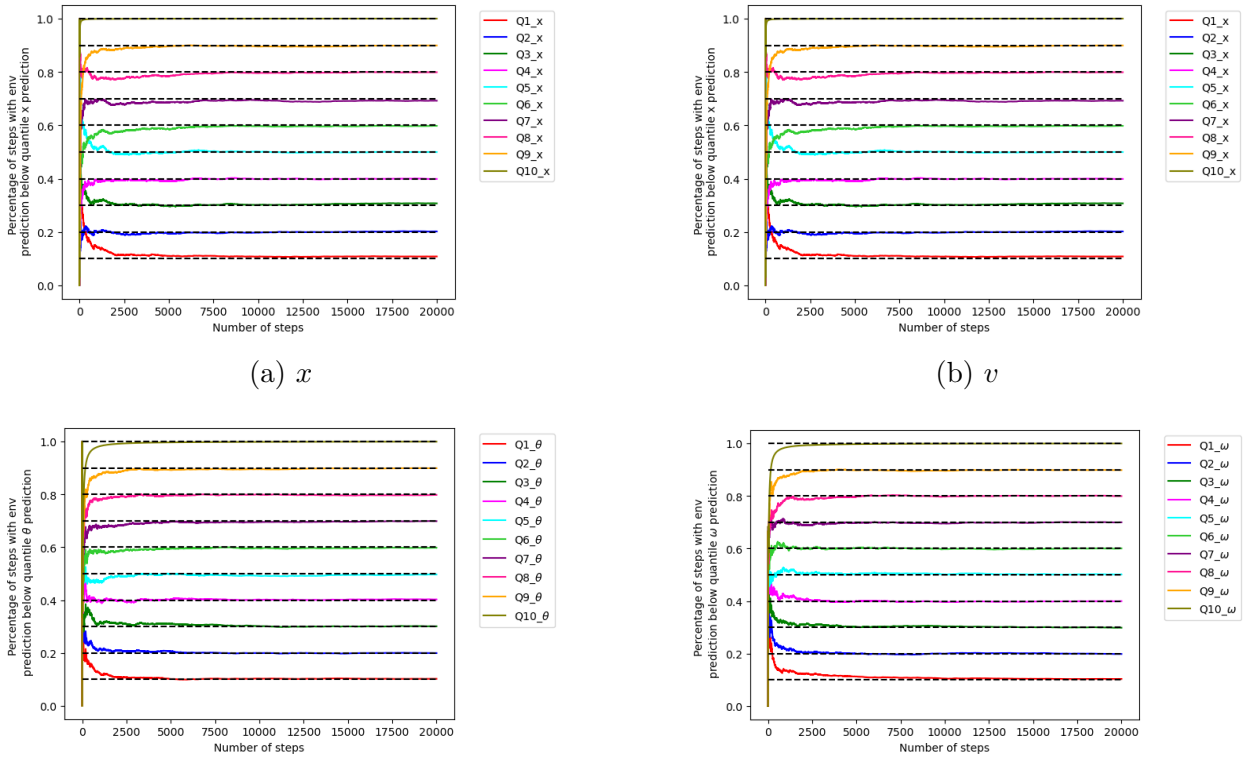


(b) $y = \sin(\theta)$



(c) ω

Figure B.8 Different quantile predictions for the next state components $x = \cos(\theta)$, $y = \sin(\theta)$ and ω of the Pendulum environment



(a) x (b) v

(c) θ (d) ω

Figure B.9 Different quantile predictions for the next state components x, v, θ, and ω of the Inverted Pendulum environment

Table B.1: Average normalized area under the curve of the episodic return for control environments with continuous spaces

Model	Average normalized AUC
QRNN-basic-PF	0.8416 ± 0.0319
50NN-ASNN-PF	0.8397 ± 0.0555
MSENN-basic-PF	0.8287 ± 0.0579
MSENN-ASNN-PF	0.8245 ± 0.0182
QRNN-ASNN-PF	0.8048 ± 0.0373
50NN-rnd-CEM	0.8001 ± 0.1361
50NN-rnd-PF	0.7963 ± 0.0611
QRNN-rnd-PF	0.7947 ± 0.0386
QRNN-rnd-CEM	0.7677 ± 0.0491
MSENN-rnd-PF	0.7664 ± 0.0528
QRNN-RS	0.7384 ± 0.0225
50NN-RS	0.7382 ± 0.0271
MSENN-RS	0.7317 ± 0.0184
MSENN-rnd-CEM	0.7043 ± 0.1320
TQC	0.6683 ± 0.0309
SAC	0.6436 ± 0.0447
iCEM	0.5308 ± 0.0331
GP-MPC	0.5083 ± 0.0095
50NN-basic-PF	0.4807 ± 0.0416
PPO	0.4740 ± 0.1092
MPPI	0.4448 ± 0.0283
DDPG	0.4072 ± 0.0174
CEM	0.4043 ± 0.0897
QRNN-basic-CEM	0.4039 ± 0.0278
TD3	0.3846 ± 0.0365
50NN-ASNN-CEM	0.3814 ± 0.0485
QRNN-ASNN-CEM	0.3744 ± 0.0273
MSENN-basic-CEM	0.3713 ± 0.0586
MSENN-ASNN-CEM	0.3303 ± 0.0647
PETS-CEM	0.3240 ± 0.0928
A2C	0.3154 ± 0.0933
50NN-basic-CEM	0.2622 ± 0.1785

Table B.2: Average normalized area under the curve of the episodic return for robotics environments with continuous spaces

Model	Average normalized AUC
QRNN-ASNN-PF	0.9722 ± 0.0300
50NN-ASNN-PF	0.9669 ± 0.0497
50NN-ASNN-CEM	0.9662 ± 0.0393
QRNN-ASNN-CEM	0.9649 ± 0.0328
MSENN-ASNN-PF	0.9494 ± 0.0588
MSENN-ASNN-CEM	0.9469 ± 0.0402
QRNN-rnd-CEM	0.9451 ± 0.0281
50NN-rnd-CEM	0.9377 ± 0.0348
QRNN-basic-CEM	0.9361 ± 0.0391
MSENN-rnd-CEM	0.9284 ± 0.0327
QRNN-basic-PF	0.9149 ± 0.0452
MSENN-basic-CEM	0.9126 ± 0.0525
QRNN-rnd-PF	0.9107 ± 0.0335
MSENN-basic-PF	0.8962 ± 0.0402
50NN-rnd-PF	0.8953 ± 0.0375
MSENN-rnd-PF	0.8868 ± 0.0324
QRNN-RS	0.8605 ± 0.0217
50NN-RS	0.8569 ± 0.0249
MSENN-RS	0.8486 ± 0.0306
MPPI	0.7880 ± 0.1759
PETS-CEM	0.6802 ± 0.2873
CEM	0.6335 ± 0.1268
DDPG	0.5861 ± 0.0192
TD3	0.5817 ± 0.0108
TQC	0.5737 ± 0.0046
SAC	0.5656 ± 0.0040
iCEM	0.4998 ± 0.2308
PPO	0.3852 ± 0.0120
50NN-basic-PF	0.3630 ± 0.0886
50NN-basic-CEM	0.3391 ± 0.0972
A2C	0.3276 ± 0.0730
GP-MPC	0.3102 ± 0.0220