UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR ROBOTIK
UND KOGNITIVE SYSTEME

An exploration scheme based on the state-action novelty in continuous state-action space

Ein auf der Zustands-Aktionen-Neuheit im kontinuierlichen Zustands-Aktionen-Raum basierendes Erkundungsschema

Bachelorarbeit

verfasst am
Institut für Robotik und Kognitive Systeme

im Rahmen des Studiengangs
Robotik und Autonome Systeme
der Universität zu Lübeck

vorgelegt von
Phillip Johann Overlöper

ausgegeben und betreut von
Prof. Dr. Elmar Rückert

mit Unterstützung von
Honghu Xue, M.Sc.

Lübeck, den 15. Oktober 2020

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Phillip Johann Overlöper

Zusammenfassung

Die Erkundung im schrittbasierten Verstärkungslernen ist ein herausforderndes und offenes Problem. Falls es in einem kontinuierlichen Suchraum angewendet wird, kann eine naive Erkundungsstrategie dazu führen, dass nur in der Nachbarschaft des initialien Zustandes erkundet wird und aus diesem Grund ein großer Anteil des gesamten Raumes unerkundet bleibt. Das nur einmalige Besuchen eines Zustands führt allerdings zu schlechter Leistung, wobei der Verstärkungslernalgorithmus in einem lokalen Minimum stecken bleibt. Diese Arbeit präsentiert ein neuartiges Erkundungsschema für kontinuierliches Zustands-Aktions-Raum-Verstärkungslernen, welches auf der Neuheit von Zustands-Aktionen-Paaren basiert. Die Neuheit eines solchen Paares wird anhand der Dichte der komprimierten Zustands-Aktionen-Paare bestimmt. Weiterhin stellt diese Arbeit eine Methode vor, die eine Aktion interpoliert, um eine glatte Trajektorie in einem Markow-Entscheidungsprozess zu ermöglichen, und auf einen Roboter angewandt werden kann. Dieses Experiment wurde in der Simulationssoftware CoppeliaSim anhand eines Roboters mit sieben Freiheitsgraden durchgeführt. Die Ergebnisse zeigen, dass dieser neue Ansatz eine effektivere Erkundung ermöglicht, als eine Baseline-Erkundung.

Abstract

Exploration in step-based reinforcement learning is a challenging and open problem. If it is applied in a continuous search space, the naive exploration strategy could result in an explored space which is only explored in the neighbourhood of an initial state, leaving a vast amount of entire space unexplored. Visiting states only once leads to poor performance, where the reinforcement learning algorithm gets stuck in a local minimum. This thesis presents a novel exploration scheme for continuous state-action space reinforcement learning, based on the novelty of state-action pairs, where the novelty is measured via the density of the compressed state-action pair. Furthermore, this thesis presents a method to interpolate the action to reach a smooth trajectory in a Markov Decision Process, which can be applied to any robot. The experiment was performed in the CoppeliaSim simulator on a robot with seven degrees of freedom. The results of the new approach show a more effective exploration than the baseline exploration.

Contents

1

Introduction

This chapter will give an overview of the challenges regarding exploration in reinforcement learning and thus motivation for this thesis. It also presents an outlook for this thesis.

1.1 Motivation and Challenges

One of the key challenges of reinforcement learning is the Exploration-Exploitation-Trade-Off. The agent needs to decide at each decision point, whether to choose an action of which the agent knows that it brings it closer to what is expected (exploitation) or risk exploring the environment more to find a better solution. Therefore, the performance of reinforcement learning algorithms is highly dependent on to what extent the agent has explored the environment to avoid convergence in a local optimum. Furthermore, the application of efficient exploration is not restricted to reinforcement learning, but also to a wider spectrum of problems, e.g., in mobile robotics where the agent needs to explore the environment mostly to generate a complete map. In this thesis, we focus on the setting of Markov Decision Process, i.e., a sequential decision process, where the selection of a new action is solely based on the current state.

For discrete state-action spaces several exploration strategies exist, e.g. the ϵ-greedy algorithm. Although they have the nice property that if the exploration is done several times, every state-action pair can be visited sufficiently to determine which action is optimal. However, they cannot be extended to continuous cases. In continuous state-action spaces it is necessary to have extensive knowledge about the environment without exploring every option, since it would not be feasible to do so.

One popular exploration strategy in continuous state-action spaces is stochastic sampling, where an action is randomly sampled from a density distribution, e.g. a normal distribution. However, these strategies explore the environment in an uninformative manner, which often results in a space that is only extensively explored in the neighborhood of some initial state, leaving a vast amount of the entire space unexplored. Hence there is a motivation to develop better exploration strategies for continuous state-action

spaces.

The goal of this thesis is to present a novel exploration strategy, which enables an efficient exploration in a continuous state-action space and belongs to the state-novelty category of the different exploration strategies. The idea is to choose the next action based on the novelty of the state-action pair. The novelty of the compressed representation of the state-action pair is then evaluated through its density.

The contribution of this thesis is to present a novel exploration strategy, which has two different variants. These variants are compared to each other, but also to a baseline exploration to evaluate the efficiency of the strategy. This evaluation is done based on a robot with seven degrees of freedom, which was simulated with the help of the simulation software CoppeliaSim. Additionally, this thesis presents a way to interpolate trajectories to ensure a smooth trajectory with respect to (angular joint) velocity profile and a continuous acceleration profile.

1.2 Outlook

The thesis is structured as follows: The second **Chapter 2** gives an overview of the background knowledge, which is required for this thesis. The third **Chapter 3** presents other exploration strategies and different examples of movement primitives. The fourth **Chapter 4** introduces the two variants of the novel exploration strategy proposed by this thesis as well as the interpolation method. The fifth **Chapter 5** presents an evaluation of the results of the novel exploration strategy. The sixth **Chapter 6** is the conclusion of this thesis.

2

Preliminaries

This chapter covers the background knowledge that is required for this thesis. Firstly the concept of reinforcement learning (RL) and the Markov decision process (MDP) are presented. The next section covers dimensionality reduction and the two reduction approaches, which are used in this thesis – Principal Component Analysis (PCA) and Autoencoders – in detail. The following section covers optimization problems and the black box optimizer Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES). The last two chapters cover movement primitives and the baseline approach which is used in this thesis as a comparison for the novel exploration strategy.

2.1 Markov Decision Process

The Markov decision process (MDP) is specified as a sequence of states and actions which adhere to the Markov property. A process has the Markov property, when the future of the process only depends on the present and not the history. The MDP is mathematically defined as a five-tuple $M = (\mathbb{S}, \mathbb{A}, T, R, p_0)$, where:

- $\mathbb{S}$ is the set of states
- $\mathbb{A}$ is the set of actions
- T is the transition probability function, such that $T(\mathbf{s}, \mathbf{s}', \mathbf{a}) = p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, where state $\mathbf{s}$ and successor state $\mathbf{s}' \in \mathbb{S}$ and action $\mathbf{a} \in \mathbb{A}$
- R is the reward function, which assigns a reward $r \in \mathbb{R}^1$ to every transition caused by action $\mathbf{a}$
- p_0 is the initial distribution, which tells how likely it is to start in a certain state

2.2 Reinforcement Learning

Reinforcement learning [(Sutton and Barto, 2018)] is a field of machine learning, in which an agent shall learn a policy π to maximize a reward. The process is unsupervised, meaning there are no labels or teaching errors given and the solution is learned autonomously and potentially from scratch. The environment in which the agent acts is

usually formulated as a MDP. The environment has a set of states $\mathbb{S}$and the agent chooses from a set of actions $\mathbb{A}$ in each state and hence gets an intermediate reward r. The selection, which action is chosen at which state, is called policy and can be modeled as:

- $\pi : \mathbb{A} \times \mathbb{S} \mapsto [0, 1]$
- $\pi(\mathbf{s}, \mathbf{a}) = p(\mathbf{a}|\mathbf{s})$

The goal now is to find a policy π, which maximizes the sum of all rewards. In value based reinforcement learning it is necessary to have knowledge of the accumulative reward starting at any state $\mathbf{s}$. This reward is determined by the value function V_π. It returns the expected accumulative reward, when policy π was followed. It is defined as shown in Equation 2.1:

$$V^\pi(\mathbf{s}) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | \mathbf{s}_0 = \mathbf{s}], \tag{2.1}$$

where the discount factor $\gamma \in [0, 1]$, such that the effect of future states is counted less importantly. The maximum possible value of V^π is defined as is shown in Equation 2.2:

$$V^*(\mathbf{s}) = \max_\pi V^\pi(\mathbf{s}) \tag{2.2}$$

When V^* has a solution, it is the optimal solution for the problem. However, it can be useful to also consider the action values. The function which does this, is defined as shown in Equation 2.3:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}] \tag{2.3}$$

The value $Q^\pi(\mathbf{s}, \mathbf{a})$ returns the accumulative reward, starting at state $\mathbf{s}$ and choosing action $\mathbf{a}$.

The goal now is to find a policy π, which chooses the probability $\pi(\mathbf{s}, \mathbf{a})$ according to an action $\mathbf{a}$ in state $\mathbf{s}$. Under the condition that π^* is the optimal policy, optimality can be achieved, when at each state $\mathbf{s}$ the optimal action is chosen from $Q^{\pi^*}(\mathbf{s}, \cdot)$.

2.3 Dimensionality Reduction

Dimensionality reduction is the process of transforming a given data set from a high-dimensional space into a low-dimensional space. This transformation has to be executed in such a way that the low dimensional representation of the dataset still retains the key properties of the original high-dimensional data for further process. There exist several ways of how to reduce the dimensions of a given data set. Two of them are discussed in detail in this thesis: Principal Component Analysis and dimensionality reduction through Autoencoders. Both these approaches are explained more thoroughly in the following sections.

PCA

The Principal Component Analysis (PCA) [(Wold, Esbensen, and Geladi, 1987; Abdi and Williams, 2010)] is a method to reduce the dimensionality of a dataset by decorrelating its data and increasing its variance. It does this by transforming the data to so called principal components. The principal components are then ordered in a way in which the first few of them that have the greatest eigenvalues hold the basic information of the original data set and account for the most variance.

The first step is to compute the covariance matrix of the entire dataset to examine whether there are any relations between different variables. The equation to calculate the covariance between to random variables X and Y is shown in Equation 2.4:

$$Cov(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mathbb{E}(X))(y_i - \mathbb{E}(Y)), \tag{2.4}$$

where n denotes the size of each random variable and $x_i \in X$ and $y_i \in Y$. $\mathbb{E}(X)$ and $\mathbb{E}(Y)$ are referring to the means of X and Y, respectively. The covariance matrix for a 3-dimensional dataset would look like this:

$$\begin{pmatrix} Cov(X,X) & Cov(X,Y) & Cov(X,Z) \\ Cov(Y,X) & Cov(Y,Y) & Cov(Y,Z) \\ Cov(Z,X) & Cov(Z,Y) & Cov(Z,Z) \end{pmatrix}$$

Since the covariance of a variable with itself is its variance, the diagonal of the matrix consists of the variance of each random variable. And because the covariance is commutative, the matrix is mirrored at the diagonal.

The second step in the PCA is to calculate the eigenvectors and eigenvalues of the covariance matrix. This is done to ascertain the principal components. The equation for calculating the eigenvalues of a matrix is shown in Equation 2.5:

$$det(\mathbf{A} - \lambda\mathbf{I}) = 0, \tag{2.5}$$

where $\mathbf{A}$ denotes the matrix, $\mathbf{I}$ the identity matrix and λ the eigenvalues to be calculated. Thus in the case of the 3-dimensional covariance matrix, the calculation is shown as follows:

$$Cov\begin{pmatrix} Cov(X,X) - \lambda & Cov(X,Y) & Cov(X,Z) \\ Cov(Y,X) & Cov(Y,Y) - \lambda & Cov(Y,Z) \\ Cov(Z,X) & Cov(Z,Y) & Cov(Z,Z) - \lambda \end{pmatrix} = 0$$

For each eigenvalue λ there is a corresponding eigenvector v. They can be calculated with the equation shown in Equation 2.6:

$$(\mathbf{A} - \lambda_i\mathbf{I})v_i = 0 \tag{2.6}$$

where $i = 1, ..., m$ and m denotes the number of eigenvalues.

Since the goal is to reduce the dimensionality of the dataset, the next step is to sort the eigenvectors (principal components) and only keep the $k \leq m$ of them, which account for the most variance. Because the eigenvectors simply indicate the direction of the new axes, the sorting is done based on the corresponding eigenvalues. Hence the k eigenvectors are chosen, which possess the highest eigenvalues. These are then used to form a new matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$, where d denotes the dimension of the original data, i.e. the amount of different labels.

The last step is to multiply this matrix $\mathbf{W}$ with the original dataset to project it into the lower dimensional subspace. The equation for this is shown in Equation 2.7:

$$\mathbf{S} = \mathbf{O} \times \mathbf{W}, \tag{2.7}$$

where $\mathbf{S}$ denotes the projection in the subspace and $\mathbf{O} \in \mathbb{R}^{n \times d}$ the original dataset. The variable n denotes the number of samples per label.

One additional potential step is to whiten the data to give all data points the same variance. This is done by dividing every dimension (column) of the matrix $\mathbf{W}$ by the square root of its corresponding eigenvalue.

Autoencoder

The special kind of neural networks used in this thesis is called autoencoder [(Baldi, 2012)] and they are wildly used in unsupervised learning. Their purpose is to reduce the dimensionality of a dataset while keeping the recreational loss minimal. They differ from PCA in the way they transform the data. While PCA performs only linear transformations, autoencoders perform nonlinear transformations, where the non-linearity is a result of the non-linear activation function in the neural network.

An autoencoder is usually comprised of two main parts: The encoding part and the decoding part. Since autoencoders are normally symmetric in nature, the decoder has the same amount of layers as the encoder. It incorporates the hidden layers and the output layer (see Figure 2.1).

The idea is that the encoder subsequently reduces the dimensionality of the input through several different layers to a compressed representation of the input. Afterwards the decoder increases the dimensionality of the now compressed input, until the input and the output have the same dimension. Given that procedure, an autoencoder is classified as an unsupervised learning model.

The two parts of an autoencoder can be mathematically represented as follows:

The encoder is a function $f : X \mapsto H$ that maps the n-dimensional input array $X \in \mathbb{R}^n$ to the compressed representation $H \in \mathbb{R}^m$, such that $m < n$.

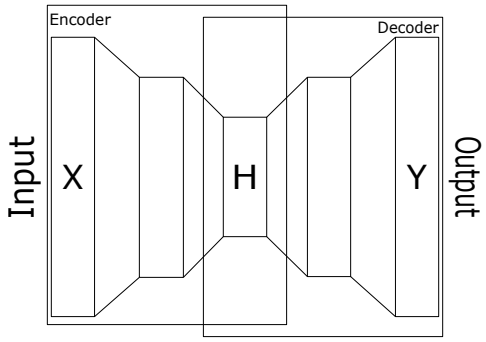


Figure 2.1: This figure shows the Autoencoder architecture. Here X refers to the input, Y to the output and H stands for the compressed representation. X and Y have the same dimensionality.

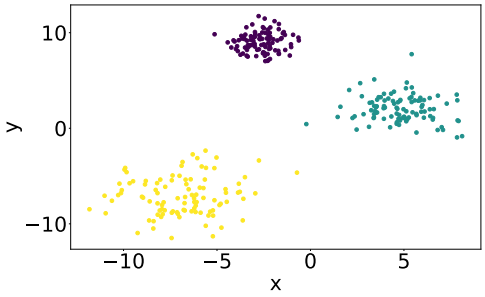
The decoder is a function $g : H \mapsto Y$ that maps the m-dimensional compressed representation $H \in \mathbb{R}^m$ to the reconstructed data $Y \in \mathbb{R}^n$.

The goal of an autoencoder is to choose f and g, such that $f, g = \underset{f,g}{\operatorname{argmin}} ||X - g(f(X))||_2$.

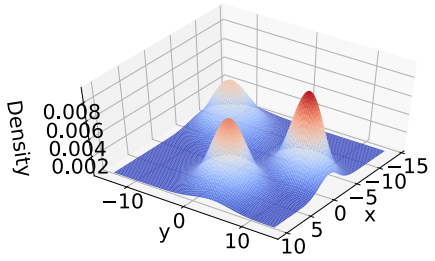
Special kinds of autoencoders are widely used in the field of image recognition, classification and denoising. Several prominent implementations exist, like the convolutional [(Chen et al., 2017)] or variational [(Kusner, Paige, and Hernández-Lobato, 2017)] autoencoder, which differ in how they encode the given input.

2.4 Density Estimation

Density estimation is a way to estimate the probability density function of given data. The probability density function describes the relative likelihood that a specific sample (here one dimension of the reduced state-action-pair) occurs anywhere in the search radius. An example of that is the normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, where μ is the mean and σ^2 the variance. The probability that a state occurs that is less than one σ^2 away from μ is 68.26 percent. Now the exploration strategy does not have to choose the next state with the lowest probability, but with the lowest probability density. However, this estimation has to be computed for every dimension of the state.
An example of how a density estimation looks like is shown in Figure 2.2. The distribution of the data is shown in Figure 2.2 a) and its density estimation is shown in Figure 2.2 b).



a) Distribution of three different clusters.



b) The corresponding density estimation of the distribution.

Figure 2.2: Distribution of data points of three different clusters and their density estimation. The means and standard deviations are $((2.5, 10), 1)$, $((6, -9), 2)$ and $((5, 0), 1.125)$.

2.5 Optimization Problems

An optimization problem [(Boyd, Boyd, and Vandenberghe, 2004)] is defined as the problem of finding the best solution from all available solutions. The best solution is evaluated using an objective, which can be represented as a scalar. It can be mathematically formulated as shown in Equation 2.8:

$$x^* = \underset{x}{argmin} f_0(x) \text{subject to} f_i(x^*) \leq b_i, i = 1, ..., m. \tag{2.8}$$

Here $f_0 : \mathbb{R}^n \mapsto \mathbb{R}$ is called the objective function and $f_i : \mathbb{R}^n \mapsto \mathbb{R}, i = 1, ..., m$ the constraint function. The constants $b_1, ..., b_m$ refer to the constraints or limits. A vector x^* is then optimal, when it has the smallest objective value (see Equation 2.8.1), while satisfying all constraints (see Equation 2.8.2).

Evolutionary Strategies

An evolution strategy is a technique, which is primarily used for optimization problems. As the name suggests, this approach tries to find the optimal solution through the methods of evolution, mutation and selection. Mutation refers to the process of changing the genes of a genome, for example changing bits in a bitstring, where a single bit is a gene and the bitstring itself is the genome. Selection refers to the process of selecting a certain amount of individuals (each with their own genome) of a population, based on a criterion. For example, selecting the five bitstrings out of a total amount of ten bitstrings, which have the greatest sum. There exist several implementations differing in what they mutate and their selection criteria. However, the general process is as follows: the evolutionary strategy generates a set of candidate solution which it analyzes on the basis of a fitness or an objective function. The proposed solutions, which yield the best fitness values are then used to generate the next generation of candidate solutions. This process will not cease until a predefined criteria has been met.

One kind of evolutionary strategy proposes new candidate solutions by randomly sampling from a multivariate normal distributions with mean μ and a fixed covariance matrix Σ. In each generation the current mean is updated based on the best candidates from the previous generation. However, because the covariance matrix Σ is fixed and with hence the search space, one shortcoming is that whenever Σ is inadequately chosen, the convergence speed can be slow for Σ is too small. Or even worse, the optimization process gets stuck in a local optimum.

The Covariance Matrix Adaptation Evolution Strategy Algorithm

The **C**ovariance **M**atrix **A**daption **E**volution **S**trategy (CMA-ES) [(Hansen, Müller, and Koumoutsakos, 2003) and (Hansen, 2016)] is a special kind of evolution strategy that overcomes this issue, since it updates not only the mean μ every generation but also the covariance matrix Σ. This modification allows for a larger search space in the beginning and thus a fast convergence to the optimum and a smaller search space towards the end for finetuning the found optimum.

The general procedure of the CMA-ES can be represented as shown in Algorithm 1:

Algorithm 1 The general Procedure of the CMA-ES

1: Create multivariate normal distribution $X \sim \mathcal{N}(\mu, \Sigma)$ (The initial values are usually $\mu_0 = 0$ and $\Sigma_0 = I$)
2: **for** i = 1,...,I (I := Number of iterations) **do**
3: Sample N points from X, such that $Y = (y_1, ..., y_N)$ with $y_i \in X \; \forall \; i = 1, ..., N$
4: Evaluate all samples from Y with a previously defined fitness function f, such that $F = (f(y_1), ..., f(y_N)) \; \forall \; y_i \in Y$
5: From F choose the M samples with the best fitness value (i.e. the highest or the lowest) and calculate the new mean μ and the new covariance matrix Σ
6: **end for**

The process ends, when the termination criteria has been met. The best solution from all iterations is then chosen by the algorithm.

2.6 Movement Primitives

The idea of movement primitives is that a policy, which could posses a high complexity, due to for example a dynamically changing environment, could be learned from a combination of simpler policies (i.e. movement primitives).

A policy π is formulated as a function that maps the (continuous) state vector $\mathbf{x}$ to a continuous control vector $\mathbf{u}$, as shown in Equation 2.9:

$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t), \tag{2.9}$$

where t denotes the possibility that $\mathbf{x}$ is time dependent and α denotes adjustable parameters specific to the problem. However, this policy can also be represented as a combination of simpler policies as shown in Equation 2.10:

$$\mathbf{u} = \pi(\mathbf{x}, \alpha, t) = \sum_{k=1}^{K} \pi_k(\mathbf{x}, \alpha_k, t), \tag{2.10}$$

where K denotes the number of simple movement primitives.

2.7 Exploration by Stochastic Sampling in Continuous State-Action Space

To evaluate the efficiency of the novel exploration strategy presented in this thesis, it is necessary to compare it to an already existing exploration strategy. In this work a baseline approach was chosen.
One popular baseline for exploration in continuous space involves sampling from a normal distribution $X \sim \mathcal{N}(\mu, \sigma)$. This is the simplest method computation wise and also the quickest overall [(Lillicrap et al., 2015)]. The pseudo-code for the algorithm is shown in Algorithm 2. Firstly the search radius and the constraint are initialized (see Algorithm 2 line 1). Then a random value is drawn from the normal distribution, with mean μ and variance σ, at every decision step. This process is repeated until the action satisfies the constraint (see Algorithm 2 line 7). Afterwards the action is performed (see Algorithm line 2 9, the interpolation is done internally). The exact specifications can be seen in Algorithm 2.

Algorithm 2 Baseline Exploration Strategy via Stochastic Sampling

1: Initialize: Action Constraint C, Search Radius σ
2: Start Environment
3: **for** i = 0,...,N (N := Number of episodes) **do**
4: $S_0 \leftarrow$ ResetEnvironment()
5: **for** j = 0,...,M (M := Number of decision points) **do**
6: **repeat**
7: Sample $a_j \sim \mathcal{N}(0, \sigma)$
8: **until** a satisfies C
9: $s_j \leftarrow$ Environment.step(s_j,a_j) (Interpolation done internally)
10: **end for**
11: **end for**

3

Related Work

This chapter presents work, which is connected to the novel exploration approach that is presented in this thesis. Section 3.1 presents research for the exploration in a continuous state-action space for four different categories: exploration through a stochastic policy, exploration through prediction error, exploration through information gain and exploration through state novelty. The research, which is presented in this thesis belongs to the latter category. Section 3.2 presents research for enabling smooth trajectories in continuous state spaces.

3.1 Exploration Strategies in continuous State-Action Space

This thesis proposes a novel exploration strategy for continuous state-action space based on state novelty. There are however several other approaches to achieve exploration in a continuous state-action space. These can be grouped in four main categories: stochastic policy, prediction error, information gain and state novelty.

Exploration through a Stochastic Policy

The methods, which belong to the category of exploration through stochasticity present the actions, which can be taken, as a distribution, typically a Normal distribution. This distribution tells the probability of taking an action **a** given a state **s**. This is called a policy. At each decision step an action is then sampled from this probability distribution $\mathbf{a} \sim f_\omega$, where f_ω denotes the probability distribution with the distribution parameters ω. In real world systems, the distribution is always bounded due to physical constraints, which can introduce a bias.

The authors [(Chou, Maturana, and Scherer, 2017)] presented a new stochastic policy for continuous reinforcement learning, which was based on the Beta policy and compared it to a Gaussian policy. They found that their proposed policy was, in contrast to the Gaussian policy, bias free and outperformed it when both were used with Trust Region Policy Optimization (TRPO) [(Schulman et al., 2015)] and Actor Critic with Experience Replay (ACER) [(Wang et al., 2016)].

Exploration through the Prediction Error

The methods, which belong to the category of exploration through prediction error, generate an intrinsic reward based on how accurate an agent predicts the outcomes of its actions. Mathematically, this reward can be represented as the distance between the real and predicted state as shown in Equation 3.1:

$$R(\mathbf{s}, \mathbf{s}') = ||g(\mathbf{s}') - F(g(\mathbf{s}), \mathbf{a})||_2, \tag{3.1}$$

where g is a function, which represents the real next state and F a function that represents the predicted next state.

This however can be lead to many problems given the stochastic nature of the agent's environment and the agent's actuators inherent noise.

The authors [(Pathak et al., 2017)] managed to avoid many problems with previous prediction approaches. They only predicted the changes in the environment of the agent that could be caused by its actions. They observed that the agent manages to move around the corridors and halls of the game *VidDoom* without any extrinsic reward. In the game *Mario* however, the agent only manages to complete around 30% of the first level due to a specific sequence of button presses that the agent has to do. They concluded that their methods performed well when the environment has only few reward signals, but did not extend to environments that have few interaction opportunities as well.

The authors [(Stadie, Levine, and Abbeel, 2015)] presented a method, which assigned exploration bonuses that were based on a learned model of the system dynamics. The method learns the state representation of the environment from observations, trains dynamics model from this representation and then uses the prediction error from this model to asses the novelty of a state. The idea is that novel states yield a greater prediction error than states which were already visited, since they were not used for training the model. Exploration bonuses were assigned based on how great the prediction error was. They found that their method achieves large improvements over other methods in a range of Atari games.

Exploration through the Information Gain

The methods, which belong to the category of exploration through the information gain, generate an intrinsic reward based on the information gain that an action yields. One potential method of computing the information gain is to calculate the change of entropy of the action. Mathematically this idea can be formulated as shown in Equation 3.2 :

$$R(\mathbf{s}, \mathbf{s}') = U'(\theta) - U(\theta), \tag{3.2}$$

where the function U refers to the current entropy, U' to the entropy after an action has been performed and θ to the parameter set of the agent.

The authors [(Stachniss, Grisetti, and Burgard, 2005)] proposed a gain-based exploration that uses Rao-Blackwellized Particle Filters [(Grisetti, Stachniss, and Burgard, 2007)]. The mapping of the agent's environment is done by the Filter. The decision of what action to choose next is done based on the action that reduces the expected uncertainty the most. This is done by calculating the change of entropy of the Rao-Blackwellized Filter for the execution of that action. By doing that, they found that they obtained a "robust decision-theoretic exploration algorithm that produces highly accurate grid maps."[1]

Exploration through the State Novelty

The methods, which belong to the category of exploration through state novelty, generate an intrinsic reward on the novelty of the state that an action brings. In discrete state spaces this can be easily done by keeping records of the already explored states and thus knowing the probability of each state. Mathematically this idea can be formulated as shown in Equation 3.3:

$$R(\mathbf{s}) = \frac{1}{N(\mathbf{s})}, \tag{3.3}$$

where the function N represents the number of times state $\mathbf{s}$ has been visited. Unfortunately this approach can not be extended to continuous state spaces, since the probability of any state would always be zero, due to the high dimensionality of the data. The solution to this are so called pseudo counts as shown in Equation 3.4:

$$N(\mathbf{s}) = \hat{N}(\mathbf{s}) = \frac{\rho(\mathbf{s})(1 - \rho'(\mathbf{s}))}{\rho'(\mathbf{s}) - \rho(\mathbf{s})}, \tag{3.4}$$

where ρ is the density model, which returns the probability of observing state $\mathbf{s}$ and ρ' the probability of observing $\mathbf{s}$ after one more pass of $\mathbf{s}$.
The authors [(Bellemare et al., 2016)] proposed pseudo-counts that were directly derived from the density model. Based on this pseudo-count, they generated an exploration bonus for a Deep-Q-Network-agent in continuous state discrete action space. The agent managed to achieve state of the art exploration performance in the game *Montezuma's Revenge* on the Atari 2600 when combined with a mixed Monte Carlo update. [(Ostrovski et al., 2017)] examined how important the quality of the density model is. They compared a pseudo-count derived from the CTS density model [(Bellemare, Veness, and Talvitie, 2014)] and a pseudo-count derived from the PixelCNN density model [(Oord, Kalchbrenner, and Kavukcuoglu, 2016)]. They concluded that the PixelCNN density model produces better samples. For the role of the Monte Carlo update they found that it sped up training time but hurt performance.

[1] (Stachniss, Grisetti, and Burgard, 2005), page 8, line 15 f.

3.2 Characterization of Robot Arm Trajectories

Since this thesis is focusing on finding an exploration strategy for robots, it is required to have a property, which guarantees that the new action, which is chosen by the exploration strategy (in this case a new velocity), does not harm the actuators of the robot. This is why it is necessary to achieve a smooth transition from the old action to the new. One approach to achieve smooth exploration is to define policies as movement primitives. Several implementations of movement primitives exist.
The authors [(Schaal, 2006)] have presented the notion of Dynamic Movement Primitives, where units of action are formulated as stable nonlinear attractor systems. The attractor here refers to the desired kinematic state, i.e., position, velocity and acceleration, instead of direct motor commands.
Thus complex tasks can be represented as a combination of multiple MPs. In [(Paraschos et al., 2013)] certain criteria a MP has to meet are formulated, so that different MPs can be combined to a complex task. These criteria are blending between motions, co-activating several MPs simultaneously and adapting to modified task variables. To achieve that they presented Probabilistic Movement Primitives (ProMP), which are formulated as a density distribution over trajectories. This enables the formulation of described criteria as operations from probability theory. They concluded that this approach is promising for "learning, modulating and re-using movements in a modular control architecture."[2]
The idea of blended trajectory in ProMP is closely related to this work, where this idea was extended to step-based MDP settings but still ensured the smoothness of the trajectory at each decision step.

[2] (Paraschos et al., 2013), page 8, line 23f.

4

State-Action Novelty-Based Exploration in MDP

The first Section explains, why it is necessary to enable a smooth trajectory in a continuous state-action space between two decision points and how this is done, when reinforcement learning itself only works with discrete state-action spaces. The second section presents the two variants of the novel explorations approach, which are presented in this thesis, autoencoder and PCA.

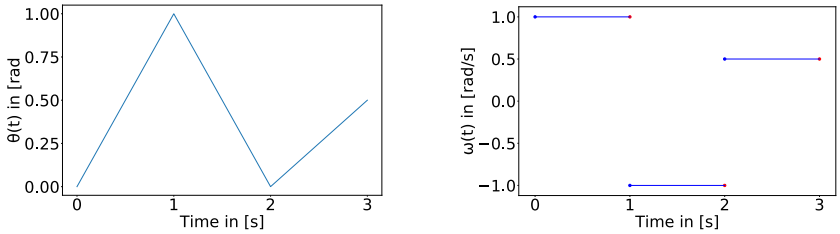
4.1 Trajectory Blending between two Decision Steps

In reinforcement learning a new action is chosen at each decision step. Both for discrete and continuous state and action spaces. The problem is that this approach does not consider a smooth trajectory, i.e. what happens in between the decision steps. Within the simulation environment an abrupt change of velocity is not a problem. However, in a real robot application this can severely harm its mechanical parts. The resulting non-smooth trajectory can oftentimes lead to oscillation. Especially in a real robot task the performance of the robot arm hardware can be severely impacted by these oscillations. One potential solution to this is to allow actions of small amplitude for each decision step. However, they are also non smooth at each decision step, as is shown in Figure 4.1. Therefore, one more elegant way is trying to retain the properties of smoothness and is based on the idea of trajectory blending in movement primitives.
In this thesis a new action is chosen at every decision step. This action is a vector of velocities of each joint. Therefore it is nice to enable a smooth transition from the current action to the new action, i.e. extend a smooth trajectory to step-based MDP settings, where the velocity is still smooth for a complete episode.
The equation, which was used to achieve this, is shown in Formula 4.1. Its derivative is shown in Equation 4.2:

$$V_i(\text{t}) = (1 - \sin((\text{t-floor(t)})t * \frac{\pi}{2})^3) * V_c + \sin((\text{t-floor(t)}) * \frac{\pi}{2})^3 * V_n \tag{4.1}$$

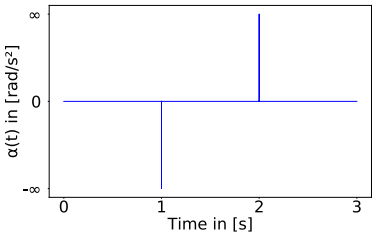
$$A_i((\text{t}) = -\frac{3}{2}\pi \cos((\text{t-floor(t)}) * \frac{\pi}{2}) \sin((\text{t-floor(t)}) * \frac{\pi}{2})^2 (V_c - V_n), \tag{4.2}$$

where V_i and A_i denote the current interpolated velocity and acceleration respectively and V_c denotes the current velocity and V_n the new velocity. The parameter t refers to the current time.

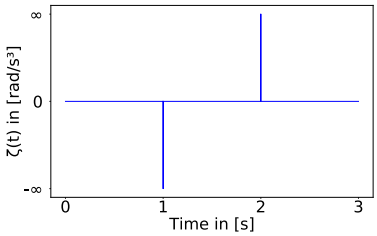


a) Joint Position Profile

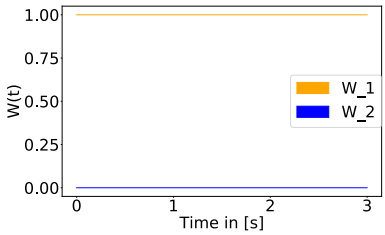
b) Joint Velocity Profile



c) Joint Acceleration Profile



d) Joint Jerk Profile

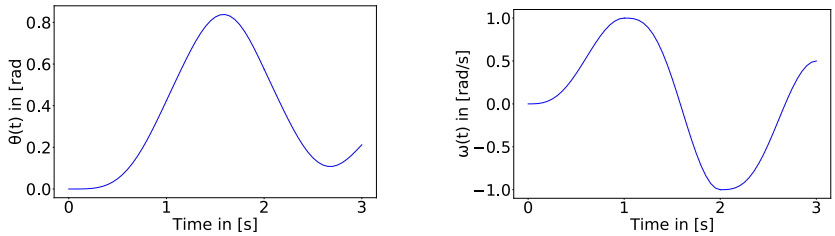


e) Weights

Figure 4.1: The plots of the position a), velocity b), acceleration c) and jerk d) profile of the trajectory before smoothing for three different actions. The actions are 1 rad/s (t = 0), -1 rad/s (t = 1) and 0.5 rad/s (t = 2). Plot d) shows the weights "W_1" and "W_2" of the two blended trajectories, which were used. The non-smoothness is clearly observable for the position a), velocity b), acceleration c) and jerk d) profile at each time step (t = 1, 2, 3). The non-continuity is also observable in b) and c) and d).

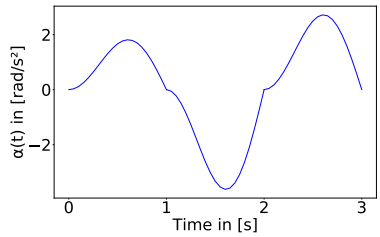
Figure 4.1 shows the profiles of position 4.1 a), velocity 4.1 b), acceleration 4.1 c) and jerk 4.1 d) as well as the weights for a trajectory 4.1 e) before smoothing. We can see that choosing an action at time step t leads to an unsmooth and noncontinuous velocity profile (see Figure 4.1 b)). The acceleration and jerk profiles have the same properties (see Figure 4.1 c) and d)). Merely the acceleration profile is continuous, however it is not smooth (see Figure 4.1 a)). The weights, which were used to blend both action trajectories (old and new action) are shown in Figure 4.1 e). Orange symbolizes the weights for the old and blue the weight for the new action.

Figure 4.2 shows the profiles for the same actions as Figure 4.1 after smoothing. A new action is still chosen at each decision step t, however, we can see that this time it

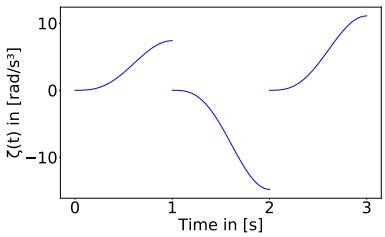


a) Joint Position Profile

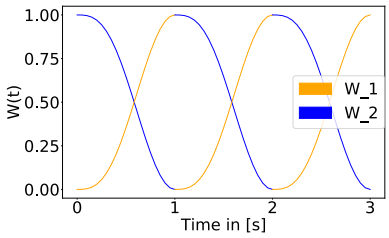
b) Joint Velocity Profile



c) Joint Acceleration Profile



d) Joint Jerk Profile



e) Weights

Figure 4.2: The plots of the position a), velocity b), acceleration c) and jerk d) profile of the trajectory after smoothing for three different actions. The actions are 1 rad/s (t = 0), -1 rad/s (t = 1) and 0.5 rad/s (t = 2). Plot e) shows the weights "W_1" and "W_2" of the two blended trajectories, which were used. We can observe continuity or the position a), velocity b), acceleration c) and jerk d) profile at each time step (t = 1, 2, 3). The position a) and velocity b) profile also show that smoothness is achieved.

leads to a continuous and smooth trajectory (see Figure 4.2 b)). The position profile exhibits the same properties (see Figure 4.2 a)). The acceleration profile is now continuous, however it is not smooth (see Figure 4.2 c)). The same is true for the jerk profile (see Figure 4.2 d)). The weights, which were used to blend both action trajectories (old and new action) are shown in Figure 4.2 e). Orange symbolizes the weights for the old and blue the weight for the new action. These weights are shown in the Equation 4.1.

4.2 Exploration by Searching for the Minimal State-Action Pair Density

This thesis presents a novel approach for exploration based on state-action pair density for continuous state-action space. Since the result of the density estimation on raw data is almost zero everywhere due to the high dimensionality of the data, the density es-

timation is performed on a representation of the data with reduced dimensionality. The first variant for compressing the data, which is used in this thesis, is an autoencoder. The second variant is PCA. The reason why the data has to be reduced in its dimensionality, is that otherwise the density would be close to zero everywhere.

The novelty of the approach presented in this thesis is twofold: firstly, the density estimation is not performed solely on the all states, but on all state-action pairs. Secondly, the density estimation itself is not used as a reward, but instead the next action is chosen with an optimizer. The utilised optimizer is the CMA-ES.

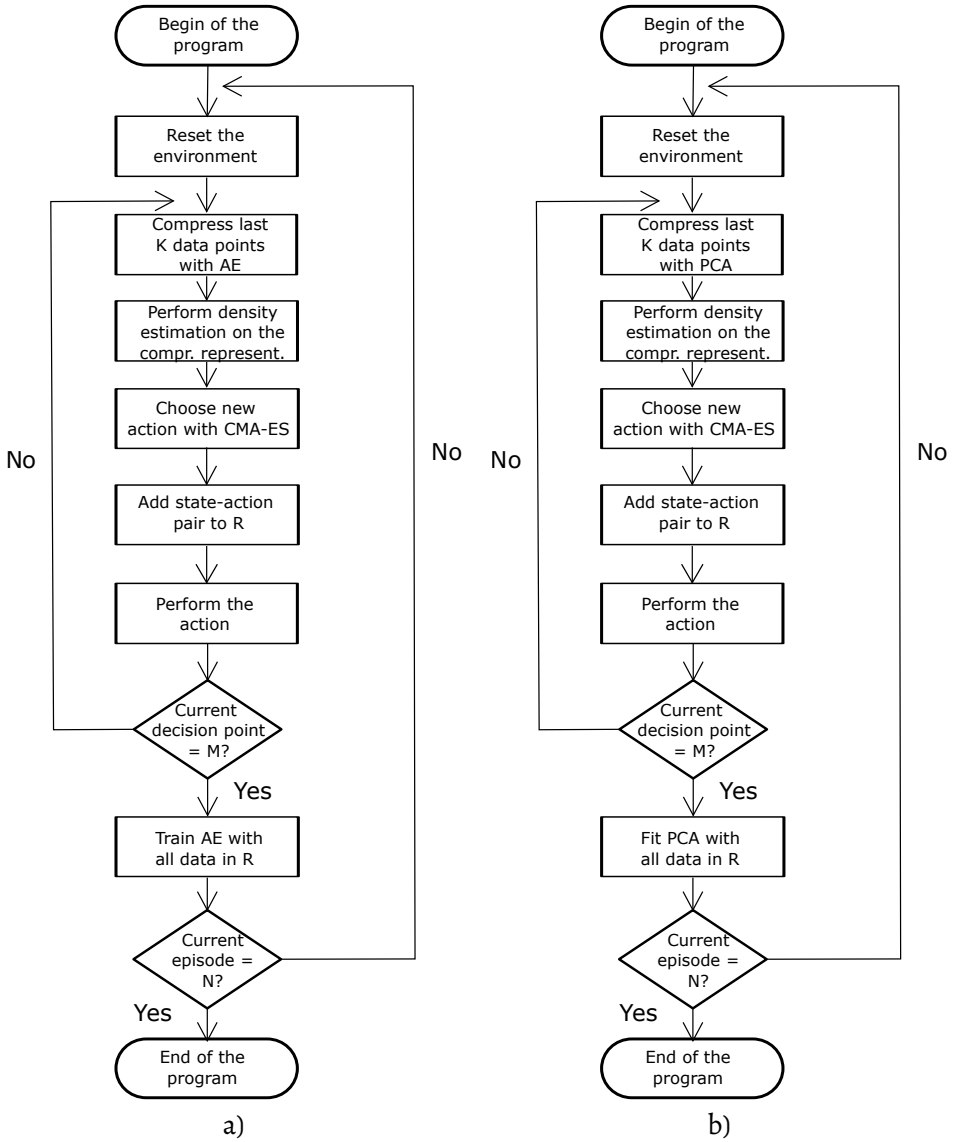


Figure 4.3: Flow Charts of the Autoencoder variant a) and the PCA variant b).

Exploration via State-Action Novelty, Variant I

The first variant of acquiring a new set of actions is through autoencoder and CMA-ES. It has already been explained how they work and what they are used for individually (see Chapter 2.3 and Chapter 2.5). However, now they are utilised in conjunction. The process itself also uses density estimation (Chapter 2.4) as an evaluation method for reinforcement learning in a continuous state-action space. The settings of the autoencoder are shown in Appendix A in Table A.3.

The flow chart of the algorithm of the autoencoder variant is shown in Figure 4.3 a) and the pseudo-code is shown in Algorithm 3. Firstly the autoencoder, the replay buffer and the constraint are set up (see Algorithm 3 line 1). The second step is the actual sampling of an action using CMA-ES. At the beginning of each decision step of the algorithm the CMA-ES is initialized with a certain mean μ and covariance matrix Σ (see Algorithm 3 line 7). Then each of the actions, which the CMA-ES has proposed, gets concatenated with the current state and propagated through the previously trained autoencoder to reduce their dimensionality and obtain their compressed representation. This compressed representation is then evaluated on the compressed representation of the state-actions-pairs of the last K episodes.[3] The action that yields the best result (lowest density) of all iterations of the CMA-ES is then chosen as the action for the current decision step (see Algorithm 3 line 8). The state-action pair is then added to the replay buffer (see Algorithm line3 9) and the action is performed until the next decision step (interpolation is done internally). When the episode is terminated, the autoencoder is trained on all available data in the replay buffer (see Algorithm line 3 12).

Algorithm 3 Exploration via state-action novelty, Variant I

1: Initialize: Replay Buffer R, Action Constraint C, Autoencoder AE
2: Start Environment
3: **for** i = 0,...,N (N := Number of Episodes) **do**
4: $S_0 \leftarrow$ ResetEnvironment()
5: **for** j = 0,...,M (M := Number of decision points) **do**
6: Perform density estimation on all K previous data points (K := Number of last episodes for density estimation)
7: Initialize CMA-ES with μ and Σ
8: $a_j \leftarrow \underset{a}{\text{argmin}}\ \text{density}(AE(s_j, a))$, where a satisfies C
9: Add (s_j, a_j) to R
10: $s_j \leftarrow$ Environment.step(s_j,a_j) (Interpolation done internally)
11: **end for**
12: Train AE on all data points (s, a) in R
13: **end for**

[3] K can be chosen as one likes. In this thesis K is fixed to 20, so the overall calculation time could be reduced. However, K can also be set to incorporate every previous episode.

Exploration via State-Action Novelty, Variant II

In the second variant of acquiring a new action, the action itself is also determined by the CMA-ES. However, the method of reducing the dimensionality of the state-action pair differs from the autoencoder approach. Here this is achieved through PCA. The settings of the PCA can be seen in Appendix A in Table A.5.

The flow chart of the algorithm of the PCA variant is shown in 4.3 b) and the pseudocode is shown in Algorithm 4. It resembles the algorithm for the autoencoder approach. Firstly, the replay buffer, the constraint and PCA are initialized (see Algorithm 4 line 1). Then, at the beginning of each decision step, PCA is performed on all L previous data points in the replay buffer (see Algorithm 4 line 7) to obtain their compressed representation. The next step is to initialize CMA-ES with the mean μ and the covariance matrix Σ (see Algorithm 4 line 8). The process of how the new action is determined is the same as in the autoencoder approach, the only difference is that the dimensionality reduction of the state-action pair is achieved through PCA (see Algorithm 4 line 9). The resulting state-action pair is added to the replay buffer (see Algorithm line 4 10) and the action is performed until the next decision step (interpolation is done internally).

Algorithm 4 Exploration via state-action novelty, Variant II

1: Initialize: Replay Buffer R, Action Constraint C, Principal Component Analysis PCA
2: Start Environment
3: **for** i = 0,...,N (N := Number of episodes **do**
4: $s_0 \leftarrow$ ResetEnvironment()
5: **for** j = 0,...,M (M := Number of decision points **do**
6: Perform density estimation on all K previous data points in (s, a) in R (K := Number of last episodes of density estimation)
7: Perform PCA on all L previous data points in (s, a) in R (L := Number of last episodes for PCA)
8: Initialize CMA-ES with μ and Σ
9: $a_j \leftarrow \underset{a}{\text{argmin}}\ \text{density}(PCA(s_j, a))$, where a satisfies C
10: Add (s_j, a_j) to R
11: $s_j \leftarrow$ Environment.step(s_j,a_j) (Interpolation done internally)
12: **end for**
13: **end for**

5

Results

The first section of this chapter introduces the performance measurements, which are used to evaluate the effectiveness of the two variants of the novel exploration strategy presented in this thesis: autoencoder 4.2 and 4.2. The second chapter applies these measurements to the state-action pairs, which were gathered from the two different variants and analyses the results and compares them to each other and the baseline approach 2.7. The third chapter presents the results of the autoencoder training.

5.1 Performance Measure for Exploration

Since the goal of this thesis can be formulated as choosing the action which minimizes the density value of the compressed representation, we expect that both the compressed representation of the autoencoder and the PCA variant have a lower mean density and are spread out more broadly than that of the baseline approach. Seeee Figure 5.1 for a visual example of how the compressed representation looks like. It shows the compressed representation of data, which was collected using the autoencoder variant and compressed using an autoencoder after 50 dimensions exemplary for the first dimension. The black cross beneath the plot show the value of the original data points (see Figure 5.5 for the compressed representation of the other approaches). To measure how broadly the compressed representation is spread out, two performance measurements are used.

The first performance measurement measures how far the compressed representation is spread out by computing its entropy. The expectation is that both the autoencoder and PCA variant have a larger entropy score than the baseline approach, since they explored more area, i.e. the compressed representation is spread out more broadly and thus has a larger information value about the environment. The entropy is computed using the following Equation 5.1:

$$H(X) = \sum_{i=1}^{n} -P(x_i)\log(P(x_i)), \text{ such that } x_i \in X \; \forall \; i = 1, ...n, \tag{5.1}$$

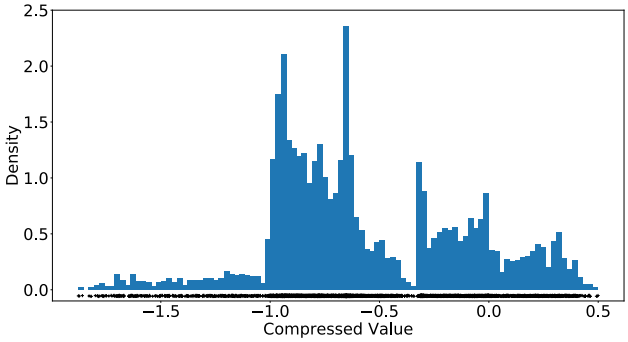


Figure 5.1: Density of the compressed representation of state-action pairs which were collected using the autoencoder variant and compressed using the autoencoder after 50 episode for the first dimension. The black crosses beneath the plot refer to the location of the individual data points.

where X denotes the entire data and n the number of samples within the data. Here the data refers to one dimension of the (five-dimensional) compressed representation. Additionally the data was discretized, so that for each episode the number of samples stays constant at 1000. The maximum entropy that can be achieved can be calculated with $log(n)$. So in this case the maximal reachable entropy is $\log(1000) = 9.966$.

The second performance measurement measures how far the compressed representation is spread out by computing the mean density of the compressed representation of the autoencoder, PCA and baseline approach. The expectation is that both the compressed representation of the autoencoder and PCA variant have a smaller mean density, since they are spread out more broadly than that of the baseline approach and therefore are not clustered around a certain value. The density estimation $\widehat{f}$ of the actual density f of any data point is done using the Kernel Density Estimation as is shown in Equation 5.2:

$$\widehat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - x_i}{h}), \text{ such that } x, x_i \in X \forall\ i = 1, ...n, \tag{5.2}$$

where X denotes the entire data, n the number of samples within the data, K the kernel and h a smoothing parameter called bandwidth. In this thesis this is done using the Kernel Density Estimation of the Scikit-learn Python Package [(Pedregosa et al., 2011)], the exact configurations are listed in Appendix A in Table A.6.

Additionally the plots of the absolute endeffector position (in Cartesian coordinates X,Y,Z) of all three approaches as well as plots of the compressed representation of the original data are shown and compared to the results of the two performance measurements mentioned above.

5.2 Effectiveness of the Exploration Strategy

Firstly we want to take a look at the results of the first performance measurement. In Table 5.2 the mean and the variant of the three different approaches after 50, 100, 150 and 200 episodes are shown. Since the state-action pairs originally had 21 dimension and were compressed to five dimensions, for each approach *five* entries are shown. The column labeled "Autoencoder" refers to state-action pairs, which were gathered using the autoencoder variant and compressed using the autoencoder. The column labeled "PCA" refers to state-action pairs, which were gathered using the PCA variant and compressed using the PCA. The columns labeled "Baseline - AE" and "Baseline - PCA" refer to state-action pairs, which were gathered using the baseline approach and compressed using the Autoencoder and PCA, respectively. The expectation was that both the PCA and autoencoder variant have a higher entropy than the baseline approach, since they should explore the available search space more efficiently. This is exactly what we can see in the Table 5.2. Throughout all dimensions and episodes we can see that both the Autoencoder and PCA variant consistently score higher mean entropy than the baseline approach. Additionally we can observe that the Autoencoder variant performed better than the PCA variant in exploring the search space, since its entropy is higher and thus the range of explored state-action pairs is greater.

Now we take a look at the results of the second performance measurement. For that we compress the state-action pairs using either the autoencoder or the PCA and forward their compressed representation to the Kernel Density Estimation. This returns the logarithmic density, which is then exponantiated to obtain the density. What is expected, is that the mean of the density of the autoencoder and PCA variant is lower than that of the baseline approach. Since the autoencoder and PCA variant compress the data differently and thus return results of a different magnitude, we can not simply compare the results directly. This is why Table 5.3 consists of two sub tables. The first table shows the mean and the variance, when the autoencoder is used as the reduction method ,i.e. autoencoder data, which has been compressed with the autoencoder (column "AE on AE"), PCA data, which has been compressed with the autoencoder (column "PCA on AE") and baseline data, which has been compressed using autoencoder (column "BL on AE"). The second table shows the mean and variance, when PCA is used as the reduction method, i.e. autoencoder data, which has been compressed with PCA (column "AE on PCA"), PCA data, which has been compressed with PCA (column "PCA on PCA") and baseline data, which has been compressed using PCA (column "BL on PCA"). Both show the mean and variance after 50, 100, 150 and 200 episodes. However, we can examine the tendency of the mean density. In both sub tables, we can observe that the autoencoder and the PCA variant have a smaller mean density than the baseline approach. This corresponds to the observations of the first performance measure shown in Table 5.2 and means that the data is spread out more in comparison to the baseline approach. Hence the exploration covered a greater area of the space. Additionally we can observe that the autoencoder has a smaller density mean than the PCA variant. This tells us that the autoencoder performed better than the PCA, which corresponds to the observations from the first performance measurement.

Now we can compare the results of the two performance measurements mentioned

Compressed Dimension	Episode	Autoencoder	PCA	Baseline - AE	Baseline - PCA
1	50	**9.15 ±0.014**	9.12 ±0.003	5.49 ±0.000	8.80 ±0.000
2	50	**9.40 ±0.007**	8.05 ±0.005	6.62 ±0,009	7.01 ±0.023
3	50	**8.93 ±0.034**	8.28 ±0.006	6.09 ±0,002	7.39 ±0.043
4	50	**8.98 ±0.139**	8.08 ±0.031	6.69 ±0.000	6.86 ±0.063
5	50	**9.52±0.000**	8.02 ±0.012	8.55 ±0.000	7.15 ±0.003
1	100	**9.31 ±0.000**	9.20 ±0.002	5.67 ±0.000	8.86 ±0.001
2	100	**9.51 ±0.014**	8.22 ±0.001	6.68 ±0.002	6.94 ±0.000
3	100	**8.90 ±0.029**	8.44 ±0.000	5.92 ±0.000	7.42 ±0.015
4	100	**9.23 ±0.016**	8.05 ±0.007	6.72 ±0.002	6.88 ±0.034
5	100	**9.55 ±0.001**	8.13 ±0.005	8.68 ±0.000	7.17 ±0.007
1	150	**9.37 ±0.004**	9.24 ±0.002	5.74 ±0.000	8.88 ±0.002
2	150	**9.46 ±0.003**	8.28 ±0.001	6.73 ±0.000	6.95 ±0.001
3	150	**8.87 ±0.004**	8.52 ±0.000	5.93 ±0.000	7.40 ±0.004
4	150	**9.06 ±0.036**	8.09 ±0.004	6.82 ±0.002	6.87 ±0.007
5	150	**9.49 ±0.015**	8.17 ±0.006	8.74 ±0.000	7.16 ±0.005
1	200	**9.32 ±0.006**	9.28 ±0.002	5.74 ±0.000	8.90 ±0.001
2	200	**9.42 ±0.014**	8.35 ±0.001	6.77 ±0.000	6.97 ±0.002
3	200	**8.86 ±0.015**	8.44 ±0.000	5.95 ±0.000	7.37 ±0.001
4	200	**9.05 ±0.023**	8.15 ±0.003	6.87 ±0.001	6.89 ±0.003
5	200	**9.45 ±0.000**	8.17 ±0.007	8.80 ±0.000	7.21 ±0.000

Table 5.2: Mean and variance of the entropy of the baseline of approach, as well as of the autoencoder and the PCA variant after 50, 100, 150 and 200 episodes each for the 1., 2., 3., 4., and 5. dimension. The "Autoencoder" column is evaluated using the autoencoder and the "PCA" column using PCA. The baseline approach is evaluated on both the autoencoder (columns "Baseline - AE") and PCA (column "Baseline - PCA"). The best (highest) entropy value per row is marked bold. Since the number of samples is constant at 1000 for all episodes, the maximum entropy is $\log(1000) = 9.966$.

above to the plots of the absolute position of the endeffector. Figure 5.4 shows the plots for the autoencoder, PCA and baseline after 50, 100, 150 and 200 episodes. We can see that the explored space of the baseline approach does not seem to significantly increase for more episodes. When we compare Figure 5.4 i) and l) we can see that the exploration is focusing on the same area of the search space, while the autoencoder and the PCA variant explore the space more broadly (compare Figure 5.4 d), h) and l)). This is confirming the observations we have made so far. The baseline approach performs worse in exploring the search space than the autoencoder and PCA variant. The same can be observed for all other joints (see Appendix B Figure B.2 for the autoencoder variant, Figure B.3 for the PCA variant and Figure B.1 for the baseline variant).

Figure 5.5 shows the plots of the density of the compressed representation after 200 episodes exemplary for the third dimension. The autoencoder data was compressed using an autoencoder (see Figure 5.5 a)), the PCA data was compressed using the PCA (see

Episode	AE on AE	PCA on AE	BL on AE
50	**0.0695 ±3.214E-05**	0.200 ±1.155E-33	0.236 ±6.784E-05
100	**0.062 ±8.509E-05**	0.196 ±0.00	0.232 ±1.240E-05
150	**0.0667 ±7.118E-05**	0.192 ±0.00	0.230 ±3.288E-06
200	**0.069 ±6.137E-05**	0.187 ±0.00	0.229 ±4.054E-06

Episode	AE on PCA	PCA on PCA	BL on PCA
50	**0.001 ±1.735E-09**	0.005 ±5.773E-19	0.035±1.861E-05
100	**0.001 ±8.919E-09**	0.003 ±4.513E-19	0.0327 ±4.940E-06
150	**0.001 ±9.991E-08**	0.003 ±3.411E-19	0.0323 ±4.783E-07
200	**0.001 ±1.444E-06**	0.003 ±2.948E-19	0.032 ±7.072E-08

Table 5.3: Mean and variance of the density of the compressed representation of the three approaches. Since two different compression methods are used, the results are of a different magnitude. The first table shows the result, when the autoencoder has been used for dimensionality reduction, i.e. autoencoder data compressed by autoencoder (column "AE on AE"), PCA data compressed by autoencoder (column "PCA on AE") and baseline data compressed by autoencoder (column "BL on AE"). The second table shows the results for the PCA dimensionality reduction, i.e. autoencoder data compressed by PCA (column "AE on PCA"), PCA data compressed by PCA (column "PCA on PCA") and baseline data compressed by PCA (column "BL on PCA"). Both sub tables show the results after 50, 100, 150 and 200 episodes. The best (lowest) result per row is marked bold.

Figure 5.5 c)) and the baseline data was compressed using both methods, autoencoder (see Figure 5.5 b)) and PCA (see Figure 5.5 d)). When we compare the compressed representation of autoencoder variant (Figure 5.5 a)) to the baseline approach compressed with the autoencoder (Figure 5.5 b)), we can see that the data of the autoencoder variant is spread out more in the available space than the baseline approach data, which results in a higher entropy and a lower mean density. The maximum density of the baseline approach data is also higher than that of the autoencoder variant data. The same observations can be made, when we compare the compressed representation of the PCA (Figure 5.5 c)) to the baseline approach compressed using PCA (Figure 5.5 d)). The PCA variant data is spread out more than the baseline approach data and has a lower maximum density, which again results in a higher mean entropy and a lower mean density. So the plots of the density of the compressed representation of the three approaches are confirming the results from performance measurement one and two.

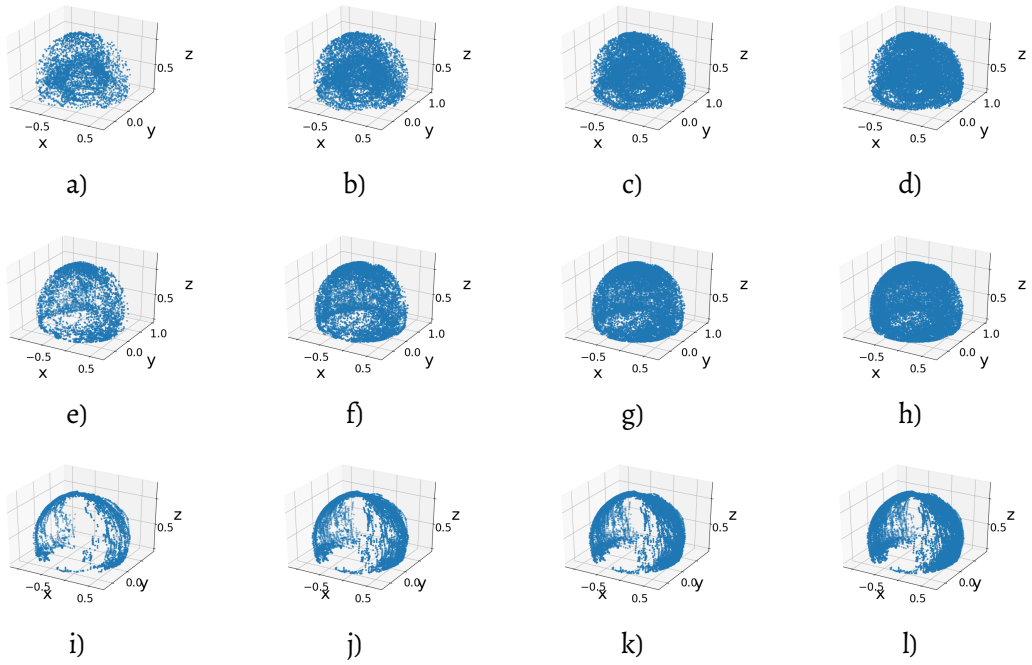


Figure 5.4: Comparison of the absolute position of the endeffector of the autoencoder and PCA variant as well as the baseline approach. The position of autoencoder variant is shown in the first row after 50 a), 100 b), 150 c) and 200 d) episodes. The position of the PCA variant is shown in the second row after 50 e), 100 e), 150 f) and after 200 g) episodes. The third row shows the position of the baseline approach after 50 i), 100 j), 150 k) and after 200 l) episodes.

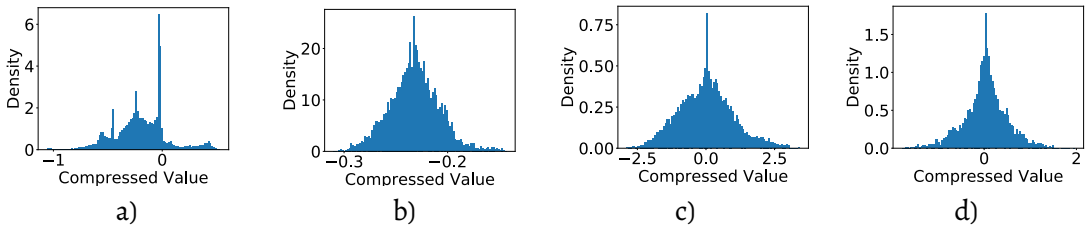


Figure 5.5: Comparison of the density of the compressed representation of the autoencoder variant, the PCA variant and the baseline approach after 200 episodes for the third dimension. The compressed representation of the autoencoder data (a)) was obtained compressing all state-action pairs gathered through the autoencoder version and compressing it with the autoencoder. The compressed representation of the PCA data (c)) was obtained compressing all state-action pairs gathered through the PCA version and compressing it with PCA. There are two plots for the compressed representation of the baseline data, one where the state-action pairs, which were gathered by the baseline approach are compressed with the autoencoder (c)) and one were it was compressed using PCA (d)).

5.3 Dimensionality Reduction

The two methods for dimensionality reduction used in this thesis are autoencoder and PCA. To ensure that the compressed representation still accurately reflects the key information of the original data, the recreational loss is computed using following Equation 5.3:

$$L(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2, \text{ for } x_i \in X \text{ and } y_i \in Y \; \forall \; i = 1, ..., n \tag{5.3}$$

where $X \in \mathbb{R}^n$ denotes the the input and $Y \in \mathbb{R}^n$ the recreated input.

The parameters, which were used for the autoencoder are listed in Appendix A in Table A.3. The autoencoder in Python was realized with the PyTorch package [(Paszke et al., 2019)]. The parameters, which were used for the PCA are listed in Appendix A in Table A.5. The PCA was realized in Python with the Scikit-Learn package [(Pedregosa et al., 2011)].

Table 5.6 shows the mean and variance of the loss of the autoencoder and PCA after 50, 100, 150 and 200 episodes. The best (lowest) value of each row is marked bold. It is observable that the recreational error is low for each episode for both variants. Hence, it can be concluded that the compressed representation of the data still accurately reflects the key information of the original data.

Episode	Autoencoder	PCA
50	0.096 ±2.304E-05	**0.0377 ±0.000**
100	0.0927 ±7.453E-06	**0.0375 ±0.000**
150	0.0983 ±5.470E-05	**0.038 ±0.000**
200	0.100 ±0.000	**0.0404 ±0.000**

Table 5.6: Mean and variance of the loss of the autoencoder (column labeled “Autoencoder”) and PCA (column labeled “PCA”) variant. The best (lowest) value per row is marked bold.

Figure 5.7 furthermore shows the mean and variance of the autoencoder learning process after different episodes. It can be seen in Figure 5.7 a) and Figure 5.7 b) that the tendency of the loss quickly decreases in the first episodes, thus showing that the learning process has been successful. In later episodes the loss is more stable and the leaning process converges (see Figure 5.7 c) and d)).

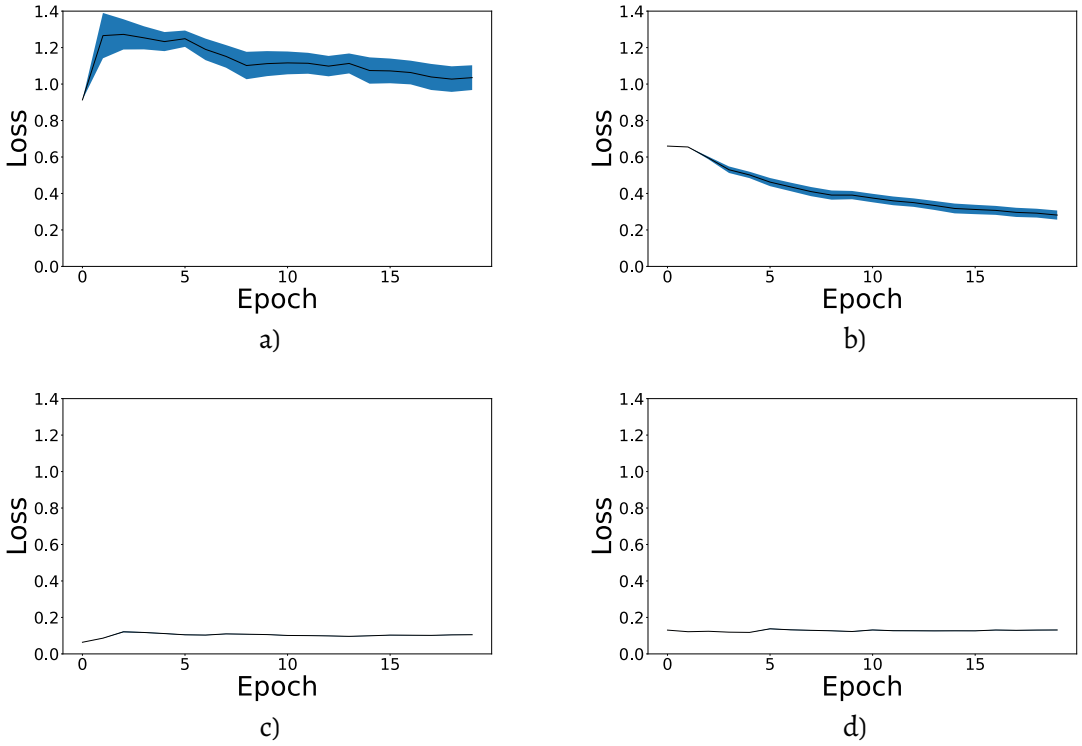


Figure 5.7: Comparison of the mean and variance of the loss of the autoencoder after 1 a), 2 b), 150 c) and 200 d) episodes.

6

Conclusion

The general problem in continuous state-action space is that traditional exploration strategies for discrete state-action space such as the ϵ-greedy algorithm cannot be directly applied. Furthermore, popular exploration strategies for continuous state-action spaces like stochastic sampling explore in an uninformative manner, i.e. it is possible that the exploration clusters around the initial state. This thesis presented a novel exploration strategy for continuous state-action space, which performs well in exploring the state-action space. The general idea is to determine the action based on the novelty of the state-action pair. The novelty of the state-action pair is evaluated using the density estimation of its compressed representation. The optimal action, which minimizes the density is then determined by an optimizer. In this thesis the CMA-ES was used.

The novel exploration approach presented in this thesis has two variants, which differ in the method of how they reduce the dimensions of the search space, namely via autoencoder and PCA. The novelty was twofold. One the one hand, instead of just encoding the state, like other exploration methods, the state-action pair was encoded. The second novelty is that the best action for each decision point is actively searched by an optimizer. The exploration itself was conducted on a simulated robot with seven degrees of freedom in the simulation program CoppeliaSim.

Two performance measurements were used to evaluate the efficiency of both variants. The results of these measurements were compared to each other as well as to a baseline approach. The first performance measurement regarded the entropy of the encoded state-action pairs. The results were as expected. Both the autoencoder and the PCA variant had a higher mean entropy score than the baseline approach, which means that their exploration range was wider than that of the baseline approach and thus their exploration was more efficient. Additionally it was observable that the autoencoder variant had a greater entropy than the PCA variant, so the autoencoder is even more efficient in the exploration. The second performance measure regarded the mean of the exponentiated density of the Kernel Density Estimation. Again the results matched the expectation that the novel exploration strategy is more effective. The autoencoder and the PCA both had a smaller mean density than the baseline approach, which means that their exploration covered a greater area than that of the baseline approach. It was also

observable that the PCA had a greater mean density than the autoencoder, which confirms again that the PCA performed worse than the autoencoder in exploration.

6.1 Potential Future Work

Future research could investigate further, which part of the novel exploration strategy presented in this thesis had which effect. For example, it could be examined, which effect the size of the data has, which was used for PCA and by the autoencoder and whether this size should be fixed or flexible. It would be also interesting to see how influential the structure of the autoencoder has been, i.e. whether an autoencoder with more layers or a different activation function could yield better results. It would also be interesting to see, which influence the CMA-ES had, i.e. whether the search process would yield better results, when the (initial) mean, the population size and the maximum number of iterations is changed.

Bibliography

Abdi, H. and Williams, L. J. (2010). Principal component analysis. Wiley interdisciplinary reviews: computational statistics *2*, 433–459.

Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In Proceedings of ICML workshop on unsupervised and transfer learning, pp. 37–49.

Bellemare, M., Veness, J., and Talvitie, E. (2014). Skip context tree switching. In International Conference on Machine Learning, pp. 1458–1466.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In Advances in neural information processing systems, pp. 1471–1479.

Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). Convex optimization. Cambridge university press.

Chen, M., Shi, X., Zhang, Y., Wu, D., and Guizani, M. (2017). Deep features learning for medical image analysis with convolutional autoencoder neural network. IEEE Transactions on Big Data.

Chou, P.-W., Maturana, D., and Scherer, S. (2017). Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In International conference on machine learning, pp. 834–843.

Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. IEEE transactions on Robotics *23*, 34–46.

Hansen, N. (2016). The CMA evolution strategy: A tutorial. arXiv preprint arXiv:1604.00772.

Hansen, N., Akimoto, Y., and Baudis, P. (2019). CMA-ES/pycma on Github. Zenodo, doi *10*.

Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary computation *11*, 1–18.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, pp. 1026–1034.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar variational autoencoder. arXiv preprint arXiv:1703.01925.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. arXiv preprint arXiv:1601.06759.

Ostrovski, G., Bellemare, M. G., Oord, A. v. d., and Munos, R. (2017). Count-based exploration with neural density models. arXiv preprint arXiv:1703.01310.

Paraschos, A., Daniel, C., Peters, J. R., and Neumann, G. (2013). Probabilistic movement primitives. In Advances in neural information processing systems, pp. 2616–2624.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems, pp. 8026–8037.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 16–17.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research *12*, 2825–2830.

Rohmer, E., Singh, S. P., and Freese, M. (2013). Coppeliasim (formerly V-REP): a versatile and scalable robot simulation framework. In Proceedings of The International Conference on Intelligent Robots and Systems (IROS),(Tokyo). Available online at: www. coppeliarobotics. com,

Schaal, S. (2006). Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. Adaptive motion of animals and machines, Springer, pp. 261–280.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In International conference on machine learning, pp. 1889–1897.

Stachniss, C., Grisetti, G., and Burgard, W. (2005). Information gain-based exploration using rao-blackwellized particle filters. In Robotics: Science and Systems, vol. 2, pp. 65–72.

Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. arXiv preprint arXiv:1507.00814.

Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and Freitas, N. de (2016). Sample efficient actor-critic with experience replay. arXiv preprint arXiv:1611.01224.

Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. Chemometrics and intelligent laboratory systems 2, 37–52.

A

Experimental Setups

This chapter presents the setup of the experiments. The first section explains which software was used and which robot was simulated. The second Section shows how the exploration task can be expressed as a RL problem. The following sections present what settings have been used for the autoencoder variant, the PCA variant, the baseline approach, the density estimation, the algorithms and the robot.

A.1 Details of the Environment

The robot which was used throughout this thesis is the KUKA LBR iiwa 7 R800 fabricated by the company KUKA AG (its settings are shown in Table A.7). To simulate the behavior of the aforementioned robot, the software CoppeliaSim [(Rohmer, Singh, and Freese, 2013)] by Coppelia Robotics was used. The initial position of the robot arm within the simulation environment is shown in Figure A.1. The code itself is written in Python.

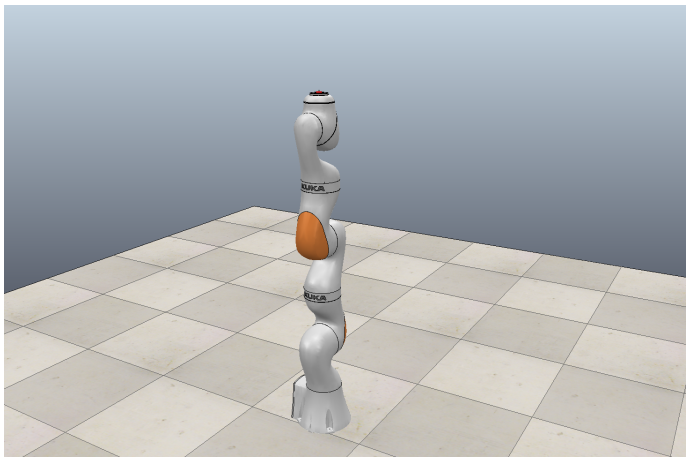


Figure A.1: The initial position of the robot arm within the simulation environment CoppeliaSim.

A.2 Task formulation as a RL problem

A state S_m is defined as $S_m = (\theta_m, \omega_m)$, where $\theta_m = (p_1, ..., p_7)$ refers to the position in rad for each joint of the robot arm and $\omega_m = (v_1, ..., v_7)$ refers to the velocity in rad/s for each joint of the robot arm while m is denoting the current decision step. The initial state S_0 is defined as $S_0 = (\theta_0, \omega_0)$. An action A_m is defined as $A_m = (a_1, ..., a_7)$ and refers to the new desired velocity in rad/s for each joint of the robot arm. Together the state S_m and A_m form the state-action pair for later compression, which is often mentioned in this thesis.

One second in the simulation consists of 20 50 millisecond steps. In this thesis this is referred to as one decision step. At the start of each decision step a new action A_t is chosen and at the end of the decision step the state velocity ω_m should be equal to the action A_m, as is shown in Figure 4.2 b). Since an entire decision step consists of 20 intermediate time steps, 18 of them have to be interpolated, based on the state velocity ω_t at the beginning of the decision step and the action A as is explained in Chapter 4.1.

A.3 Covariance Matrix Adaptation Evolution Strategy Settings

Parameters

In this thesis, a new action A_0 is chosen by the CMA-ES in the exploration variant I 4.2 and II 4.2. Since the performance of the CMA-ES can vary based on what parameters were used for the initialization, the following paragraph briefly states the parameters, which were used in the experiments in this thesis. All further information refers to the Python package *pycma* [(Hansen, Akimoto, and Baudis, 2019)]. The settings of the CMA-ES are shown in Table A.2.

Mean	Standard Deviation	Population Size	Maximum of Iterations
Current state velocity	0.25	16	50

Table A.2: Parameters used for the CMA-ES.

Objective Function

CMA-ES is supposed to choose the best action A_0, which is available at a certain decision step. The metric for evaluating the "quality" of an action is done by the objective function.

The objective function takes in all actions A_0, which were proposed in this iteration by the CMA-ES, each concatenated with the current state S_0. These state-action pairs

are then reduced in their dimensionality either by the autoencoder or by the PCA. Afterwards, a density estimation is performed on every state-action pair and evaluated on the density estimation of the compressed representation of the entire data (in the case of the autoencoder) or of the last L data points (in the case of the PCA). The action, whose state action pair yielded the lowest density is then returned. The equation for this is shown in Equation A.1:

$$f(\mathbf{s}, \mathbf{a}) = \begin{cases} \mathbf{B} + |\mathbf{a} - \mathbf{a}_c| & \text{if } \mathbf{a} > \mathbf{a}_c \\ \text{densityEstimation}(\mathbf{s}, \mathbf{a}) & \text{else} \end{cases}, \tag{A.1}$$

where $\mathbf{s}$ refers to the state, $\mathbf{a}$ to the (seven-dimensional) action and $\mathbf{a}_c$ to the action constraint. $\mathbf{B}$ is the penalty value which is assigned, when the action exceeds the constraint. In this thesis, $\mathbf{B}$ is set to 10 and $\mathbf{a}_c$ to 10 degree (both for seven dimensions).

A.4 Autoencoder Settings

A two layer network architecture was chosen. The encoder reduces the input from 21 dimensions to 16 and the second layer reduces the dimension further to 5. Since the autoencoder is symmetric, the decoder increases the dimensions subsequently from 5 to 16 to 21 dimensions. For the optimizer the Adam Optimization Algorithm [(Kingma and Ba, 2014)] was chosen (see the settings of Adam in Table A.4) and for the activation function PReLU [(He et al., 2015)] was used. The loss is calculated with the mean squared error and the learning rate is set to 0.001. This is also shown in Table A.3.

The autoencoder itself is trained at the end of each episode for 20 epochs with all the data, which has been acquired up to this point.

Layers	Nr. of Hidden Layers	Optimizer
2	16 and 5	Adam Optimization Algorithm

Act. Function	Nr. of Epochs	Loss Calculation
PReLU	20	Mean Squared Error

Table A.3: Parameters used for the Autoencoder.

Epsilon	Betas	Learning Rate
1E-08	(0.9,0.999)	0.001

Table A.4: Default settings of the Adam Optimization Algorithm.

A.5 PCA Settings

The PCA in this thesis was realized with the Scikit-learn package for Python [(Pedregosa et al., 2011)]. The only parameters changed from their standard settings were whitening, which was set on true and the number of components, which set to five (as in the autoencoder). The (standard) settings can be seen in Table A.5.

Nr. of components	Whiten	SVD Solver
5	True	Auto

Table A.5: The settings used for the PCA.

A.6 Baseline Settings

To evaluate the efficiency of the autoencoder and PCA variant, we compare it to the baseline approach, which is simply randomly sampling an action from a normal distribution $X \sim \mathcal{N}(\mu, \sigma)$. The mean μ is set to zero and the variance σ is set to $\frac{\omega}{\sqrt{2}}$, where ω refers to the joint velocity constraint, which is set to 10 deg/sec.

A.7 Density Estimation Settings

To evaluate the novelty of a new state-action pair, density estimation is performed. This is done using the Kernel Density Estimation of the Scikit-learn Python Package [(Pedregosa et al., 2011)]. The settings, which were used in this thesis are shown in Table A.6.

Kernel	Bandwidth
Gaussian	0.5

Table A.6: Kernel density estimation settings.

A.8 Algorithm Settings

In the pseudo-pseudos the parameters N, M, L and K are used. The parameter N denotes the number of episodes per experiment and is set to 200, M denotes the number of decision points per episode and is set to N, L denotes the number of previous episodes of which the data is used for fitting the PCA and is set to 50 and K is the number of previous episodes of which the data is used to fit the Kernel Density Estimator and is set to 20.

A.9 Other Settings

In this thesis the agent, which conducted the exploration was the robot Kuka LBR iiwa 7 R800 fabricated by the company KUKA AG. Its joint and velocity constraints are shown in Table A.7. However, in this thesis the velocity constraint was set to 10 deg/sec for each joint.

Joint	1	2	3	4	5	6	7
Joint constraint [deg]	170	120	170	120	170	120	175
Velocity constraint [deg/s]	98	98	100	130	140	180	180

Table A.7: Constraints of the robot Kuka LBR iiwa 7 R800.

B

Position Plots

B.1 Baseline Exploration

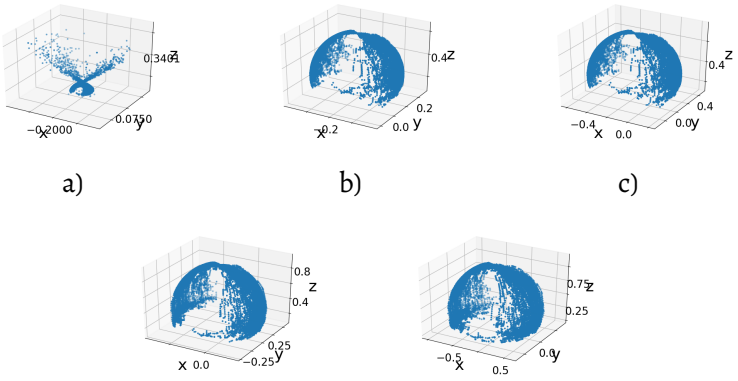


Figure B.1: Position of 2. a), 3. b), 4. c), 5. d) and 6. e) joint after 200 episodes for one run of the experiment of the baseline approach.

B.2 Autoencoder Exploration

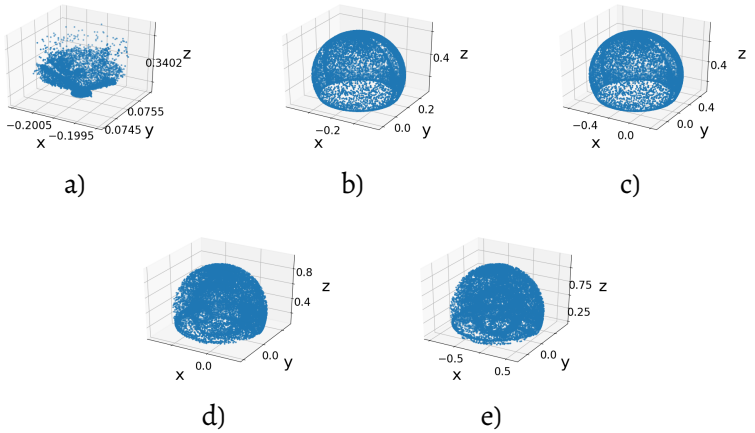


Figure B.2: Position of 2. a), 3. b), 4. c), 5. d) and 6. e) joint after 200 episodes for one run of the experiment of the autoencoder variant.

B.3 PCA Exploration

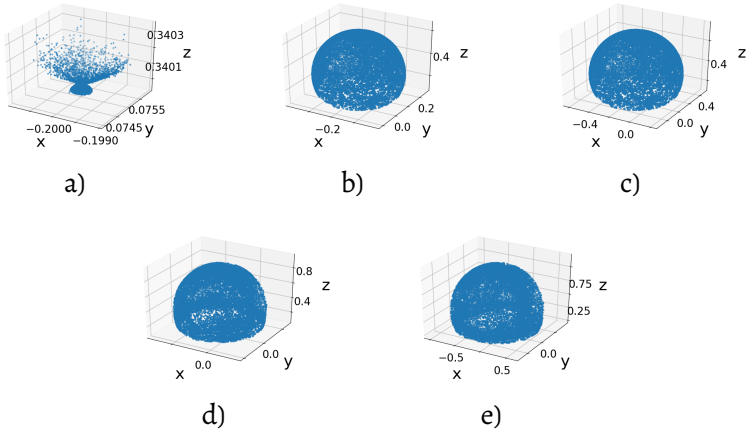


Figure B.3: Position of 2. a), 3. b), 4. c), 5. d) and 6. e) joint after 200 episodes for one run of the experiment of the PCA variant.