1 Introduction

Many machine learning tasks involve high dimensional data in order to accurately represent their complex nature. The problem is that reinforcement learning is not scaling up well to accommodate the high dimensional data. For example, an usual approach for exploring a space with help of RL would be to count the exact states that have been visited already and through that calculate their individual relative probability. Now an even exploration can be achieved, when at each state the action is chosen that yields the lowest probability. The predicament that arises in high dimensional states is that they cannot be counted. In a real space the probability for each state would be zero and that can severely limit the scope of the exploration. So the dimensionality of the state itself has to be reduced to be able to work with the data
Two common methods to reduce the dimensionality of said data are the PCA and Deep Learning, i.e. autoencoders. In this thesis we want to look at both option and see how efficiently and evenly either approach explores the working space of a robot's endeffector.

2 Related Work

3 Preliminaries

3.1 Working with the Data

The aim of this thesis is, as previously discussed, to find an optimal exploration strategy, such that the endeffector of the robot thoroughly explores its working space. That means that preferably no single state occurs more than once. We can measure that by counting how often a single state has been occupied and from that derive the state's probability of occurrence. Now whenever the exploration strategy needs to decide which state to occupy next, the available state with the lowest probability will be chosen. But that is only practicable when the state consists of whole numbers ($\mathbb{Z}$). However, since the state of the robot does not in fact consists of whole numbers – but of real numbers ($\mathbb{R}$) – we run into a problem. The probability that exactly this single state exists is infinitesimal. The result can be that the exploration will concentrate on a very tiny area within the search space, because even the smallest change of one of the state variables yields a probability of zero, for the state was not yet occupied. This is aggravated immensely by the increase of dimensions. The solutions to these kinds of problems are twofold. Firstly, not the probability of each single state is calculated but their combined density and furthermore, the dimensions of the state are being reduced to an amount that is more manageable to work with.

3.1.1 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 $X \sim \mathcal{N}(\mu, \sigma)$ (see Figure 1). The probability that a state occurs that is less then one σ away from μ is 68.26 percent.

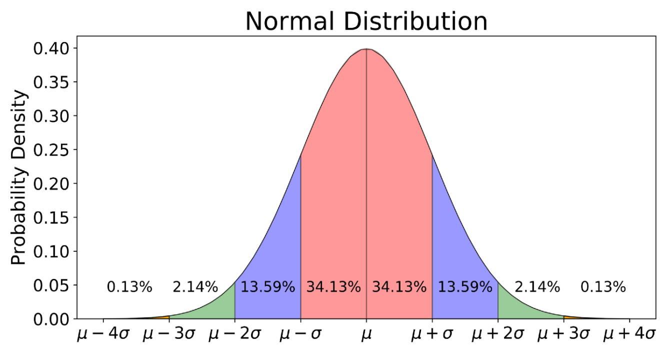


Figure 1: Normal distribution: Placeholder

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.

3.1.2 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 data still retains the key properties of the original high-dimensional data in order to be able to properly work with it. 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 a neural network called autoencoder. Both these approaches are explained more thoroughly in their dedicated chapters.

3.2 Autoencoder

3.2.1 What is an Autoencoder?

The special kind of neural networks used in this thesis is called autoencoder. Its purpose is to reduce the dimension of a given input, then increase it again and reconstruct the original information.

An autoencoder is usually comprised of two main parts: The encoding part or encoder and the decoding part or decoder. The encoder incorporates at least two layers: the input layer and one or more hidden layers. Since autoencoders are usually symmetric in nature, the decoder has the same amount of layers as the encoder. It incorporates the hidden layer(s) and the output layer (see figure ?).

The idea is that the encoder subsequently *reduces* the dimensionality of the input through several different layers to a compressed representation of the input. Following that 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.

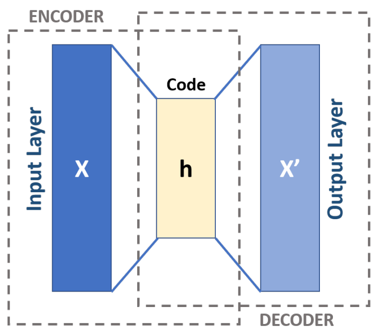


Figure 2: Simple Autoencoder: Placeholder

3.2.2 Architecture of a simple Autoencoder

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

The autoencoder itself can be represented as the concatenation of these two functions:

- Choose f and g, such that $f, g = \underset{f,g}{\operatorname{argmin}} ||X - g(f(X))$.

An autoencoder often only has just one hidden layer, but is does not have to be limited to it. More layers can bring certain advantages in some situations like reducing the computational costs[1] or decrease the amount of training data[2]. In this thesis an autoencoder with three hidden layers is used (see figure ?).

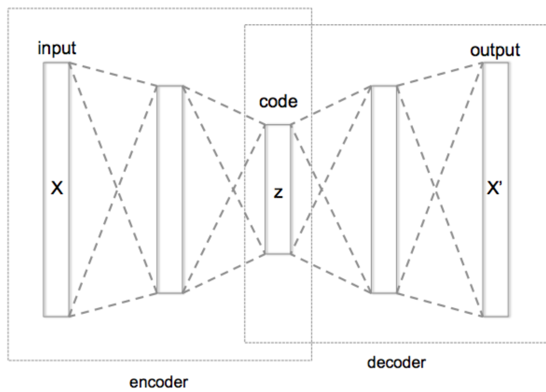


Figure 3: Simple Autoencoder: Platzhalter

3.3 Optimization Problems

3.3.1 Evolution Strategy

An evolution strategy is a technique that is primarily used for optimization problems. As the name suggests, this technique tries to find the optimal solution for a problem through the methods of evolution: Mutation and selection. Several implementations exist that differ in what they mutate and what they select. But the general process is, it generates a set of candidate solution which it analyzes on the basis of a fitness or an objective function. The proposed solutions that yield the best fitness values are then used to generate the next generation of candidate solutions. This process only ceases until a predefined criteria has been met.

3.3.2 The CMA-ES

One kind of evolution strategies proposes new candidate solutions by randomly sampling from a multivariate normal distributions with μ and a fixed Σ. Each generation the current mean is updated based on the best candidates from the previous generation. But because the Σ is fixed and with that the search radius, one shortcoming is that when Σ is inadequately chosen, the search can be rather slow for Σ is too small or even worse, the search gets stuck in a local optimum.

[1] Wikipedia:Platzhalter
[2] Wikipedia:Platzhalter

The **C**ovariance **M**atrix **A**daption **E**volution **S**trategy (CMA-ES) is a special kind of an evolution strategy that overcomes this issue, since it not only updates the mean μ every generation but also the covariance matrix $\mathbf{\Sigma}$. As is illustrated in figure ?, this modification allows for a large search radius in the beginning and thus a fast convergence to the optimum and a smaller search radius towards the end for finetuning the found optimum.

The general procedure of the CMA-ES can be presented as follows:

- Create multivariate normal distribution $X \sim \mathcal{N}(\mu, \mathbf{\Sigma})$ (The initial values are usually $\mu_0 = 0$ and $\mathbf{\Sigma}_0 = I$)
- Sample N points from X, such that $Y = (y_1, ..., y_N)$ with $y_i \in X \; \forall \; i = 1...N$
- Evaluate all samples from Y with a previously defined fitness function f, such that $F = (f(y_1), ..., f(y_N)) \; \forall \; y_i \in Y$
- 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 $\mathbf{\Sigma}$

This procedure is repeated until a termination criteria has been met, for example a certain amount of generations have passed or a certain threshold was surpassed.

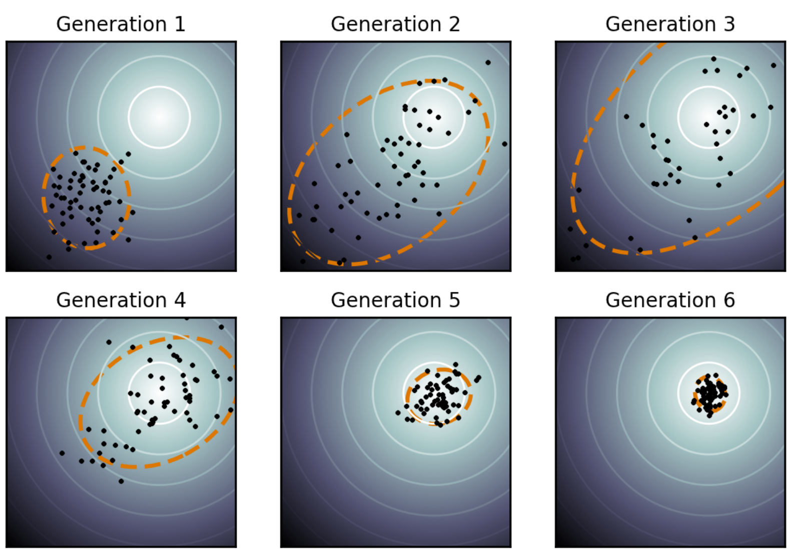


Figure 4: CMA-ES: Placeholder

4 Methods

4.1 System Setup

The robot that was used throughout this thesis is the kuka lbr iiwa 7 r800 fabricated by the company called KUKA AG. To simulate the behavior of said robot, the software CoppeliaSim by Coppelia Robotics was used. The code itself is written in Python.

4.2 Interpolating trajectory

When a new action is chosen, it is necessary to to enable a smooth transition between the current action and the new action that is supposed to be taken. In this example it means that a smooth transition between the current and the new velocity has to be found, because otherwise an abrupt change could harm the robot's mechanical parts. Since a new action is chosen every second and the simulation works in 50 ms steps, about 20 points between the current and the new velocity have to be interpolated. The formula that was used to achieve that is the following:

$$1 - sin(t * \frac{\pi}{2})^3 * currrentvelocity + sin(t * \frac{\pi}{2})^3 * newvelocity$$

Here t refers to the time that has passed within the one second interval in 50 millisecond steps, i.e. 50ms, 400ms or 850ms.

4.3 Choosing new desired Action

Three different ways of generating a new set of actions are presented here. The first method is to just simply sample a new action from a normal distribution, the second option is to calculate a new set of actions with the help of an autoencoder and the CMA-ES and the last option involves sampling through PCA.

4.3.1 Sampling from normal distribution

Sampling a new action from a normal distribution $X \sim \mathcal{N}(\mu, \sigma)$ is the simplest method computation wise and also the quickest. A random value is drawn from the normal distribution, where μ is equal to zero and σ is equal to $\frac{constraint}{\sqrt{2}}$ (see figure ?, line 11). In this thesis the constraint is referring to the angular speed of the joint and is set to ten degrees. This process is repeated for every joint, seven in this thesis (see figure ?, line 4).

After an action is sampled for each joint, it is checked whether the actions satisfy all the constraints. If even one of them is violated, the entire process is started anew (see figure ?, line 10).

Algorithm 1 Base Line

```
procedure MAIN
    EPISODES ← N
    ITERATIONS ← M
    K ← Number of joints
    constraint ← Max joint velocity
    Start environment

    for i ← 1, EPISODES do
        for j ← 1, ITERATIONS do
            while constraint is not satisfied do
                Sample new set of actions {a_{k,j}}_{k=1}^{K} with N(0, constraint/√2)
            Interpolate trajectory between {a_{k,j−1}}_{k=1}^{K} and {a_{k,j}}_{k=1}^{K}
            Reset environment
```

Figure 5: Pseudocode Baseline

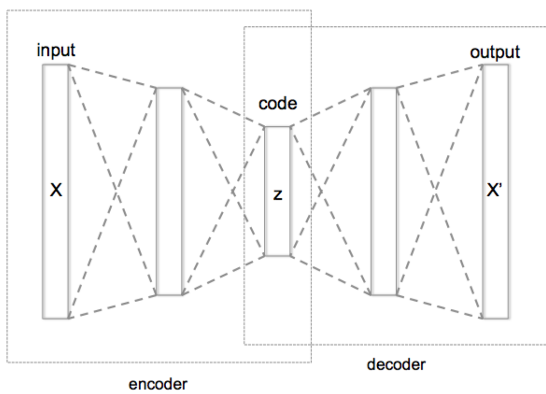


Figure 6: Autoencoder structure Placeholder

4.3.2 Sampling through an Autoencoder and the CMA-ES

Another way of sampling a new set of actions is through the CMA-ES and an autoencoder. It has already been explained how they work and what they are used for individually, but now we use them in conjunction. The process itself also uses density estimation as an evaluation method for RL.

The first part is to set up the autoencoder itself. In this thesis a two layer approach has been chosen. The first layer reduces the 21 dimensional input vector to a 16 dimensional vector. This vector is reduced further to five dimensions in the second layer (see figure ?).
For the activation function PReLU was used. After the autoencoder has been setup, it is trained after each episode with all the to this point acquired data

for thirty epochs (see figure ?,line 17).

The second step is the actual sampling of the actions with the help of the CMA-ES. In each iteration of the program the CMA-ES is initialized with the last action (the current velocity), a standard deviation of 0.25, a population size of 16 and a maximum amount of iterations of 50 (see figure ?, line 10). That means that the CMA-ES runs at max 50 iterations and that each iteration 16 sets of candidate solutions are being sampled by the algorithm. Each of those sets gets concatenated with the current state and propagated through the autoencoder to obtain the compressed representation. The compressed representation of the current state-actions-pairs are then evaluated on the compressed representation of the state-actions-pairs of the last N episodes. The set of actions that yields the best result (lowest density) is then chosen as the base for the next iteration of the CMA-ES. This is repeated until no better result can be obtained or 50 iterations are reached, The best overall result is the chosen as the new action (see figure ?, line 11).

Algorithm 2 Autoencoder

1: **procedure** MAIN
2:     *EPISODES* $\leftarrow N$
3:     *ITERATIONS* $\leftarrow M$
4:     $K \leftarrow$ *Number of joints*
5:     *constraint* $\leftarrow$ *Max joint velocity*
6:     Initialize Autoencoder
7:
8:     **for** $i \leftarrow 1,$ *EPISODES* **do**
9:         **for** $j \leftarrow 1,$ *ITERATIONS* **do**
10:             **while** *constraint not satisfied & max iteration not reached* **do**
11:                 $\triangleright$ X refers to set of candidate solution proposed by CMA-ES
12:                 $\{a_{k,j}\}_{k=1}^{K} \leftarrow \underset{x \in X}{\text{argmin}}\, \text{density}(AE(\{s_{k,j}, a_{k,x}\}_{k=1}^{K}))$
13:             Interpolate trajectory between $\{a_{k,j-1}\}_{k=1}^{K}$ and $\{a_{k,j}\}_{k=1}^{K}$
14:             Add current state and action $\{s_{k,j}, a_{k,j}\}_{k=1}^{K}$ to replay buffer R
15:             Reset environment
16:
17:         Train Autoencoder
18:         Save hidden layer data

Figure 7: Pseudocode Autoencoder

4.3.3 Sampling from PCA and CMA-ES

The second option in which density estimation is used as a evaluation method for RL and new actions are sampled with the CMA-ES, is the approach in which the input vector is compressed in its dimensionality with the help of PCA.

In each iteration the data of the last N episodes is compressed in its dimensionality using the PCA (see figure ?, line 14). The sets of actions that are proposed by the CMA-ES are also compressed using the same method. The density estimation of this compressed representation of the sets of actions is then evaluated on the density estimation of the compressed representation of the data of the last N episodes. The set of actions that yields the best result (lowest density) is then chosen as the base for the next iteration of the CMA-ES. This is then repeated, as it was in the autoencoder approach as well, until the best result has been found or 50 iterations have been surpassed (see figure ?, line 9f).

Algorithm 3 PCA

```
1:  procedure MAIN
2:      EPISODES ← N
3:      ITERATIONS ← M
4:      K ← Number of joints
5:      constraint ← Max joint velocity
6:
7:      for i ← 1, EPISODES do
8:          for j ← 1, ITERATIONS do
9:              while constraint not satisfied & max iteration not reached do
10:                 ▷ X refers to set of candidate solution proposed by CMA-ES
11:                 {a_{k,j}}_{k=1}^K ← argmin_{x∈X} density(AE({s_{k,j}, a_{k,x}}_{k=1}^K))
12:             Interpolate trajectory between {a_{k,j−1}}_{k=1}^K and {a_{k,j}}_{k=1}^K
13:             Add current state and action {s_k, a_{k,j}}_{k=1}^K to replay buffer R
14:             Fit PCA with state-action-pairs from replay buffer R
15:             Reset environment
16:
```

Figure 8: Pseudocode PCA

5 Results

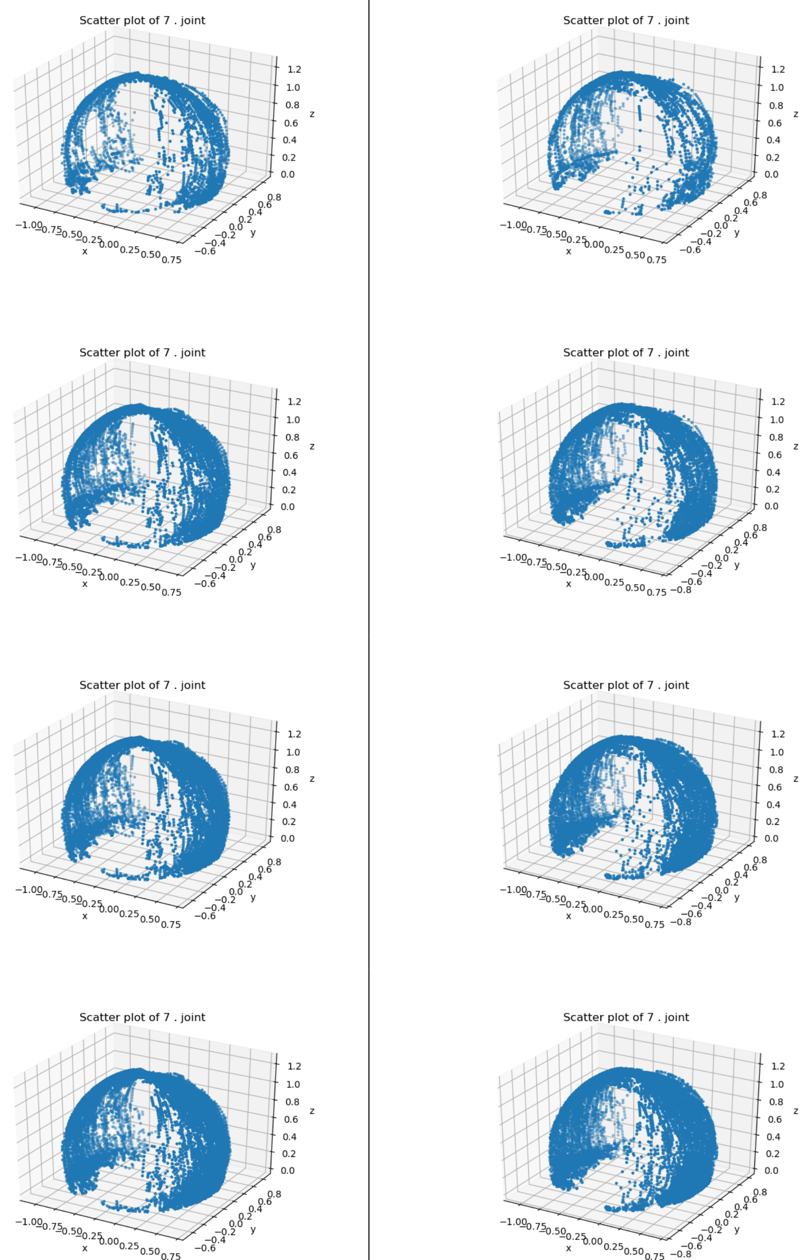
For the presentation two kinds of plots are available. Firstly the plot that shows all the positions where a specific joint has been during all iterations and all episodes in Cartesian (x,y,z) coordinates. The second shows a density estimation of the position of a joint in radian. It illustrates how often a specific angle has been occupied.
The autoencoder has an additional kind of plots. It shows the density estimation of the compressed representation of the input layer.

Only a part of the the research is shown here directly. Other plots can be found in the appendix.

5.1 Baseline Exploration

First we want to take a look on how the baseline exploration fared in exploring the working space of the robot. As a quick reminder, the next action in this approach was randomly sampled from a normal distribution. In figure ? we can see the result of that sampling method. The plots show all the positions the endeffector has occupied after 60, 120, 160 and 200 episodes for two examples in Cartesian Coordinates. We can see that only a small portion of the entire working space has been explored by the robot's endeffector. If we also compare the progression of the exploration after certain episodes, we can also observe, than the explored room itself is only shrinking slowly, since the endeffector keeps visiting the same coordinates.

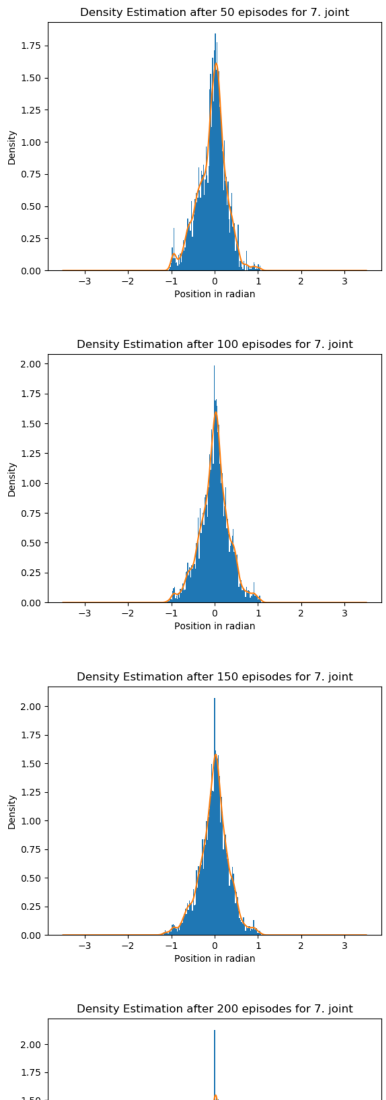
The effect of the sampling through the normal distribution can be seen in figure ? as well. It shows plots of the density of the robot's inherent endeffector position in Radians for the same two examples as in figure ?. As wee can see, the density looks similar to a normal distribution. Most of the density is centered around 0 degree (the mean of the normal distribution). Additionally we can see that the density quickly fades to zero and never reaches the constraint that the joint has (3.054326 Radian in this case). The same can be observed for all other joints as well (see Appendix).



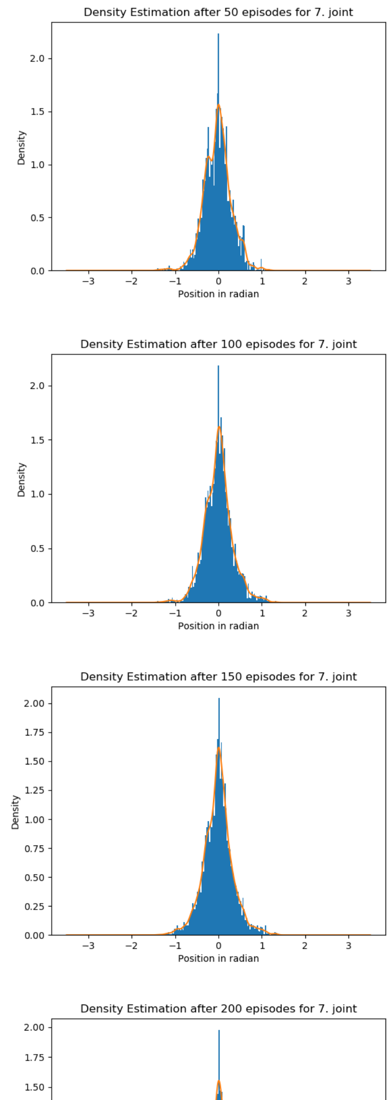
Plot of endeffector of one simulation after 60, 120, 160 and 200 episodes

Plot of endeffector of another simulation after 60, 120, 160 and 200 episodes

Figure 9: Comparison of the endeffector positions after different episodes



Plot of density of endeffector position of one simulation after 60, 120, 160 and 200 episodes



Plot of density of endeffector position of another simulation after 60, 120, 160 and 200 episodes

Figure 10: Comparison of the endeffector position density after different episodes

5.2 Autoencoder exploration

5.2.1 Training of the autoencoder

At the end of each episode the autoencoder is trained with all the to this point acquired state-action-pairs. What setting were used, you can see in the ??? chapter. From figure ? we can see that the loss is roughly between 0.05 and 0.35.

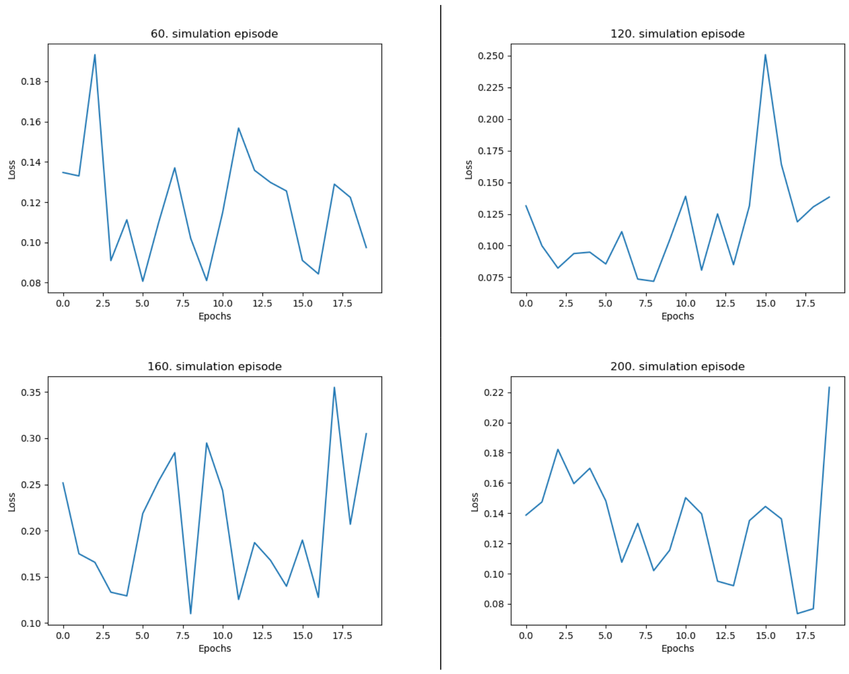


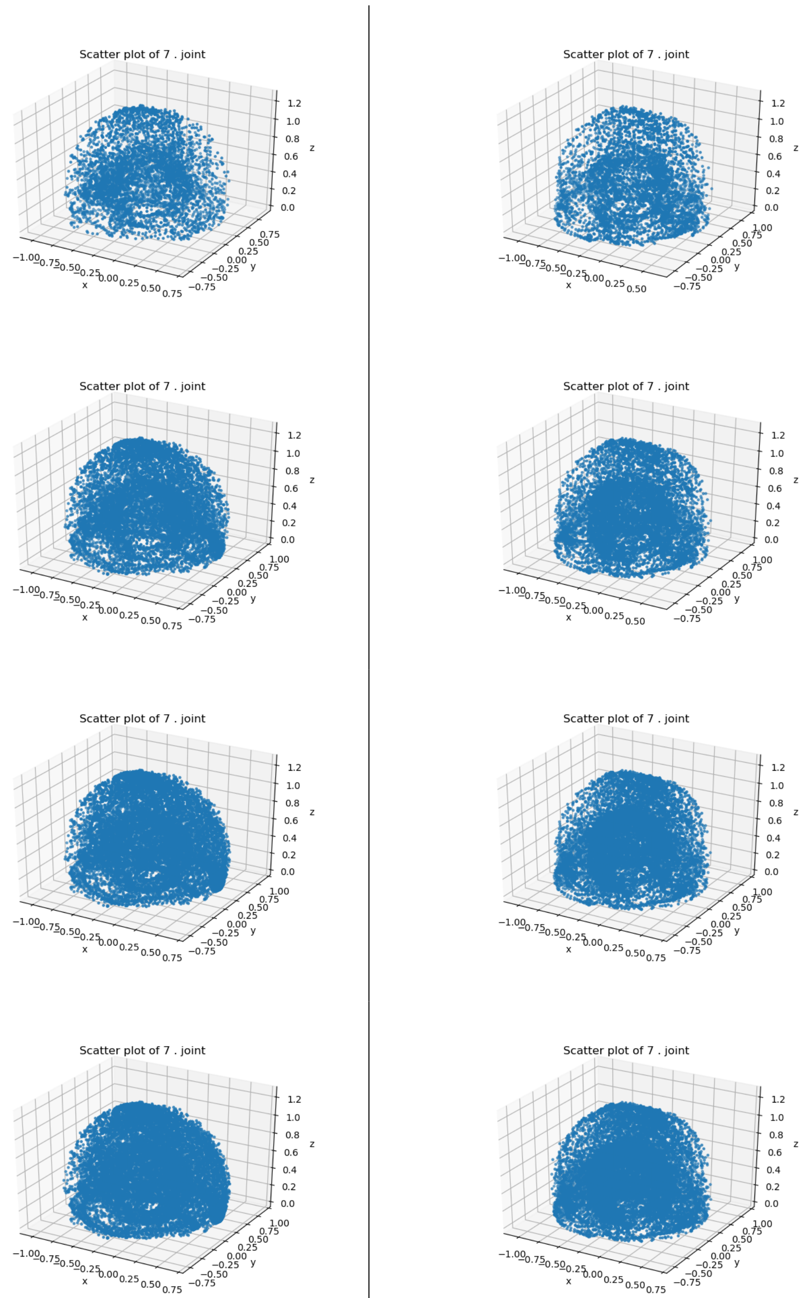
Figure 11: Comparison of the loss of the autoencoder after 60, 120, 160 and 200 episodes

5.2.2 Exploration of the working space

Now we take a look how the autoencoder approach fared in exploring the working space of the robot. Again as a reminder, the autoencoder approach used said autoencoder to reduce the state-action-pair in its dimensionality and thus, with the help of the CMA-ES, allowed to choose the next action, based on the lowest density. Figure ? shows the robot's endeffector position after 60, 120, 160 and 200 episodes for two examples in Cartesian coordinates. In contrast to the baseline approach, we can easily see that robot explored far more area of the working space. Even after 120 episodes, we can already observe that the autoencoder approach has explored the area quite evenly. Furthermore it seems that this method does not keep visiting the same coordinates again and again, since with increasing episodes the plots become more densely packed (compare the plots after 120 and 200 episodes).

The same can be observed in figure ? as well. It shows the plots of the density of the endeffector's position after 60, 120, 160 and 200 episodes. We can see that in contrast to the baseline approach, the density is not as centered around the mean. It is more evenly distributed between the positive and negative constraint of the joint. Nevertheless, we can also observe that the endeffector (as well as the other joints, see Appendix) seem to often occupy the fringe position, that is the position that is equal to the constraint. It seem that in the sampling process it often is the case that the lowest density can be reached, when one or more joints are locked in place and only the other joints are moving then.

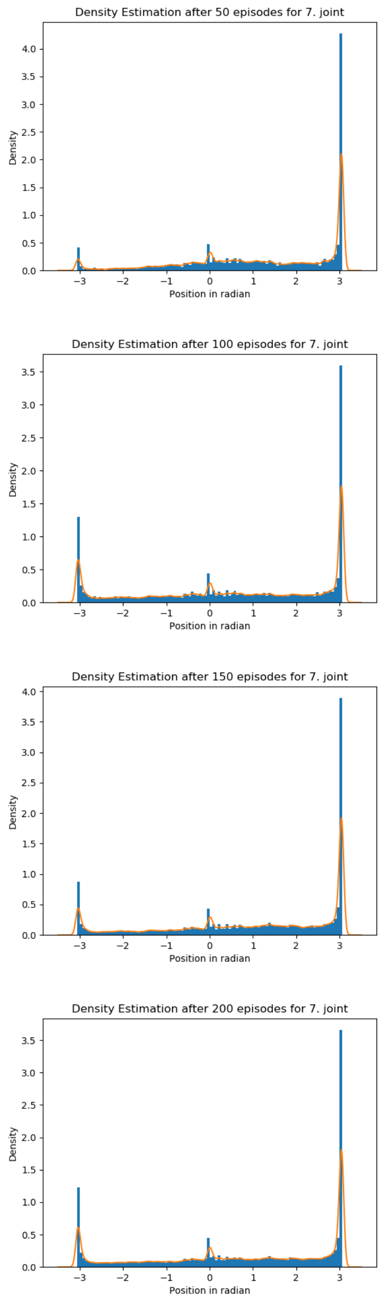
The evolution of the density distribution can be seen in figure ?. It shows the the density of the entire data (state action pairs) after 1, 50, 100 and 150 episodes. Since the goal of this approach is the minimization of exactly this density, the first the plots (after 1 and after 50 episodes) shows what is expected. The density decreased more or less significantly. But when we look at the density after 100 episodes, something unanticipated happened. The density increased. But if we now compare that to the figure ? and figure ?, we can see that already after 100 episodes, the approach has explored the working space quite evenly, which makes it harder for the process to find an action that leads to a position that has not yet been occupied and therefore has low density. Consequently the algorithm will oftentimes choose an action that will lead to an already visited position, what increases the total density.



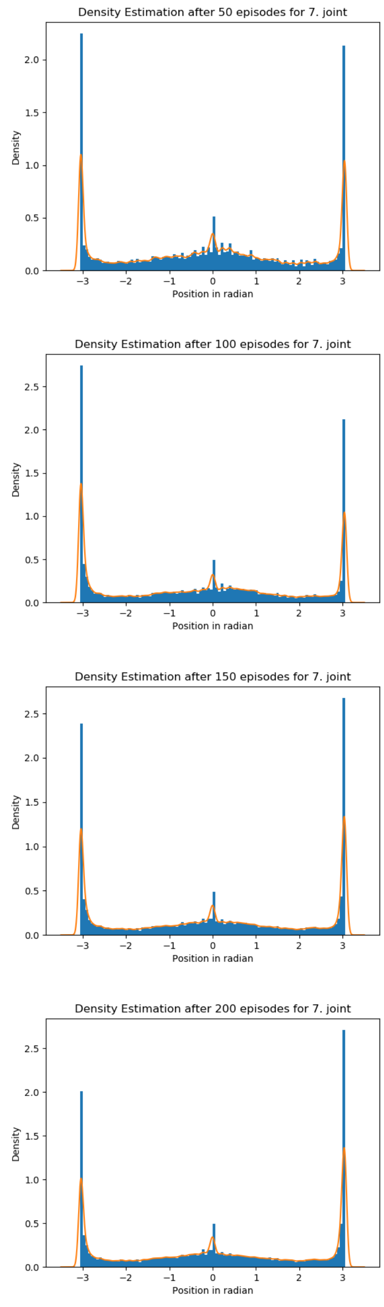
Plot of endeffector of one simulation after 60, 120, 160 and 200 episodes

Plot of endeffector of another simulation after 60, 120, 160 and 200 episodes

Figure 12: Comparison of the endeffector position after a different episodes

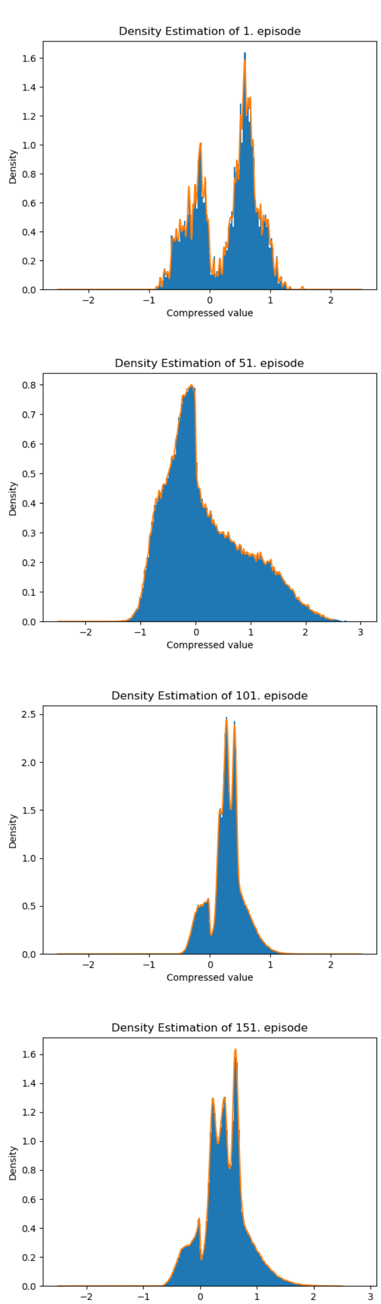


Plot of density of endeffector position of one simulation after 60, 120, 160 and 200 episodes

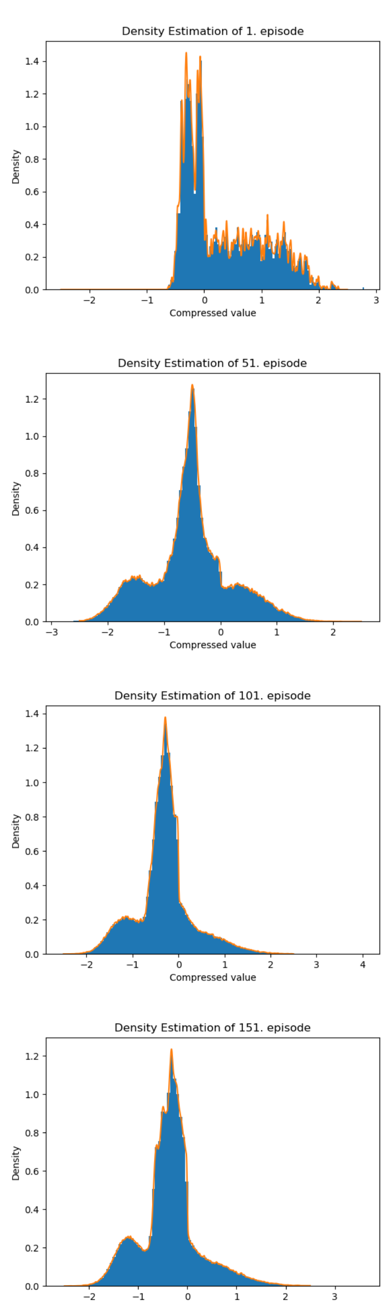


Plot of density of endeffector position of one simulation after 60, 120, 160 and 200 episodes

Figure 13: Comparison of the density of the position of the endeffector



Plot of density of endeffector position of one simulation after 60, 120, 160 and 200 episodes



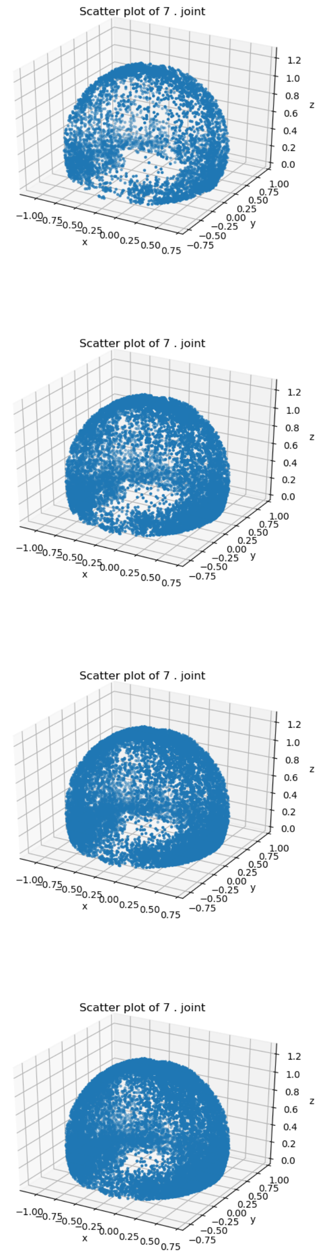
Plot of density of endeffector position of one simulation after 60, 120, 160 and 200 episodes

Figure 14: Comparison of the density of the position of the endeffector

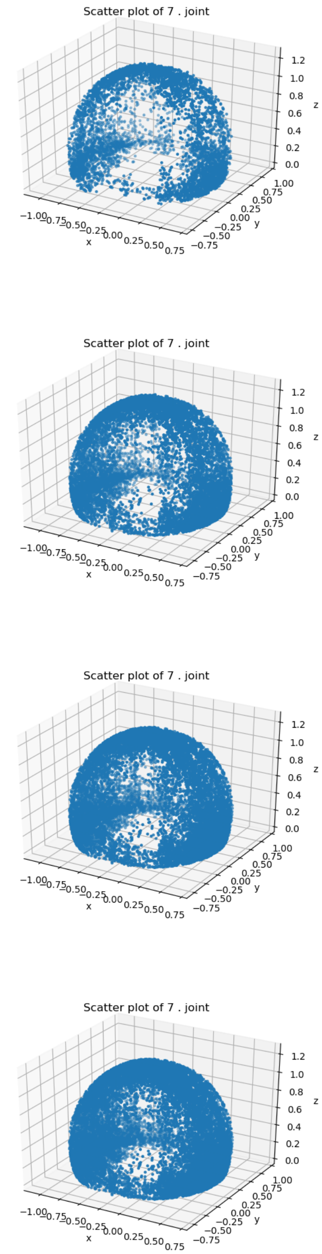
5.3 PCA exploration

Let us now look at the last approach of this thesis. Again, as a reminder, the PCA-approach reduced the dimensionality of the state-action-pair and thus, with the help of the CMA-ES, allowed to choose the next action based on the lowest density. In figure ? we can see the result of that approach. It shows the plots of the position of the robot's endeffector after 60, 120, 160 and 200 episodes for two examples in Cartesian coordinates.In contrast to the baseline approach we can once more see that the PCA approach seems to explore the space much more quickly and evenly. Nevertheless, when we take a closer look and compare it to the results of the autoencoder approach, we can observe that the PCA approach is nowhere close to explore the same area. The entire interior of the half sphere was apparently not visited by the endeffector.

The same conclusion can be drawn from figure ?. It shows the density of the endeffector's inherent position in Radian. Since the density does not spread out evenly between the positive and negative joint constraint and since the same can be observed for all other joints (see Appendix), we can deduce that not the entire working space of the robot has been visited, given that not the entire range of the joints has been used.

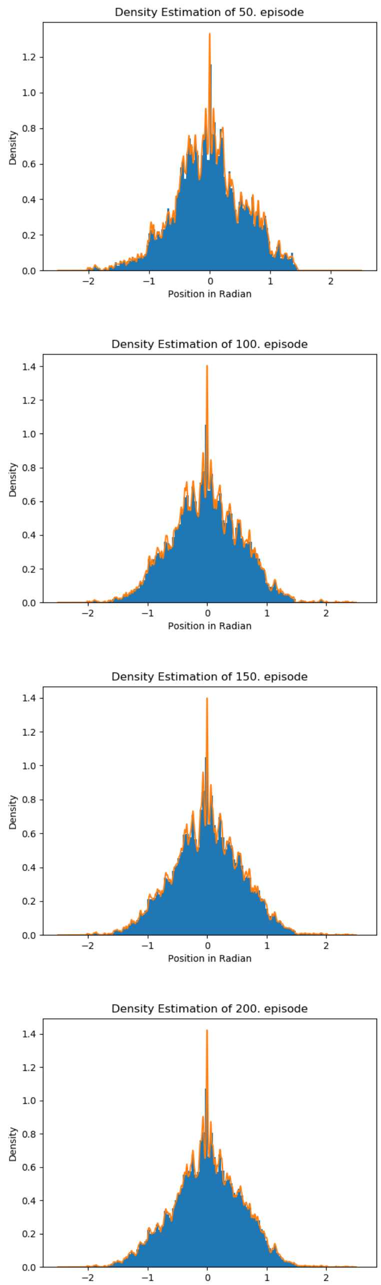


Plot of endeffector of one simulation after 60, 120, 160 and 200 episodes

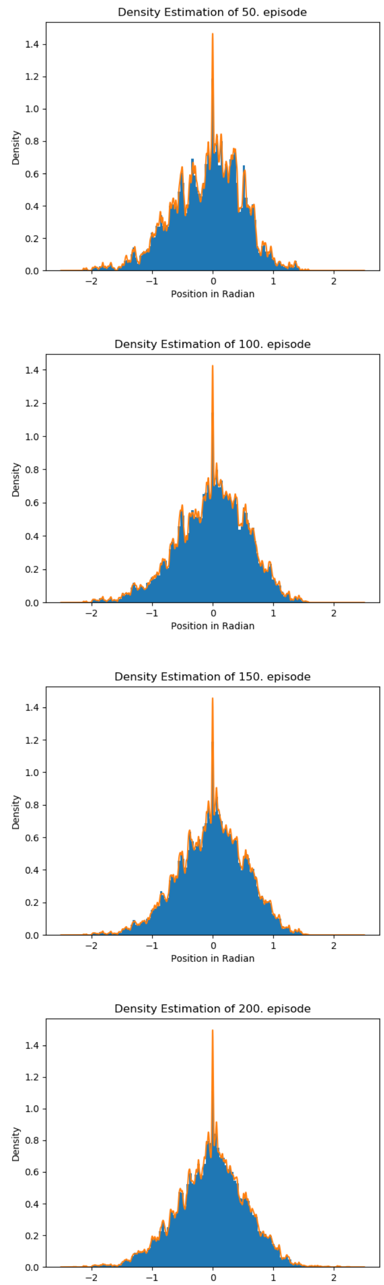


Plot of endeffector of another simulation after 60, 120, 160 and 200 episodes

Figure 15: Comparison of the endeffector position after a different episodes



Plot of density of endeffector position of one simulation after 60, 120, 160 and 200 episodes

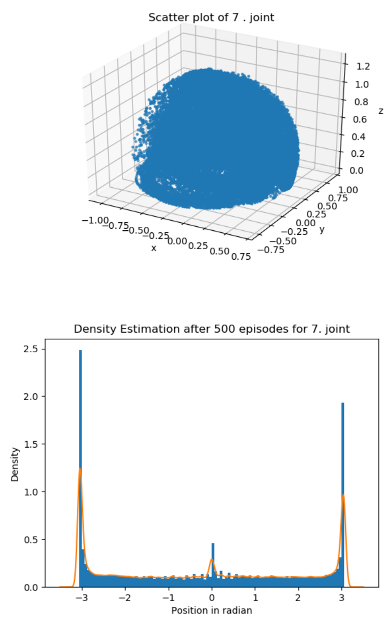


Plot of density of endeffector position of one simulation after 60, 120, 160 and 200 episodes

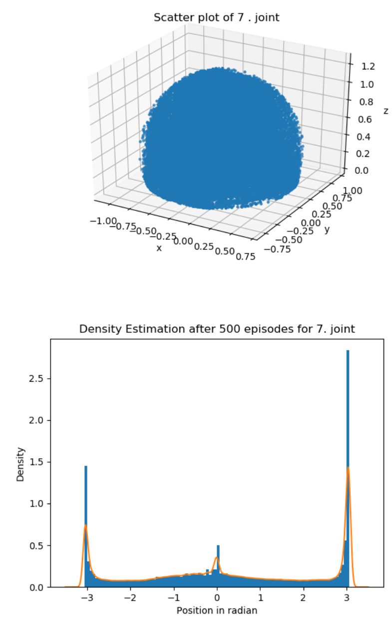
Figure 16: Comparison of the density of the position of the endeffector

5.3.1 Additonal

Each approach has been tested at least in two different simulations with the same settings for 200 episodes. That holds true for the baseline and the PCA approach. The autoencoder approach however has been conducted for 500 episodes. You can see the additional data in the figure ? below.



Plot of the position of the density of the position of the endeffector after 500 episodes for one simulation



Plot of the position of the density of the position of the endeffector after 500 episodes for another simulation

Figure 17: Comparison of position and the density of the position of the endeffector

5.4 Conclusion

This thesis presented three approaches for exploring the working space of a robot: Sampling from a normal distribution, sampling with the CMA-ES with density reduction through an autoencoder and sampling with CMA-ES with density reduction through PCA. We can evaluate these approaches through two different methods.
The first would be the time required to complete all 200 episodes. Here the random sampling from a normal distribution took the least and a constant amount of time, because the calculation was the same in each iteration and the computational cost did not rise with an increase in episodes. So it took

roughly 200*100*1 seconds for the program to finish. The PCA approach was the second fastest. The most time intensive computation here was the fitting of the PCA with the data. To limit the computational cost, this data was restricted to the last 20 episodes. So, after these 20 episodes, each iteration took about 1.5 seconds. Hence the entire time is roughly 200*100*1.5 seconds. The autoencoder approach the most amount of time. In each iteration the existing autoencoder had to be evaluated on the collected data. Like with the previous approach, to limit computational time, this data has been restricted to the last 20 episodes. After these an iteration took about 1.5 seconds. Thus, the computational time during the episode is similar to PCA approach. But after each episode it was also necessary to train the autoencoder. That was done with all the to this point acquired data. Hence the computational time increased with the increase of the episodes. While in the first few episodes the training did not take make than a few seconds, in the later episodes the same training took almost an hour.
The first method for evaluation is how well the space has been explored. Each approach ran for 200 episodes, which means that the amount of occupied points is the same. Now we can look how evenly they are distributed though out the space. For that we can look onto the figures from the previous chapter, namely figure ?, ? and ?. Here we can see that the random sampling from a normal distribution fared the worst. Only a chunk of the entire space has been explored and it seems that the robot's endeffector keeps visiting the same spots over and over. The other two approaches seem to explore the space much more evenly. But we can clearly see, when we compare figure ? and figure ? that the autoencoder approach was exploring the working space of the robot more extensively than the PCA approach, for the interior of the sphere has not been well explored in the latter.