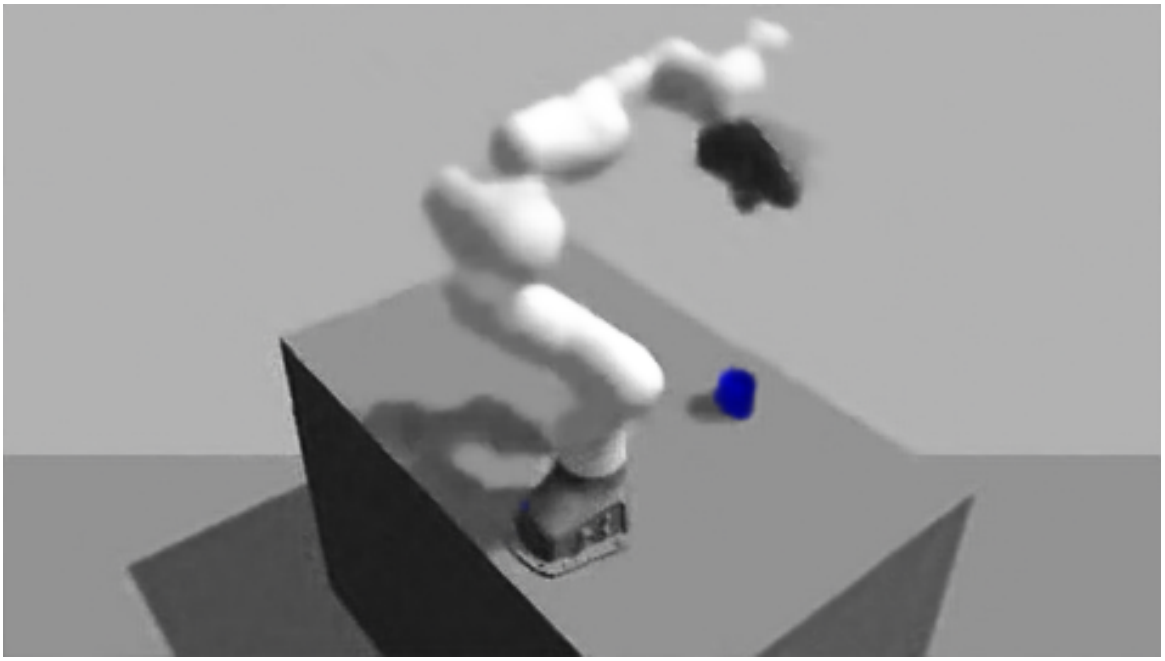
MSc Artificial Intelligence
Master Thesis

Bridging the Reality Gap for Image-Based Robot Learning using Auto Encoders



Caitlin Gwendolyn Lagrand
August 12, 2019

Supervisor:
Dr N. van der Stap (TNO)
Dr H. van Hoof

Assessor:
Dr A. Visser

University of Amsterdam

MSc Artificial Intelligence
Master Thesis

Bridging the Reality Gap for Image-Based Robot Learning using Auto Encoders

---

by
Caitlin Gwendolyn Lagrand
10759972

August 12, 2019

36 E.C.
January 2019 - August 2019

Supervisor:
Dr N. van der Stap (TNO)
Dr H. van Hoof

Assessor:
Dr A. Visser

Graduate School of Informatics

Abstract

With recent developments in Artificial Intelligence and Robotics, the use of robots in daily life is rising. However, letting a robot interact with the real world has not become easier. Currently, a lot of manual engineering and fine tuning is needed to let a robot perform tasks, and adjusting the robot to let it perform another task can be time-consuming. One of the developments is the increase of the use of Reinforcement Learning (RL) for a variety of tasks. This development arose our interest and therefore we want a robot to learn to reach a cup using RL. However, training RL on a real robot can also be time-consuming and even dangerous for the robot, since the robot has to act (randomly) for a long time before learning something useful. Instead, a simulator can be used. Unfortunately, transferring the learned policy from the simulator to the real world does often not result in the same behaviour, since the simulator is never exactly the same as the real world.

With the goal to perform RL in a safe setting and make it achievable with limited resources such as one real robot, one GPU and limited time (a few days to train), this thesis has the objective to bridge the visual reality gap to be able to learn in simulation and transfer the learned policy to the real robot. In this thesis, we focus on training a common state representation for the simulated and real world in order to use this common state as input for RL. We extend the method from Inoue et al. [2018] by learning the common state representation using two Variational Auto Encoders (VAE) and Deep Deterministic Policy Gradient (DDPG) is used for the robot to learn how to reach a cup.

From the experiments that are performed, we can conclude that a common latent space for the simulated and real world can be learned using VAEs if the environment does not change over time. For environments in which this is not the case, the VAEs fail to generalise to data obtained at different moments. The RL experiments have shown that this latent space can be used to let a robot learn how to reach a cup using RL in simulation. Compared to using the position of the cup directly, the performance of using the latent state as input is the same. This shows that all information needed to reach the cup, such as the position of the cup, is captured in the latent state. A policy that uses the latent state as input has learned to reach a cup at ten different positions with an accuracy of 74% within a range of 10cm in simulation.

Since the results of the latent state were not sufficient to reconstruct the scenes with the robot in the real world, the trained policy is only tested in simulation and not in the real world. Thus, it is not possible to determine whether the latent space generated by VAEs can be used to bridge the visual reality gap, which would allow policies trained in simulation to be transferred to the real robot.

Acknowledgements

I would first like to thank my supervisors Dr. Nanda van der Stap from TNO and Dr. Herke van Hoof from the University of Amsterdam. They have supported me throughout my thesis with valuable feedback and ideas. I would also like to thank Dr. Arnoud Visser for agreeing to be my assessor and more importantly for making me enthusiastic about robotics during my study.

Furthermore, I am grateful for the thesis coaching sessions from Dr. Sander van Splunter and Yasmin Santis. These sessions were really helpful and it was a nice opportunity to discuss thesis struggles with fellow students.

I would also like to thank all colleagues and interns at TNO, who have supported me during my thesis by helping me setting up or by discussing some of the problems I ran into. Finally, I would like to thank my fellow students, friends and family for always supporting me and listening to me when I needed it.

Contents

Chapter 1

Introduction

With recent developments in Artificial Intelligence and Robotics, the use of robots in daily life is rising. For example, service robots are showing up at hotel receptions and stores, robot vacuum cleaners are cleaning more and more houses and self driving cars are being tested on highways and in cities. However, a lot has to be done before we will have robots helping out on a daily basis, such as in the kitchen for example by cooking diner.

A robot that helps out in the kitchen by grabbing a cup would need to recognise the cup and grab it. Recognising the cup is relatively easy due to the rise of Deep Learning (DL), which can be used to learn an object detector that can detect a variety of objects under different circumstances, such as lighting, occlusion and orientation [Zhao et al., 2018]. Grabbing the cup, however, is a harder task, since the gripper of the robot should fit perfectly around the object and the robot should use the right amount of force while the stiffness and mass of the object might be unknown. If the position of the object is precisely known, for example by using an accurate object detector, the robot might still not be able to grab a cup that has fallen on its side, since another way of grabbing the cup needs to be employed compared to a standing cup. If the orientation is also precisely known, different ways of grabbing that cup can be employed, but the engineer needs to provide the robot with all these different ways. This can be time-consuming, and some orientations can be missed resulting in the robot not being able to grab the cup.

The use of Reinforcement Learning (RL) has increased for a variety of tasks. RL uses rewards obtained from the environment to learn actions that should be taken, resulting in an action policy that is learned from own experiences. RL enables us to learn different tasks without a lot of manual engineering, by letting an agent perform actions in an environment by trial and error. So, instead of telling the robot to first move forward, then turn left, then grasp, it will learn this by itself, or even learn a better solution. For example when grabbing the cup with different orientations, RL could learn how to do this, instead of manually engineering it. RL has shown to be effective in different areas, for example in playing games like Pong [Mnih et al., 2015] or Go [Silver et al., 2016], in controlling helicopters [Abbeel et al., 2007] and robots [Tai and Liu, 2016]. The idea emerges that if Reinforcement Learning could be used to let a robot learn a task by itself, a lot of time spent on manual engineering and fine tuning would be saved to complete complex tasks with robots.

Currently in robotics, RL has mainly been used for navigating, reaching, grasping or moving objects [Tai and Liu, 2016]. For example, Levine et al. [2016] trained robotic arms to successfully grasp different objects. They did this by using between 6 and 14 robotic arms at the time and letting them train for two months. This shows that training RL with robots in the real world can be time-consuming and dangerous, since the robot has to act (randomly) for a long time before

learning something useful, which arises an interest in learning control policies using a simulator. Unfortunately, transferring the learned policy from the simulator to the real world does often not result in the same behaviour in the real world, since the simulator is never exactly the same as the real world. This *reality gap* between simulation and the real world is even larger when using images as input data: rendered images resemble real images, but are not exactly the same.

Current methods that focus on bridging this reality gap try to generate a lot of variety in simulation to capture the environment of the real world [Tobin et al., 2017] or use progressive neural networks to continue learning in the real world [Rusu et al., 2016b]. However, these methods often require a lot of training in simulation or even some training in the real world. With the goal to perform RL in a safe setting and make it achievable with limited resources such as one real robot and limited time (a few days to train), this thesis has the objective to bridge the visual reality gap to be able to learn in simulation and transfer the learned policy to the real robot. Therefore, the research question is:

> **RQ** *How can the visual gap between the simulated and real world be bridged in order to perform RL with robots?*

Our hypothesis is that this gap can be bridged by learning a common representation for the input image from the simulated and real world using Variational Auto Encoders (VAE) as proposed by Inoue et al. [2018]. RL can then use this common representation as input state to learn how to perform tasks.

To answer the research question, we extend an existing method from Inoue et al. [2018] that uses VAEs to bridge the gap between simulation and the real world. Instead of learning a common representation for training an object detector as Inoue et al. do, we use the common representation directly as input for RL. We choose this method, since it uses knowledge about the real world in contrast to other methods that only use information from the simulator [Tobin et al., 2017] [Tremblay et al., 2018]. We believe that using this data can be beneficial and therefore we want to extend current work that uses this information. We experiment with a reaching task to show the possibilities of the proposed method combined with RL.

1.1 Outline

This thesis is organised as follows. Firstly, Chapter 2 provides background knowledge about (Variational) Auto Encoders and Reinforcement Learning, followed by related work about transferring trained models from simulation to the real world and motion control for robots in Chapter 3. Next, our proposed method to bridge the visual reality gap to perform RL with robots is explained in detail in Chapter 4. In Chapter 5 the experimental setup is described, including the hardware and the software implementation of our method. The experiments and results of the VAE are then described in Chapter 6, followed by the experiments and results of the RL tasks in Chapter 7. Finally, the conclusion and discussion of this thesis are described in Chapter 8, along with suggestions for future work.

Chapter 2

Background

This chapter provides background knowledge about (Variational) Auto Encoders in Sec. 2.1 and Reinforcement Learning in Sec. 2.2. Variational Auto Encoders are used in this thesis to learn a common state representation to be able to learn tasks with Reinforcement Learning in simulation and transferring the learned policy to the real world.

2.1 Auto Encoders

To be able to learn from simulation data, a representation which is the same for the same state in the simulated and real world is desired. An auto encoder [Ballard, 1987] can learn this representation by learning to generate the same image for both the input image from the simulated world and the input image from the real world. This representation should only capture the information that is in both the simulated and real world and can be used instead of the raw input.

An auto encoder is an unsupervised learning algorithm that learns a latent space for the input. A latent space is a low dimensional representation that holds the information needed to represent the input. An auto encoder consists of two parts: an encoder and a decoder. Figure 2.1 shows a schematic depiction of an auto encoder: the input image is given to the *encoder*, which needs to learn to encode it into a representation z that is often lower dimensional than the input. This representation is given to the *decoder*, which generates a resembling image of the input image. The encoder and decoder itself are often (deep) neural networks to be able to learn a good low dimensional representation. The generated image should resemble the input image as closely as possible, thus the objective of the auto encoder is to minimise the distance between the input image and the generated image, also known as the reconstruction error:

$$loss = ||x - decoder(encoder(x))||_2^2. \tag{2.1}$$

2.1.1 Variational Auto Encoder

In contrast to auto encoders, Variational Auto Encoders (VAE) [Kingma and Welling, 2013] learn the parameters of a probability distribution representing the data, a latent variable model. This makes sampling from the model possible, which means that new input data can be generated. In a VAE, the encoder is represented by a variational distribution, $q_\phi(z|x)$ and the decoder is represented by a conditional distribution $p_\theta(x|z)$ used to reconstruct the input (x) given the

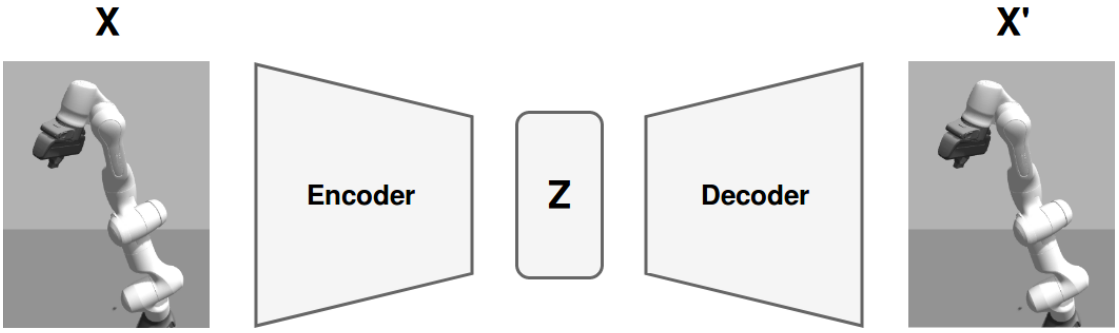


Figure 2.1: Schematic of an auto encoder: the input image is fed to the encoder, resulting in a latent representation z, which is passed through the decoder to reconstruct the input image.

latent state (z). Since the encoder is a variational distribution, a lower bound for $p(x)$ can be derived with Jensen's inequality [Jensen, 1906]:

$$\begin{aligned} log p_\theta(x) &= log \int p_\theta(x,z)dz \\ &= log \int q_\phi(z|x)\frac{p_\theta(x,z)}{q_\phi(z|x)}dz \\ &\geq \int q_\phi(z|x)\frac{p_\theta(x,z)}{q_\phi(z|x)}dz \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}[log p_\theta(x|z)] + D_{KL}(q_\phi(z|x)||p_\theta(z)), \end{aligned} \tag{2.2}$$

with $\mathbb{E}_{z\sim q_\phi(z|x)}[log p_\theta(x|z)]$ being the reconstruction loss used to encourage the decoder to reconstruct the data and $D_{KL}(q_\phi(z|x)||p_\theta(z))$ the KL-divergence used to regularize how much information is lost when using $q_\phi(z|x)$ to represent $p(z)$.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a general framework for decision problems and is used to learn what actions should be taken to achieve a goal. In RL, an agent interacts with its environment by taking actions and observing the state of the environment and obtaining rewards as shown in Fig. 2.2. The rewards are used to learn a good policy: what action to take at a certain state. By starting out with a random policy and updating it according to the rewards, the robot should be able to learn a good policy for its task in the environment. In order to learn by trial and error, the agent needs to start with exploring the environment by trying different actions. Actions that resulted in high rewards are likely to be good actions and need to be exploited to achieve a high final reward. However, always taking the actions that led to high rewards will often not result in the highest cumulative reward, since their might be a better solution which the agent has not explored yet. Thus, there always exists a trade-off between exploration and exploitation in RL.

A Markov Decision Process (MDP) [Puterman, 1994] is often used in RL to learn a policy. An MDP is a mathematical framework for modelling decision making and consist of a tuple containing states (S), actions (A), transition probabilities ($P(s'|s,a)$) and rewards ($R(s,a,s')$). The states are observations an agent makes in a given environment. For example, a state can be the raw input image or the joint values and velocities of a robot arm. The actions are the

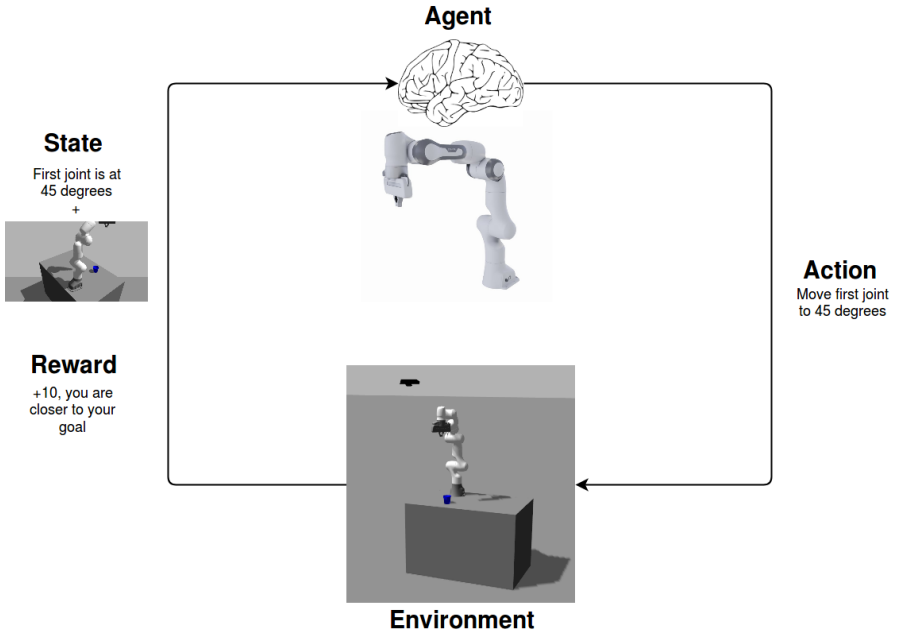


Figure 2.2: Overview of Reinforcement Learning: an agent interacts with its environment by taking actions, observing the state of the environment and obtaining rewards. With the obtained state and reward, the agent updates its policy in order to predict better actions.

possible actions an agent can take, which can be discrete, such as move forward or continuous, such as move to this joint value or go to this position. The transition probabilities describe how the environment changes and can be deterministic: if the agent moves forward, it will be at $x+1$, or stochastic: after moving forward the agent will be at $x+0.5$ with a probability of 0.2 and at $x+1$ with 0.8 probability. The rewards can be sparse, like winning (+1) or losing (-1), or dense: the closer to the target object the higher the reward. The goal of RL is to find an optimal policy π: what action to take at a certain state. In order to find the optimal policy, the cumulative rewards are maximised.

RL methods can be divided into three main categories: model-based methods, value-based methods and policy-based methods. Model-based methods try to learn the model underlying the MDP and are often used in cases where interactions with the environment are limited. The model is a model of the environment and is used to simulate more episodes for performing RL. In contrast, value-based and policy-based methods are model-free methods and depend on sampling from the environment. In this thesis, training will be performed in simulation and interactions with the environment are not limited, thus a model-free method is used.

2.2.1 Value-based

Value-based methods try to learn a value function V that can be used to compute the expected discounted reward if starting in some state s and following policy π:

$$V_\pi(s) = \mathbb{E}_{s,a\sim\pi}[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t)] \quad \forall s \in S, \tag{2.3}$$

where T is the episode length and can be infinite for continuous environments and γ is the discount factor that determines the importance of future rewards. The optimal policy is then the policy that optimises this value function:

$$\pi^* = \text{argmax}_\pi V_\pi(s) \quad \forall s \in S. \tag{2.4}$$

The action that will be taken in state s according to the best policy π^*, is thus the action that results in the highest expected discounted reward.

Value-based methods are a key building block for many RL methods and are good at solving simple tasks, such as solving a maze. However, they are limited to discrete actions and thus not suitable for more complex tasks that need continuous actions.

2.2.2 Policy-based

Policy-based methods directly learn the parameters of the policy, instead of finding the policy that optimises the expected discounted return. Policy Gradient methods use the gradient of the policy to update the policy using gradient ascent: the policy is updated in the directions with the steepest reward increase. The best parameters of a policy are found by maximising a policy score function:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} \gamma^{t_1} r_t]. \tag{2.5}$$

Instead of optimising the value function as value-based methods, policy-based methods try to learn the optimal policy directly by learning the parameters of the policy. This has three main advantages over value-based methods. Firstly, policy-based methods are more effective in high-dimensional action spaces and even continuous actions are possible, while value-based methods can only handle discrete actions. Secondly, policy-based methods can handle both deterministic and stochastic policies, while value-based methods can only learn deterministic policies. The advantages of stochastic policies are that they can handle exploration and exploitation and that one state can have multiple actions that can be taken. Lastly, value-based methods can have big oscillations during training, since a small change in the estimated value function can lead to really different actions, while policy-based methods just follow the gradient resulting in a smooth update of the policy at each step. Since the gradient is followed, policy-based methods are guaranteed to converge to a local or global maximum. Unfortunately, they often converge to a local maximum, but value-based methods do not have this guarantee when function approximations over states are used. The biggest disadvantage is that policy-based methods converge slower, so training will take longer.

2.2.3 Actor-Critic

Actor-Critic [Konda and Tsitsiklis, 2000] methods are hybrid methods that use a Critic to measure how good the taken action is (value-based) and an Actor to control the behaviour of the agent (policy-based). One of the drawbacks of a policy gradient is that the future reward can only be calculated at the end of an episode, meaning that high future rewards are given to all taken actions in that episode if the final future reward is high. However, bad actions will also get this high future reward and thus to find the optimal policy, a lot of samples are needed to give these bad actions lower rewards, which slows down learning. Instead of waiting until the end of the episode, Actor-Critic methods update at each step, using the expected reward (Critic) instead of the real future rewards. The policy (Actor) is now updated directly using this expected reward instead.

Chapter 3

Related Work

This chapter provides an overview of the related work. Firstly, Sec. 3.1 describes the related work that has been done to bridge the gap between the simulated and real world. The described methods are general methods for bridging the visual gap, but they are or can be extended to bridge the visual gap for Reinforcement Learning (RL) purposes. Next, related work about controlling the robot is provided in Sec. 3.2. In this section, classic methods such as path planning are reviewed, as well as Reinforcement Learning methods.

3.1 Simulation to Real World

Learning specific tasks in the real world requires a lot of interaction with the real world and can be expensive, dangerous and time-consuming, since the robot has to act (randomly) for a long time before learning something useful. As mentioned in Chapter 1, this makes it interesting to learn control policies using a simulator. Unfortunately, transferring the learned policy from the simulator to the real world often does not result in the same behaviour in the real world, since the simulator is never exactly the same as the real world. This reality gap between simulation and the real world is even larger when using images as input data: rendered images resemble real images, but are not exactly the same. A lot of effort could be put into making the simulated world look exactly like the real world with all textures and lighting identical, but this will never be completely the same.

One way to bridge the gap between the simulated and real world is using domain randomisation or adaptation [Tobin et al., 2017] [Ben-David et al., 2010]. Domain randomisation uses randomised environments in simulation with enough variability. The variability is obtained by randomising several aspects in the environment, such as the position and texture of objects on the table, the position, orientation, and field of view of the camera, etc. The idea of domain randomisation is that *if the variability in simulation is significant enough, models trained in simulation will generalise to the real world with no additional training* [Tremblay et al., 2018] [Tobin et al., 2017] [James et al., 2017]. This way, a model can be trained in simulation and then used in the real world. However, it requires a lot of training in simulation with the different environments and the environments should have enough variability to prevent the rise of a bias. Besides domain randomisation, domain adaptation can be used to fine tune a model learned on simulated data ($\sim 10,000$ samples) with a small amount of data from the real world (~ 100 samples) [Ben-David et al., 2010] [Higgins et al., 2017]. For RL, this method still needs a lot of training in simulation. Moreover, some training with the real robot in the real world will always be needed.

Another approach uses Progressive Neural Networks (PNN) to bridge the reality gap between simulation and the real world [Rusu et al., 2016b]. PNNs [Rusu et al., 2016a] are immune to forgetting and thus can continue learning from new data, without forgetting the previous data. This can be used to first train a PNN in simulation, followed by training it in the real world. Compared to fine tuning a simulation model with data from the real world, PNNs need less data from reality to reach the same performance as the PNN in simulation. Moreover, less training in simulation is needed compared to domain randomisation, since only one environment is needed in simulation. However, as domain adaptation, PNNs still need some training with the real robot in the real world.

Instead of learning in an end-to-end manner as domain randomisation and PNNs, learning can also be split into phases [Zhang et al., 2017]. For example a vision phase and a motion control phase. The advantage of splitting it into phases is that the phases can be trained separately and that they can be trained on data from other tasks, such as pretraining the vision network on Imagenet [Deng et al., 2009]. However, an inaccurate vision model influences the motion model and in most cases the vision model will not learn from the errors that the motion model makes.

A method that uses two phases for learning is learning a common representation for the simulated and real world using Auto Encoders [Inoue et al., 2018]. This method focuses on learning a latent space which is the same for the same state in the simulation and the real world. Either the latent space or the generated image can then be used to learn from, instead of the original image. The idea behind learning this latent space is that only the relevant information needed to describe the same state in simulation and the real world is captured. In contrast to domain randomisation and PNNs, a data set is needed with image pairs from simulation and the real world. This requires some interaction with the real world, but this interaction does not have to be random and is minimal. On the other hand, just one world can be built in simulation to obtain a data set from, which does not have to be as big as the one used for domain randomisation. Furthermore, no training in the real world has to be performed in contrast with domain adaptation and PNNs. Inoue et al. [2018] use two Variational Auto Encoders (VAE) [Kingma and Welling, 2013] to learn a latent space for both the simulated and real world. The VAEs are used to detect object positions either by using the generated images and a Convolutional Neural Network (CNN) or the latent representation and a MultiLayer Perceptron (MLP). The resulting object detector could estimate the position of an object in the real world with an average error of 2.2mm, which shows that accurate object manipulation could be performed on the latent space instead of on the raw image.

3.2 Motion Control

Motion control is determining the mapping between the control commands and the motion of the robot. The commands are the direct controls sent to the actuators of the robot, such as the velocity and angle for a joint. Setting these commands should take the workspace of the robot into account, since it should be capable of moving to the joint values while avoiding collisions. Controlling the robot can be divided into two main parts: path planning, which computes the optimal path from A to B, and reinforcement learning, which lets a robot learn how to move by itself from (raw) sensor data.

In classic control, path planning is often used to compute the path from A to B while avoiding obstacles. With robots, forward and inverse kinematics are used to compute the pose of the robot in the real world in order to compute a path for moving towards a goal using a path planning algorithm such as Dijkstra [Dijkstra, 1959], A* [Hart et al., 1968] or STOMP [Kalakrishnan et al., 2011]. The motion planning algorithm then computes the controls the robot should perform to

reach its goal, like the velocity and angle for each joint, while avoiding obstacles. These algorithms need a good representation of the world, including all obstacles it should avoid. With a good representation, motion planning works well in static environments, but small changes in the environment can cause the robot to not be able to perform its task. Furthermore, the end of a path needs to be known, meaning that the position of the goal needs to be known in the world, thus the goal needs to be detected by a different algorithm before being able to move towards it.

With the rise of Reinforcement Learning, it is possible to let a robot learn how to perform specific tasks by itself, saving the labour of defining every possible action manually. From raw sensor data, such as images, a robot can learn tasks in an end-to-end manner, instead of first detecting its goal and then reaching it. For example, in [Meyes et al., 2017] a UR5 Robot has learned to successfully play the classic wire loop game using Reinforcement Learning, with a camera to capture the input state, and move forward/backward/left/right/turn-left/turn-right as actions. Instead of directly learning in the real world, Pinto et al. [2017] use training in simulation as advantage by exploiting the full state observability provided by the simulation. The full state observability is all information about the environment that exists, but which might not always be visible in the real world. This is done by using an asymmetric actor-critic algorithm in which the critic is trained using the full states while the actor only gets rendered images as input. Combined with domain randomisation, their policy could be transferred to the real world without training on any real world data while being able to perform the task in the real world successfully in all trials.

These Reinforcement Learning methods are more flexible and adaptive than path planning methods, since they use the sensor data directly as input and thus can react easier to moving obstacles. Moreover, they do not need a representation of the world, since this is already captured by the sensor data. However, they do require a lot of interaction with the environment to train a good policy, which might lead to problems in the real world, such as time and wearing out the robots joints. However, by using a simulation as in this thesis, a lot of these interactions can be performed without any problems.

This thesis focuses on bridging the visual reality gap to be able to learn tasks in simulation using Reinforcement Learning and transfer the learned policy to the real robot. A real robot is available to collect a data set in a controlled setting, enabling us to use information about the real world for bridging the reality gap. Therefore, we combine the VAE as proposed by Inoue et al. [2018] with Reinforcement Learning. Instead of only using the VAE for the object detector as in [Inoue et al., 2018], the latent state obtained with the encoder of the VAE will be used as input for RL.

Chapter 4

Method

This thesis builds on using Variational Auto Encoders (VAE) to bridge the gap between the simulated and real world [Inoue et al., 2018]. Since a real robot is available to collect a data set in a controlled setting, this enables us to use information about the real world, in contrast to domain randomisation or adaptation [Tobin et al., 2017] [Higgins et al., 2017], which need a lot more simulation data to capture the real world in random environments. The method of Inoue et al. [2018] is extended by using the latent space obtained from the VAE as input for Reinforcement Learning (RL) to learn tasks.

This chapter explains the proposed method, which consists of two parts: the Variational Auto Encoder and the Reinforcement Learning part. The VAE is used to encode the same state in simulation and the real world to the same latent state and is explained in Sec. 4.1. This latent state is then, together with the current joint values, the input state for the RL part, which uses this input state to compute the desired joint values of the robot as described in Sec. 4.2. The RL part is trained in simulation and then transferred to the real world to test the policy with the real robot. An overview of this method is shown in Fig. 4.1.

4.1 Variational Auto Encoder (VAE)

To learn a latent space in which a state from both the simulated and the real world is represented as the same state, two Variational Auto Encoders are used. The encoders of the VAEs output parameters for a Gaussian probability density: the mean and the log variance, from which noisy values of the representation z can be sampled. The decoders try to reconstruct the input image, given the latent representation z.

The two VAEs have the same network architecture which is shown in Fig. 4.2. The architecture is based on the network in [Inoue et al., 2018] with small modifications to handle the image size required for our camera. The encoder is used to obtain a latent state from an image and has three convolutional layers with max pooling. After the last convolutional layer, two separate fully connected layers follow, one for the mean of the latent space and one for the log variance. The decoder is used to reconstruct the input image from the latent state. Therefore, a fully connected layer with a reshape afterwards is followed by three deconvolutional layers with upsampling. A final convolutional layer is used to obtain three channels (RGB) again.

The VAEs are trained on a data set that consists of images from the simulated and real world. A small part of the data set contains image pairs with images from both the simulated and real world of the same state as shown in Fig. 4.3. The rest of the data set contains only images from simulation.

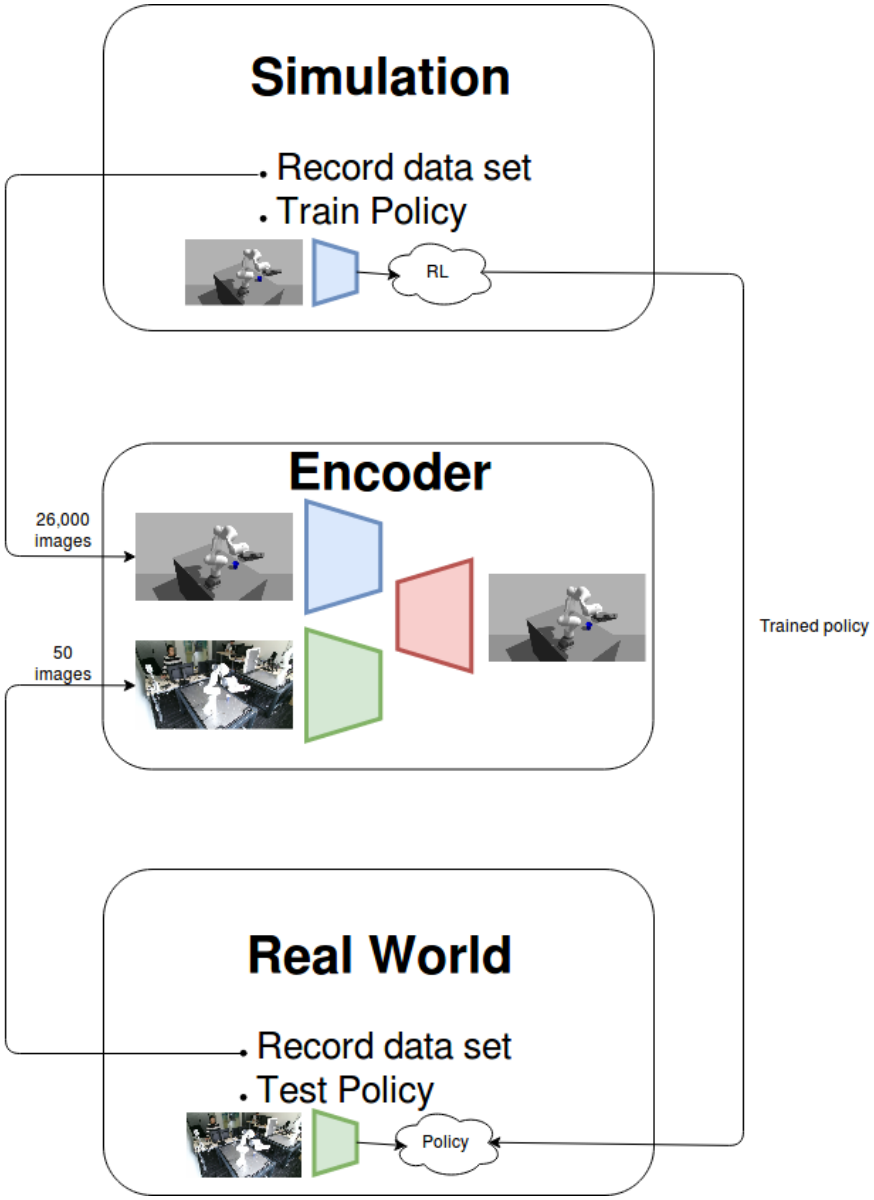


Figure 4.1: An overview of the proposed method: firstly, a data set containing a lot of images from the simulated world and a few from the real world is created and used to train a VAE to learn a common state representation. Next, a policy is learned in simulation using the simulation encoder (blue). This policy is then transferred to the real world using the real world encoder (green).

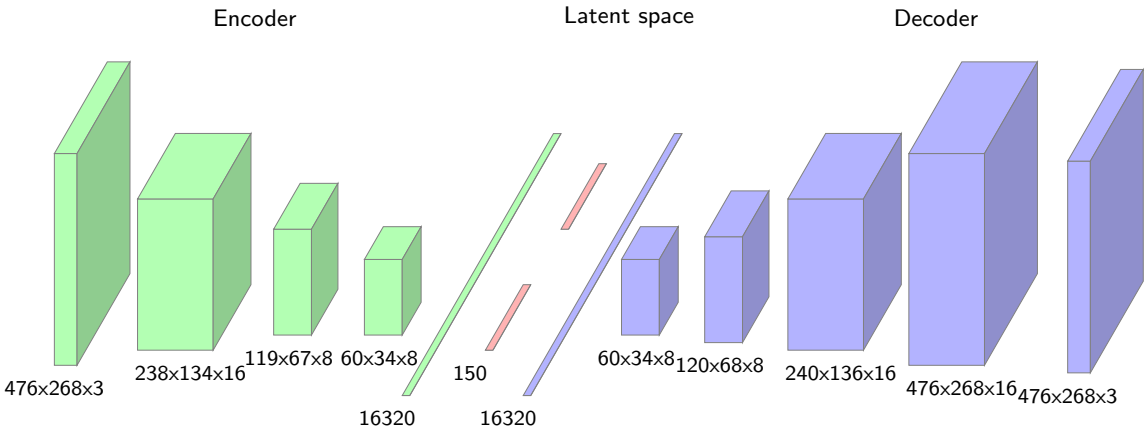
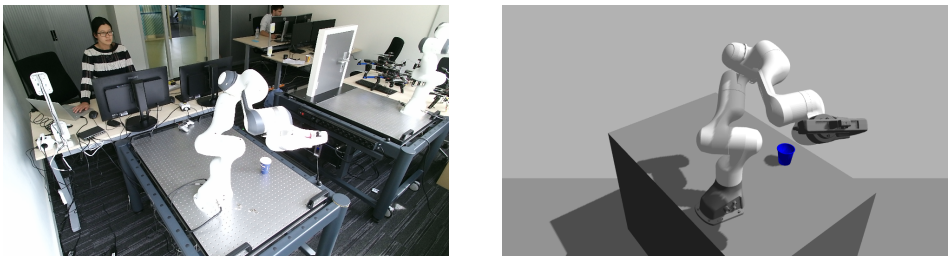


Figure 4.2: The network architecture for the VAEs. The encoder (green) consists of three convolutional layers with max pooling, followed by two separate fully connected layers, resulting in the mean and log variance (red) of the latent space. The decoder (blue) has one fully connected layer with a reshape afterwards, followed by three deconvolutional layers with upsampling to reconstruct the input image.



(a) Real world (b) Simulated world

Figure 4.3: Example image pair with the same state from the real (a) and simulated world (b).

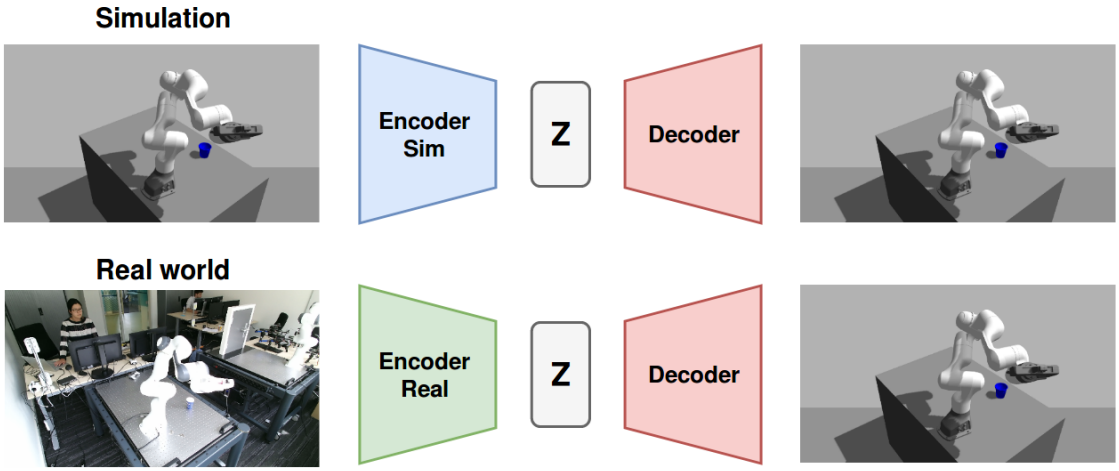


Figure 4.4: Training procedure for the VAE. Firstly, the simulation encoder and decoder are trained on a large data set. Next, the real world encoder is trained on image pairs, while keeping the decoder fixed.

4.1.1 Training VAEs to Learn a Common State Representation

Figure 4.4 shows an overview of the training procedure of the two VAEs, which is based on the method as described by Inoue et al. [2018]. Firstly, a VAE is trained on only simulation data to learn a good latent representation. Next, the weights of the simulation VAE are copied to the real world VAE and the real world encoder is then trained on image pairs, with the input image being the real world image and the output image the simulation image. To ensure that the encoder of the real world will learn to encode the real world state as the same latent state as the simulation encoder would, the weights of the decoder are kept fixed. With the weights of the decoder fixed, the model is forced to learn the same latent state for the real and simulated image, since both VAEs need to generate the same simulation image as output image.

4.2 Reinforcement Learning (RL)

Reinforcement Learning is used to let the robot learn a task in such a way that the robot is capable of dealing with a changing environment. As a representative experiment, we use a cup-reaching task and change the cup position. Given the current joint values and the latent state the robot has to learn what its joint values should be to perform its task. Thus, the input state is the latent state concatenated with the current joint values, and the actions are the desired joint values as shown in Fig. 4.5. Since the action space is continuous, a policy gradient method is used. Specifically, Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015], since this algorithm has shown to be successful for robotic applications [Gu et al., 2017] [Pinto et al., 2017] [Tai et al., 2017].

4.2.1 Deep Deterministic Policy Gradient (DDPG)

Deterministic Policy Gradient (DPG) [Silver et al., 2014] models the policy as a deterministic decision, meaning that each state can result in only one action. In this manner, less samples are required to learn a policy, but there is no guarantee on adequate exploration. To ensure enough

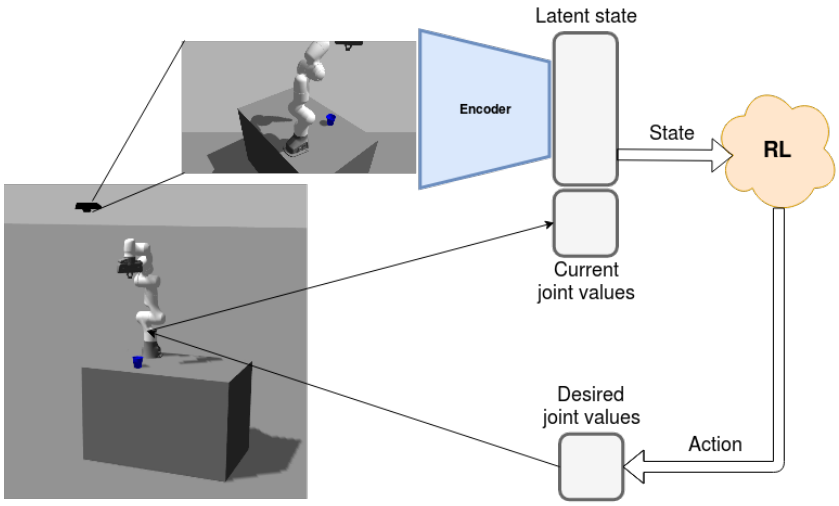


Figure 4.5: An overview of Reinforcement Learning using a VAE: the input for the RL algorithm are the latent state, obtained using the encoder of the VAE, concatenated with the current joint values. The actions that the agent should learn are the desired joint values it should move to.

exploration, an off-policy Actor-Critic is used that learns a deterministic target policy while following a stochastic behaviour policy. Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015] extends the DPG algorithm to be able to handle deep networks by adding experience replay and a target network as Mnih et al. [2015] do for Deep Q-Learning. Furthermore, batch normalisation was added to make it more stable, an off-policy algorithm with some noise was used to handle exploration and action repeats were added to be able to infer velocities.

The resulting DDPG algorithm is not guaranteed to converge, since non-linear function approximators are used, but experimental results in [Lillicrap et al., 2015] demonstrate that stable learning is possible. The largest drawback of DDPG is, like most model-free algorithms, the need of a large number of training episodes. However, DDPG can perform more efficiently than other policy gradient methods due to the use of a deterministic policy and is therefore more suitable for higher dimensions.

4.2.2 Learning from Demonstrations

Instead of letting the robot learn from performing random actions, expert demonstrations [Zhang and Ma, 2018] are used to speed up the learning process. Expert demonstrations are example demonstrations that show how the task can be performed. In our case, these are examples of the arm reaching the cup. In simulation, these examples can be easily obtained, since the position of the cup can be accessed by the simulator and a motion planner can be used to compute the desired joint values. The expert demonstrations are recorded by letting the robot move to the position of the cup while observing the new state and obtaining the reward for that action from the simulator. The policy is pretrained on samples from these demonstrations to enable it to start with useful actions instead of random actions.

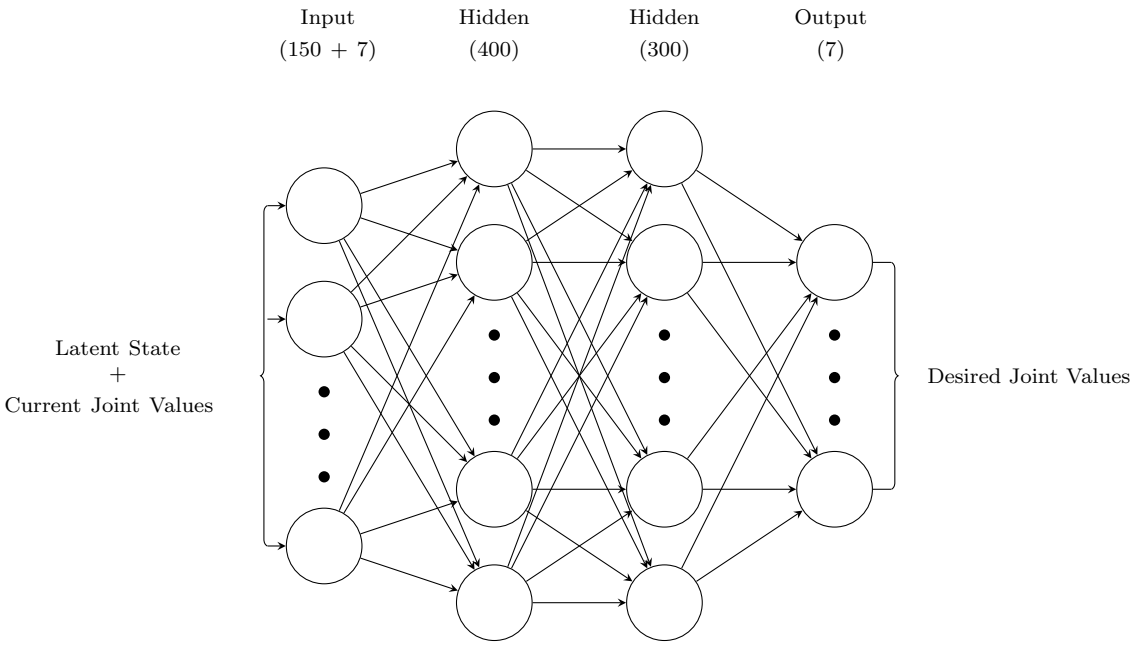
4.2.3 DDPG for Robot Control

In this thesis, the robot has to determine the joint values it has to move to, given the current joint values and the latent state obtained using the trained VAE. The input state of the algorithm is thus the latent state obtained using the VAE concatenated with the current joint values and the output actions are the joint values that the robot should move to. DDPG uses a normalised output between -1 and 1, thus the actions need to be normalised. This is done as follows:

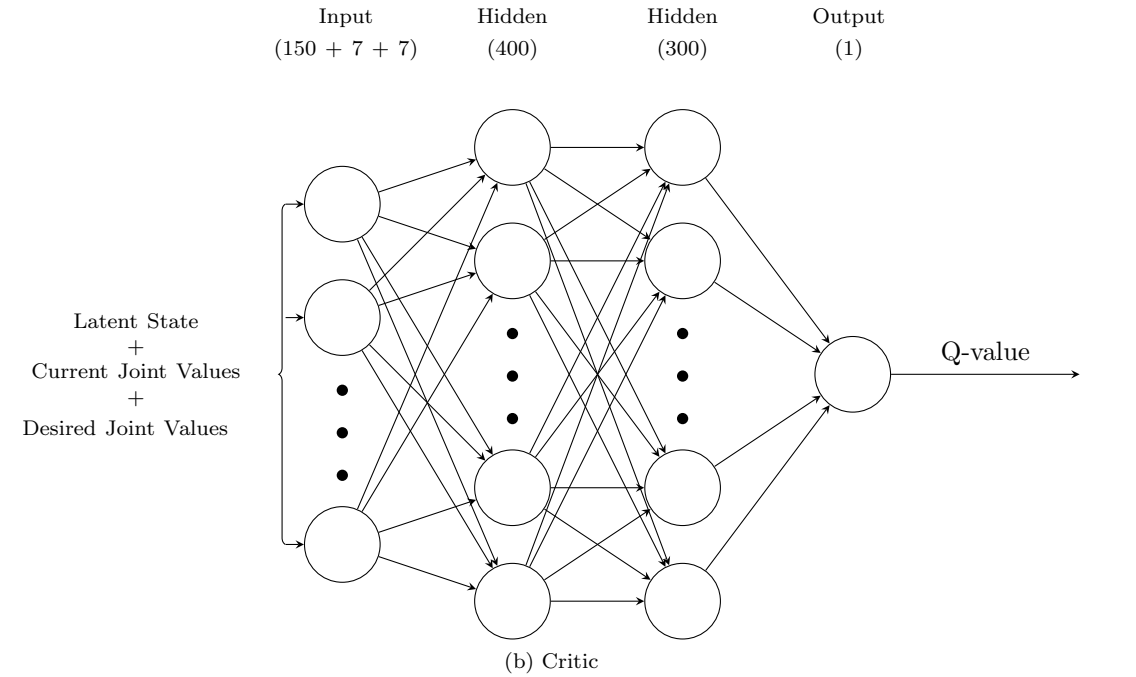
$$normalised\_joints = \frac{j + lower\_joint\_limits}{upper\_joint\_limits - lower\_joint\_limits} \times 2 - 1 \tag{4.1}$$

where j is the joint values, $lower\_joint\_limits$ are the lower limits of the joints and $upper\_joint\_limits$ are the upper limits of the joints.

The network architectures of the Actor and Critic are similar to the networks used in the original DDPG paper [Lillicrap et al., 2015], since they argue that these networks are capable of learning more than 20 different physics tasks. Both the Actor and Critic are Multi Layer Perceptrons (MLP) with two hidden layers of 400 and 300 units with layer normalisation and a Rectified Linear Unit (`ReLU`) [Nair and Hinton, 2010] as activation function. The input layer of the Actor is a fully connected layer with the size of the input state and the output layer is a fully connected layer with a `tanh` activation function that predicts the (normalised) joint values of the robot. The Critic has a fully connected layer with the size of the input state plus the actions as input layer. As output layer, it uses a fully connected layer without any activation function, since it predicts the Q-value for the state-action pair. An overview of the two networks is shown in Fig. 4.6.



(a) Actor



(b) Critic

Figure 4.6: The network architectures of the Actor (a) and Critic (b). The Actor has the current state as input and is used to predict the desired joint values. The Critic has the current state concatenated with the action as input and is used to compute the expected Q-value given the state-action pair.

Chapter 5

Experimental Setup

This chapter describes the experimental setup that was used to perform the experiments. Firstly, the hardware, such as the robot and camera, is described in Sec. 5.1, followed by an explanation of the implementation in Sec. 5.2.

5.1 Hardware

The *Franka Panda*[1] with seven joints (7 DOF) and a parallel gripper is used as robot arm. The robot arm can carry a 3kg payload and a maximum reach of 850mm. The limitations of the joints can be found in Tab. 5.1 and the work space of the arm is shown in Fig. 5.1. The Kinect 2[2] is used as camera, which provides RGB and depth images with a resolution of 960×540. All experiments are performed on a Linux PC (Ubuntu 16.04) with an Intel Core i7-4790 processor, 32GB RAM, and an NVIDIA GeForce GTX TITAN X 12GB graphics card.

5.2 Software Implementation

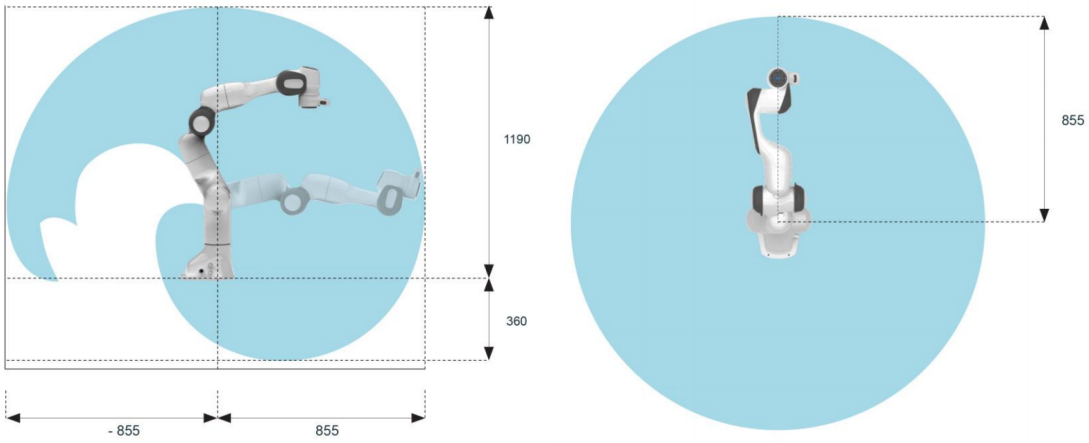
5.2.1 Variational Auto Encoder (VAE)

The VAE is implemented in Python 3.5 using Keras 2.2.4 [Chollet et al., 2015] and Tensorflow 1.13.1 [Abadi et al., 2015]. The implementation of the VAE is based on the example from Keras.[3]

	A1	A2	A3	A4	A5	A6	A7
Position (°)	-166/166	-101/101	-166/166	-176/-4	-166/166	-1/215	-166/166
Velocity (°/s)	150	150	150	150	180	180	180

Table 5.1: Joint limits of the Franka Panda.

[1] `https://www.franka.de/panda/`
[2] `https://developer.microsoft.com/en-us/windows/kinect`
[3] `https://blog.keras.io/building-autoencoders-in-keras.html`



(a) Side view (b) Top view

Figure 5.1: The workspace of the Franka Panda arm with all dimensions in mm obtained from the data sheet[4]of the Franka Panda.

5.2.2 Robot Operating System (ROS)

To interact with the robot, the Robot Operating System (ROS) [Quigley et al., 2009] is used. ROS is a framework for writing robot software and contains a lot of tools and libraries, such as MoveIt [Sucan and Chitta, 2013] for motion planning. The experiments in this thesis use ROS Kinetic on Ubuntu 16.04 and a simulation of the real world is created in Gazebo 7.0 [Koenig and Howard, 2004]. ROS is made for Python 2, so most of the nodes use Python 2.7. However, the implementation for DDPG is written for Python 3, so the node that handles the RL part is started from a virtual environment[5] with Python 3.5. Figure 5.2 shows an overview of the ROS nodes that are used and each node is explained below.

Gazebo / robot + Kinect: provides the data of either the simulation (Gazebo) or the real robot. This data consists of an image of the scene, using a Kinect, and the current joint values from which the transform of the arm can be computed. To obtain images from the real Kinect, IAI Kinect2 [Wiedemeyer, 2014 – 2015] is used. This is a package that reads the data from the Kinect and publishes it as a message within ROS. Furthermore, the simulated or real robot performs an obtained joint trajectory. In simulation, the robot moves as fast as the simulation can handle, which results in a speed up of 4×.

compute_end_pose: is used to compute the pose of the end effector in real world coordinates to be able to give a reward based on the distance between the current pose and the desired pose. To compute this pose, the transform of the arm is needed, which is provided by Gazebo or the real robot. Since the end pose is only used to compute the reward, this node is only needed during training, and not during testing.

[4]`https://s3-eu-central-1.amazonaws.com/franka-de-uploads-staging/uploads/2018/05/2018-05-datasheet-panda.pdf`

[5]`https://virtualenv.pypa.io/en/latest/`

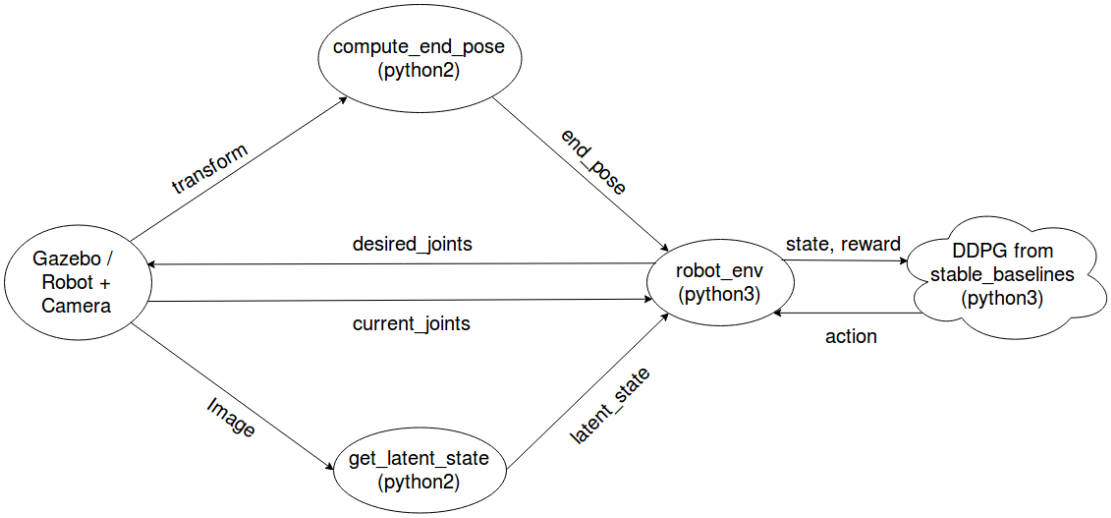


Figure 5.2: An overview of the ROS nodes that are used to perform RL with the latent state obtained using the VAEs together with the current joint values as input state.

get_latent_state: encodes the image obtained from the Kinect into the latent state using the trained VAE.

robot_env: obtains all information needed to perform Reinforcement Learning, such as the current joint values and the latent state for the input state and the end pose for computing the reward. It is implemented as an environment class which contains the same functions as an environment from OpenAI Gym [Brockman et al., 2016] in order to easily interact with different RL algorithms from stable baselines. This node interacts with the DDPG algorithm that computes the next action, which is then published as desired joint values. During testing, the trained model is used to compute the next action and it is not needed to compute the reward.

DDPG: is not a node itself, but is started within robot_env. This part uses the DDPG implementation from stable baselines [Hill et al., 2018] to learn a policy. Given the state and reward obtained by `robot_env`, the policy is updated and the computed action is returned to `robot_env`. During testing, the trained policy is used to compute the next action. This part is also used to collect expert demonstrations and pretrain the network with those demonstrations. Collecting the demonstrations is done using the `generate_expert_traj` function from stable baselines and pretraining the network uses the `pretrain` function.

Chapter 6

VAE Exploration

This chapter describes the experiments that have been performed for studying the performance of the Variational Auto Encoder (VAE) and their results. The experiments and their results are discussed in Sec. 6.1, Sec. 6.2 and Sec. 6.3, followed by a conclusion about the performance of the VAE in Sec. 6.4.

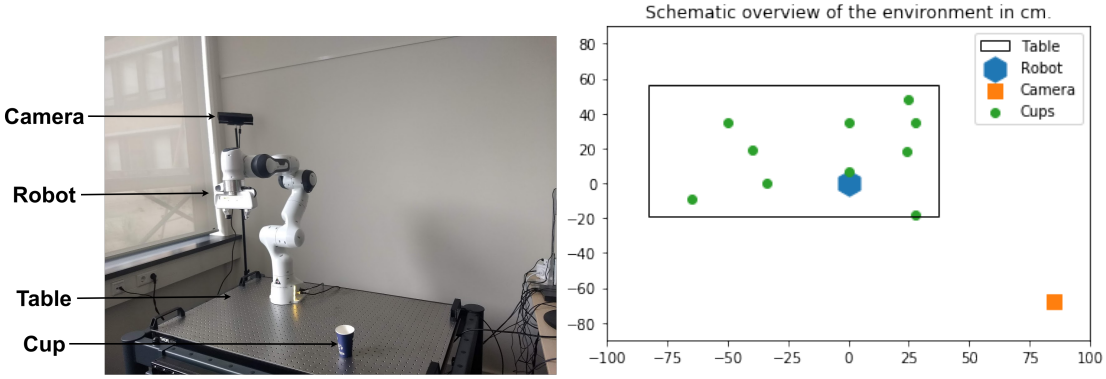
6.1 Entire Scene

In this experiment, the robot is placed on a metal table on which a blue paper cup is placed as shown in Fig. 6.1. The camera is situated in a corner in order to capture the entire robot and table. This setup represents a realistic scene that can be used in daily life situations and will be referred to as *entire scene*, since the robot, table, cup and background are all visible in the camera images. Ten cup positions are defined to record the real world data set and to collect expert demonstrations.

Firstly, the optimal latent dimension size is determined by training the simulation VAE with different latent dimension sizes $(10, 50, 100, 150, 250, 500)$. The VAEs are trained for 50 epochs, with a learning rate of 0.001 and a batch size of 64. The input image is resized to 467×268 and a subset of the data (3000 images) is used to perform this experiment to speed up the training time. After having found the optimal latent dimension size, the final VAEs that will be used to obtain the latent state for the RL part are trained for 300 epochs, with a learning rate of 0.001 and a batch size of 64. Again, the input images are resized to 467×268. This experiment is used to show that a common latent space can be learned using VAEs. We expect that the real world encoder can reconstruct the simulation image given an image from the real world, while the simulation encoder is not capable of doing this.

6.1.1 Data set

The VAEs are trained on a recorded data set. For the simulation VAE, only data generated in simulation is needed. This data is obtained by letting the robot and cup move randomly in simulation while saving images taken by the simulated Kinect. The real world VAE needs image pairs of the simulated and real world which are obtained by saving the joint values and the position of the cup while recording a real world data set. The same data can now be collected in simulation by performing the same movements and putting the cup at the same positions. Example image pairs are shown in Fig. 6.2 and the amount of data used per VAE for both the simulated and the real world can be found in Tab. 6.1.



(a) (b)

Figure 6.1: The environment in which the robot has to perform tasks. The robot is placed on a metal table on which a blue paper cup is placed. The camera is situated in a corner in order to capture the entire robot and table. Ten cup positions are defined to record the real world data set and to collect expert demonstrations.

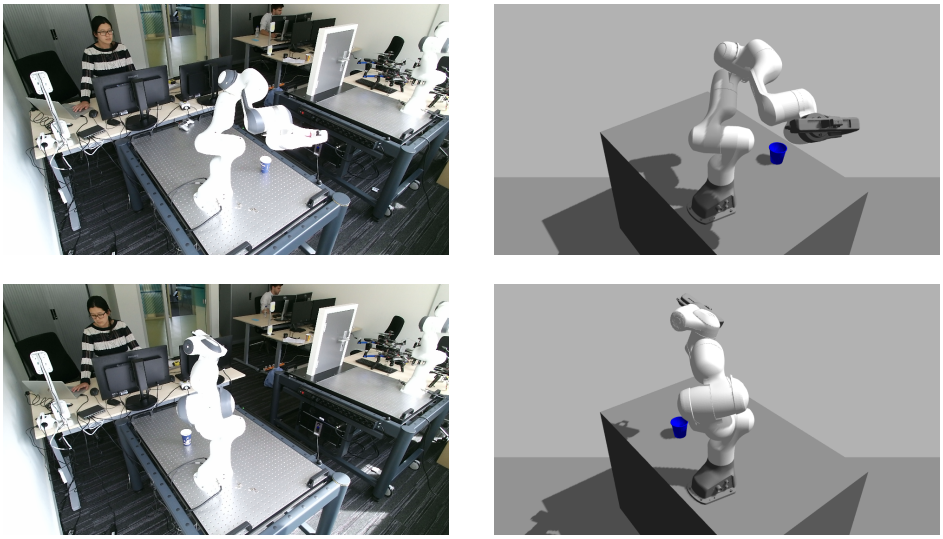


Figure 6.2: Example image pairs with the same state from the real (left) and simulated world (right) for the entire scene.

	Simulation	Real World
Simulation VAE	26,000	-
Real World VAE	120	120

Table 6.1: The amount of data needed to train the simulation and real world VAEs for the entire scene.

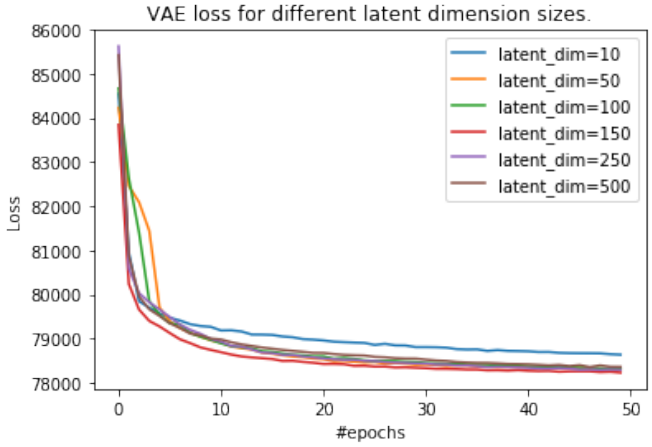


Figure 6.3: Loss per epoch for different latent dimension sizes trained on data from the entire scene.

6.1.2 Results

The results of training the simulation VAE with different latent dimensions is shown in Fig. 6.3. No big difference can be observed between the different sizes, but since the latent dimension of 150 slightly outperforms the other dimension sizes, a latent dimension of 150 will be used in the other experiments.

Figure 6.4 shows some images generated with the VAEs on the test set from the same real and simulated state. As can be seen, the simulation VAE is able to reconstruct the simulated image given a simulated image, and the real world VAE is capable of reconstructing the simulated image given a real world image. In contrast, the simulation VAE is not capable of reconstructing the simulated image given a real world image. The VAEs are able to reconstruct the table perfectly, but the reconstruction of the cup and arm are noisy. However, the cup is always reconstructed at the correct position, thus useful information is still available in the latent state. For our experiments, a perfect reconstruction of the arm is less important, since the joints of the arm are available as well. Thus these reconstructions suggest to be sufficient to contain all information needed to learn a good policy. The average difference between the generated images from simulation and the real world is 7512.59, which is an average difference of 0.06 per pixel in the range from $0 - 1$. The average difference between the generated images is computed as follows:

$$\frac{1}{N}\sum_{n}^{N}\sum_{p}^{P}|\text{generated\_sim\_img}_p^n - \text{generated\_real\_img}_p^n|, \tag{6.1}$$

where N is the amount of images and P is the amount of pixels in an image (width $\times$ height). Similarly, the average difference per pixel is computed as:

$$\frac{1}{N}\sum_{n}^{N}\frac{\sum_{p}^{P}|\text{generated\_sim\_img}_p^n - \text{generated\_real\_img}_p^n|}{P}, \tag{6.2}$$

where N is again the amount of images, and P is the amount of pixels in an image (width $\times$ height).

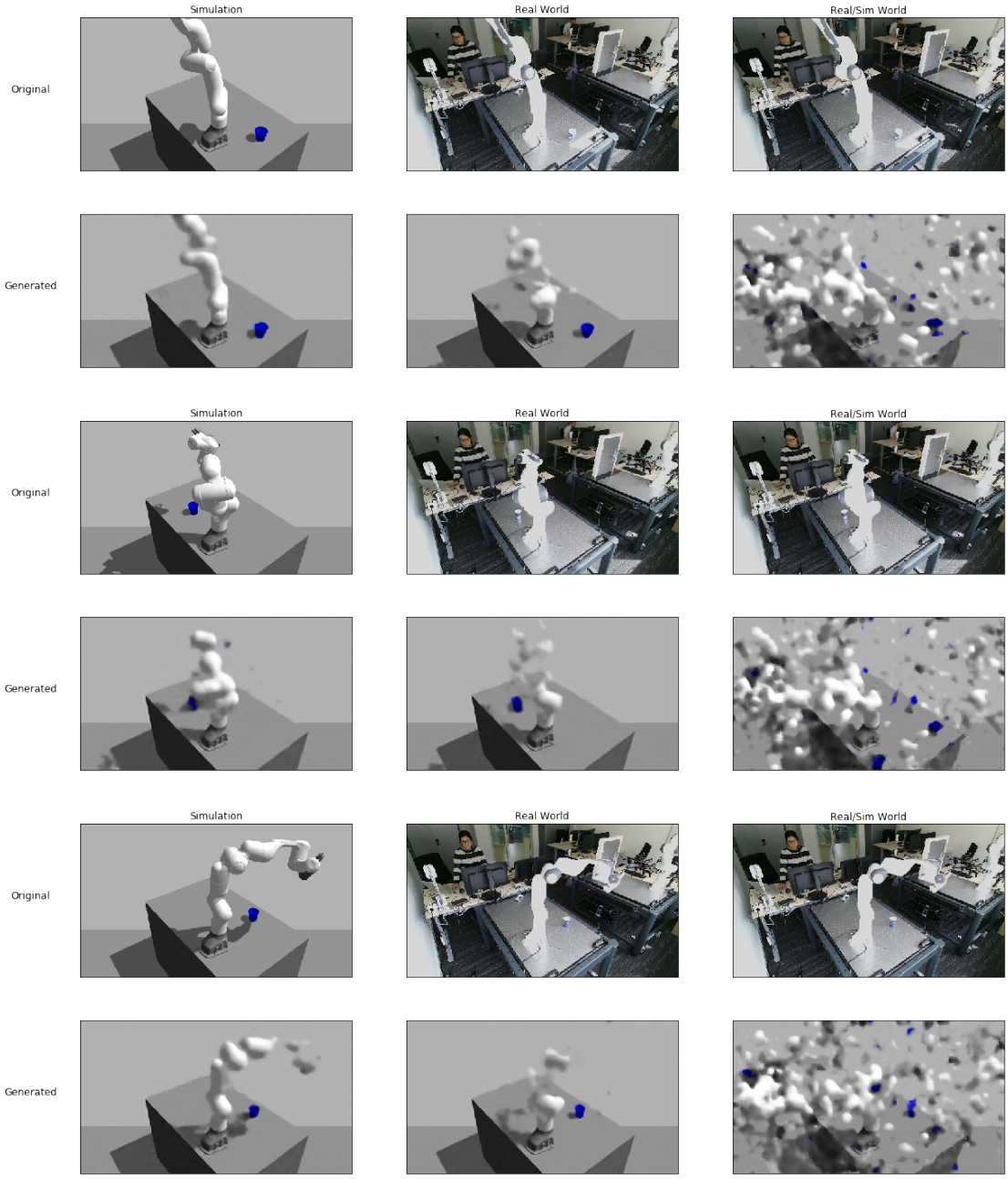
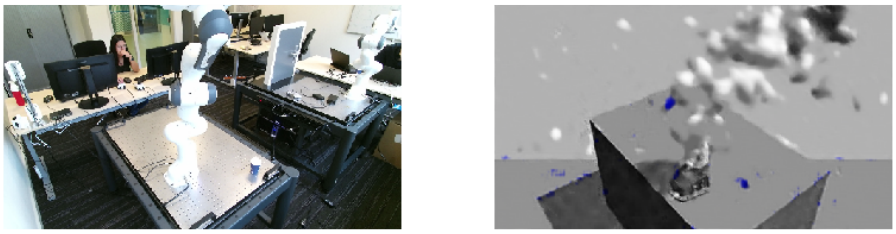


Figure 6.4: Results of the VAEs on the real and simulated images. The simulation VAE is able to reconstruct the simulated image given a simulated image, and the real world VAE is capable of reconstructing the simulated image given a real world image. In contrast, the simulation VAE is not capable of reconstructing the simulated image, given a real world image.



(a) Real world (b) Reconstructed image

Figure 6.5: Reconstructed image (b) of the real world (a) with the model trained on data from the first day, but tested on a new day. The VAE is not capable of reconstructing the scene correctly.

(a) Real world (b) Reconstructed image

Figure 6.6: Reconstructed image (b) of the real world (a) with the model trained on data from three different days, but tested on a new day. The VAE is not capable of reconstructing the scene correctly.

VAE on Data from another Day

The results of the VAE look promising on the collected test data set. However, when testing it again in the real world environment on another day, the VAE was not capable of reconstructing the scene as shown in Fig. 6.5. This might be explained by the limited amount of real world data from only one specific day. However, when trained on data from three different days, it is still not able to correctly reconstruct images from a new day as shown in Fig. 6.6. In contrast, when trained on data of the same day in the morning and tested on data from the afternoon, some useful reconstructions were made, but it always reconstructs the cup on the same position (Fig. 6.7). This suggests that the images vary too much between different days and no generalisation can be learned from this (limited) data set. This variation can be in the different backgrounds, since that slightly changed from day to day, for example due to wearing a different shirt. Figure 6.8 shows an example of adding a black box to the background of the image. The VAE fails to reconstruct the simulated image. This suggests that a small change in the background effects the reconstruction as well, thus a small difference in the background between the data set on which the VAE is trained and the tested image, can cause a large failure in the reconstruction.

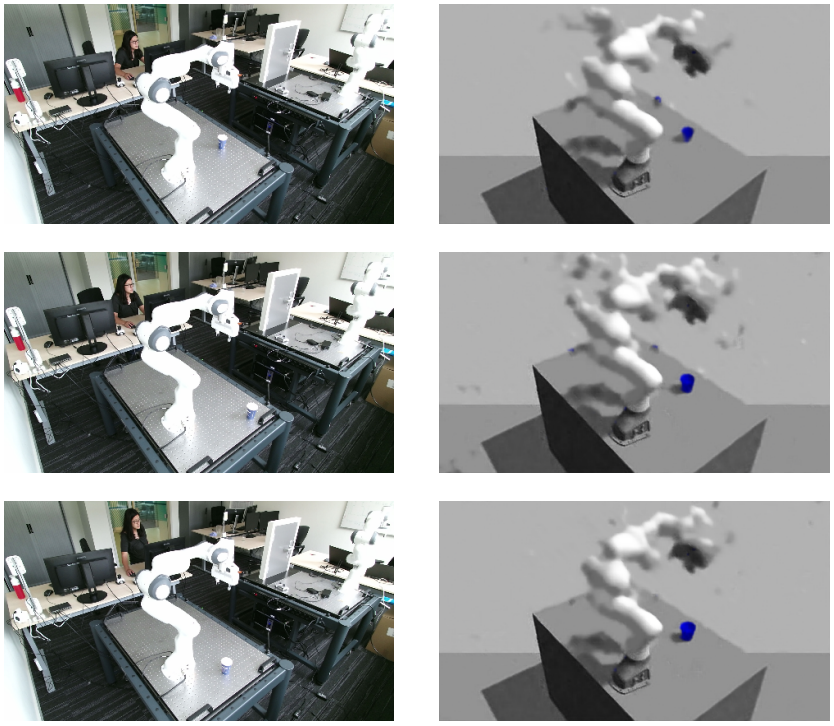


Figure 6.7: Reconstructed images (right) of the real world (left) with the model trained on data from the same day. When collecting data on the same day, the VAE seems to be able to reconstruct the scene correctly. However, it has not learned to reconstruct the cup at the correct position.

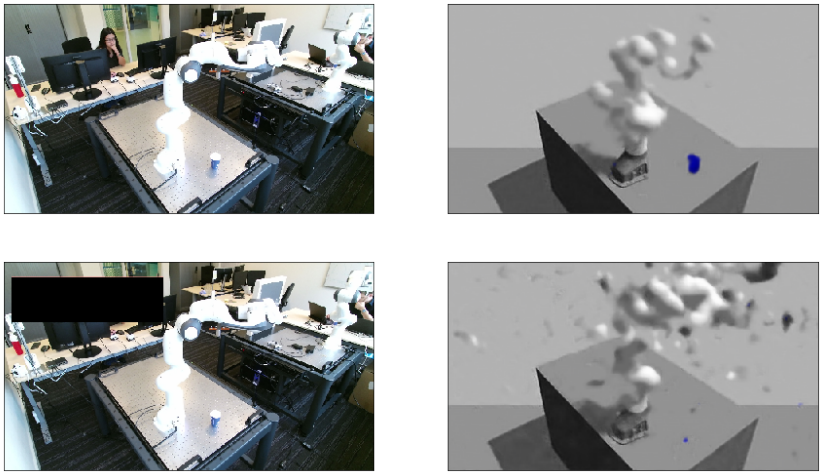
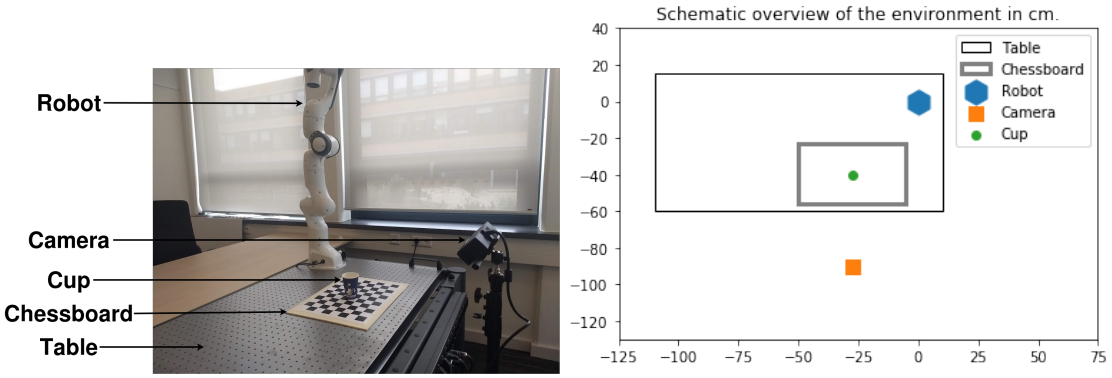


Figure 6.8: Example of adding a black box to the background of the image. Without adding the black box a normal reconstruction can be made (top), while adding a black box in the background causes the reconstruction to fail (bottom).



(a) (b)

Figure 6.9: The second scene in which the robot has to perform tasks. The robot is placed on a metal table on which a chessboard and a blue paper cup are placed. The camera is situated in front of the table, such that only the chessboard and the cup are visible in the obtained images.

6.2 Scene with Table and Cup

The results from the previous experiments suggest that a small difference in the background between the data set on which the VAE is trained and the tested image, can cause a large failure in the reconstruction. To exclude this component, the next experiment uses a scene in which the camera is positioned in such a way that only the table and the cup are visible as shown in Fig. 6.9. To make recording a data set easier, a chessboard is placed on top of the table, to be able to determine the position of the cup more accurately. The VAE should now only learn to reconstruct the table, the chessboard and the cup, and not a moving robot or (changing) background. We expect that the VAE is able to reconstruct both the test images and images from the same scene obtained on another day, since the background cannot influence the reconstruction anymore. The VAEs are trained for 300 epochs on images that were resized to 467×268, with a latent dimension size of 150, a learning rate of 0.001 and a batch size of 64.

6.2.1 Data Set

The VAEs are trained on a recorded data set. In contrast to the data set for the entire scene, no robot is needed to collect this data set. For the simulation VAE, the data is obtained by randomly moving the cup over the chessboard while saving images taken by the simulated Kinect. The real world VAE needs image pairs of the simulated and real world which are obtained by saving the position of the cup while recording a real world data set. The same data can now be collected in simulation by putting the cup at the same positions. Example image pairs are shown in Fig. 6.10 and the amount of data used per VAE for both the simulated and the real world can be found in Tab. 6.2.

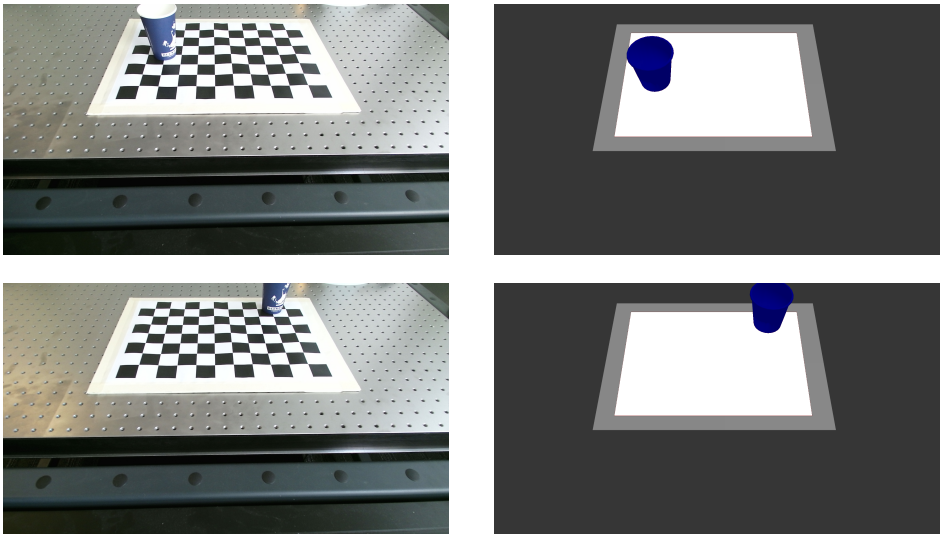


Figure 6.10: Example image pairs with the same state from the real (left) and simulated world (right) for the scene with only the table and the cup.

	Simulation	Real World
Simulation VAE	5,000	-
Real World VAE	70	70

Table 6.2: The amount of data needed to train the simulation and real world VAEs for the scene with only the table and the cup.

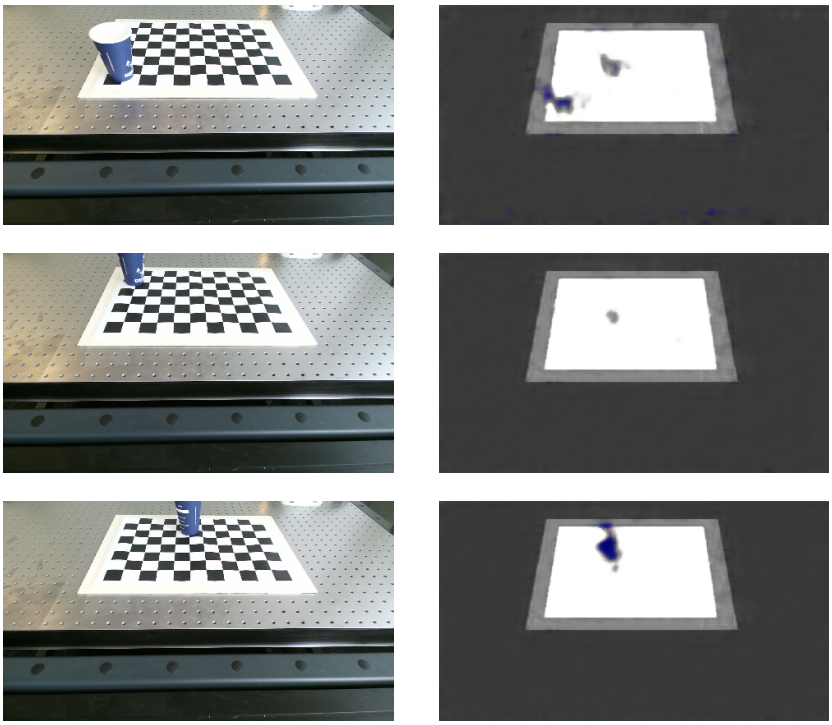


Figure 6.11: Reconstructed images (right) of the real world (left) with the model trained on data from the first day, but tested on a new day. The VAE is not capable of reconstructing the scene correctly.

6.2.2 Results

Figure 6.12 shows some images generated with the VAEs on the test set from the same real and simulated state. Again, the simulated VAE is able to reconstruct the simulated image given a simulated image, and the real world VAE is capable of reconstructing the simulated image given a real world image. The VAEs are able to reconstruct all components of the scene perfectly, in contrast to the VAE of the entire scene. Furthermore, the simulation VAE, given a real world image, is able to reconstruct the scene, but not the correct position of the cup. This means that the decoder obtains all necessary information about the scene, and that only the position of the cup should be obtained from the latent space. The average difference between the generated images from simulation and the real world is 9366.25, which is an average difference of 0.07 per pixel in the range from $0 - 1$.

VAE on Data from another Day

Again, the results of the VAE look promising on the collected test data set, but testing it on another day in the real world environment resulted in imperfect reconstructions as shown in Fig. 6.11. The images from two different days still vary too much, which can be explained by the various lighting conditions on different days. Furthermore, the metal table reflects differently depending on the lighting conditions and these small differences might already effect the results of the VAE.

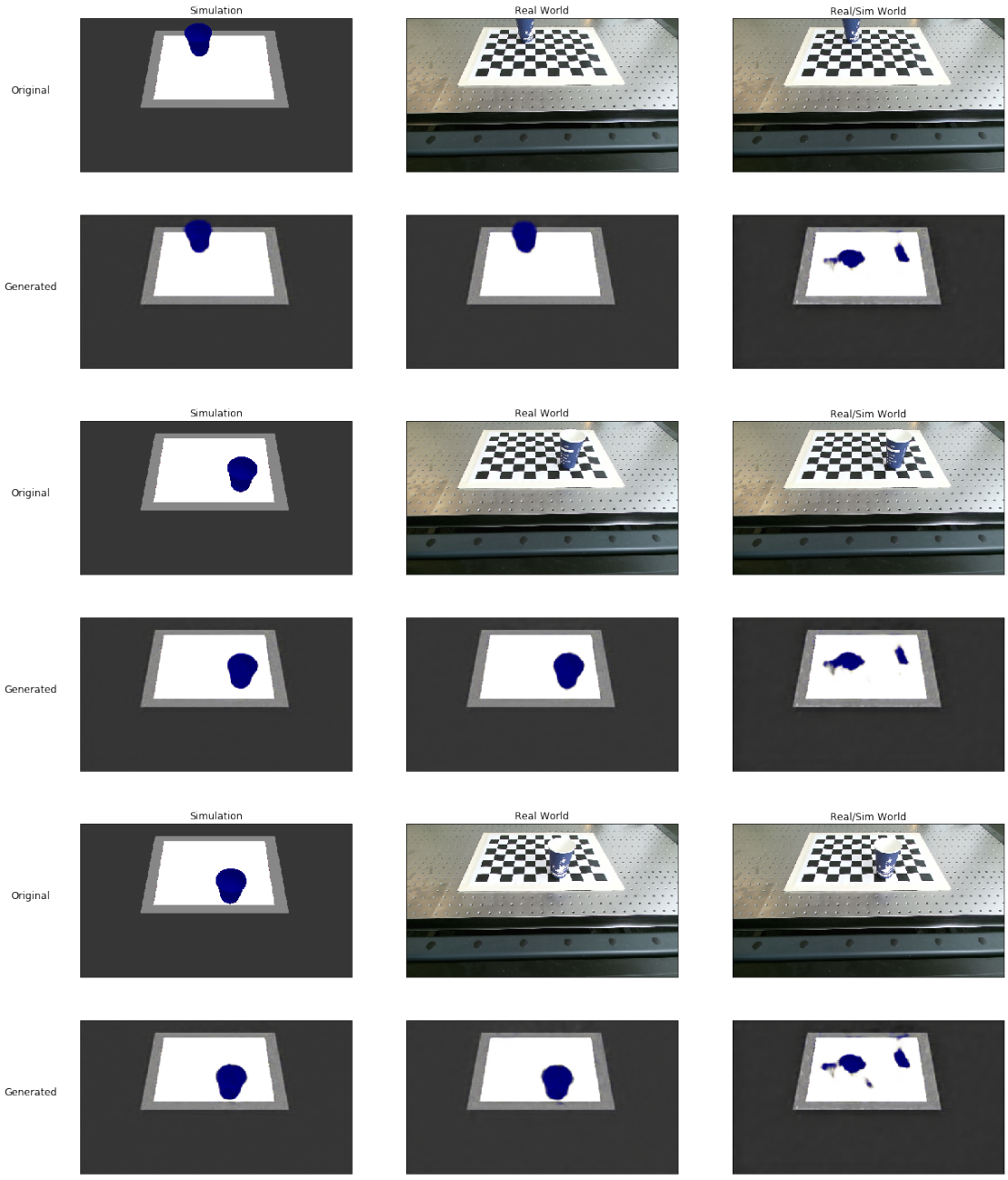
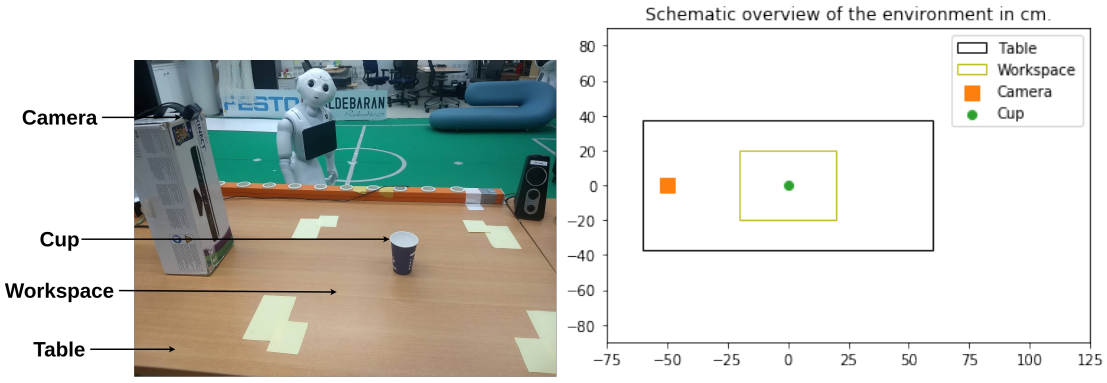


Figure 6.12: Results of the VAEs on the real and simulated images for the scene with only the table and the cup. The simulation VAE is able to reconstruct the simulated image given a simulated image, and the real world VAE is capable of reconstructing the simulated image given a real world image. In contrast, the simulation VAE is is can reconstruct the scene, but is not capable of reconstructing the cup on the right position, given a real world image.



(a) (b)

Figure 6.13: The third environment with stable lighting. No robot arm is present and the blue paper cup is placed on a wooden table. The camera is situated in such a way that only the table and the cup are visible in the obtained images.

6.3 Scene with more Stable Lighting

In the last experiment, a different environment is used which is not right next to a window and does not have a metal table that reflects differently depending on the lighting conditions. In this environment, no robot is present and the blue paper cup is placed on a wooden table instead of on a metal table as shown in Fig. 6.13. Furthermore, during this experiment a Microsoft LifeCam HD-5000 with a resolution of 640×480 is used to collect the data set instead of a Kinect. We expect the VAE to be able to reconstruct both the test images and images from the same scene obtained on another day, since the background or lighting conditions cannot influence the reconstruction anymore. The VAEs are again trained for 300 epochs on images that were resized to 467×268, with a latent dimension size of 150, a learning rate of 0.001 and a batch size of 64.

6.3.1 Data Set

Similarly to the data set for the scene with the table and cup, no robot is needed to collect this data set. For the simulation VAE, the data is obtained by randomly moving the cup over the table while saving images taken by the simulated Kinect. The real world VAE needs image pairs of the simulated and real world which are obtained by saving the position of the cup while recording a real world data set. The same data can now be collected in simulation by putting the cup at the same positions. Example image pairs are shown in Fig. 6.14 and the amount of data used per VAE for both the simulated and the real world can be found in Tab. 6.3.

	Simulation	Real World
Simulation VAE	2,500	-
Real World VAE	125	125

Table 6.3: The amount of data needed to train the simulation and real world VAEs for the scene with only the table and the cup and stable lighting.

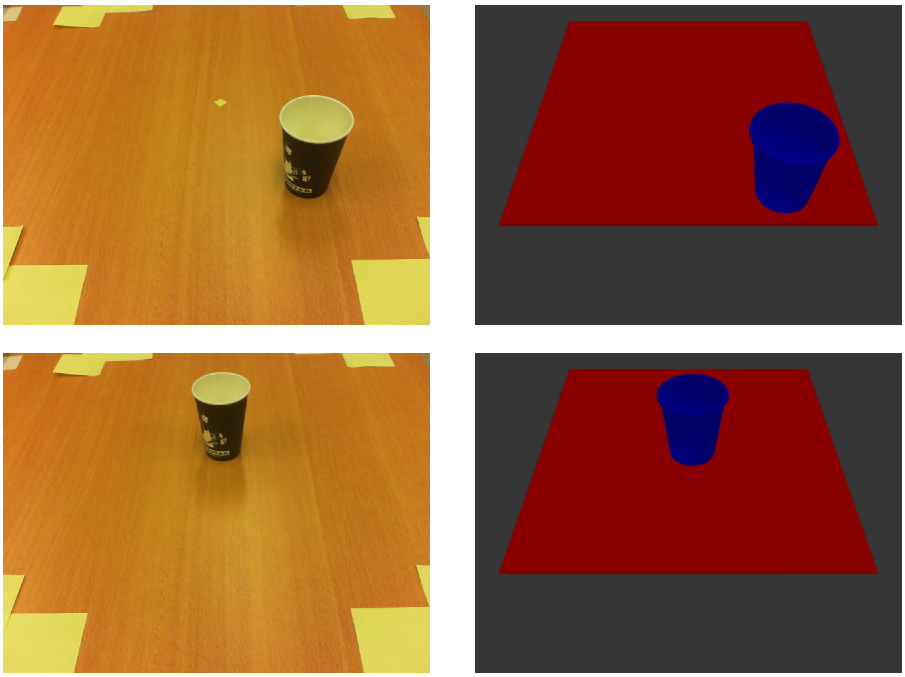


Figure 6.14: Example image pairs with the same state from the real (left) and simulated world (right) for the scene with stable lighting.

6.3.2 Results

Figure 6.15 shows some images generated with the VAEs on the test set from the same real and simulated state. Again, the simulated VAE is able to reconstruct the simulated image given a simulated image, and the real world VAE is capable of reconstructing the simulated image given a real world image. The VAEs are able to reconstruct all components of the scene perfectly, in contrast to the VAE of the entire scene. The average difference between the generated images from simulation and the real world is 791.14, which is an average difference of 0.006 per pixel in the range from $0 - 1$.

VAE on Data from another Day

Figure 6.16 shows the results of testing the VAE on another day in the real world environment. In contrast to the other scenes, the VAE is capable of reconstructing the simulation images on another day when the lighting conditions are kept the same and no metal table that reflects differently depending on the lighting conditions is used.

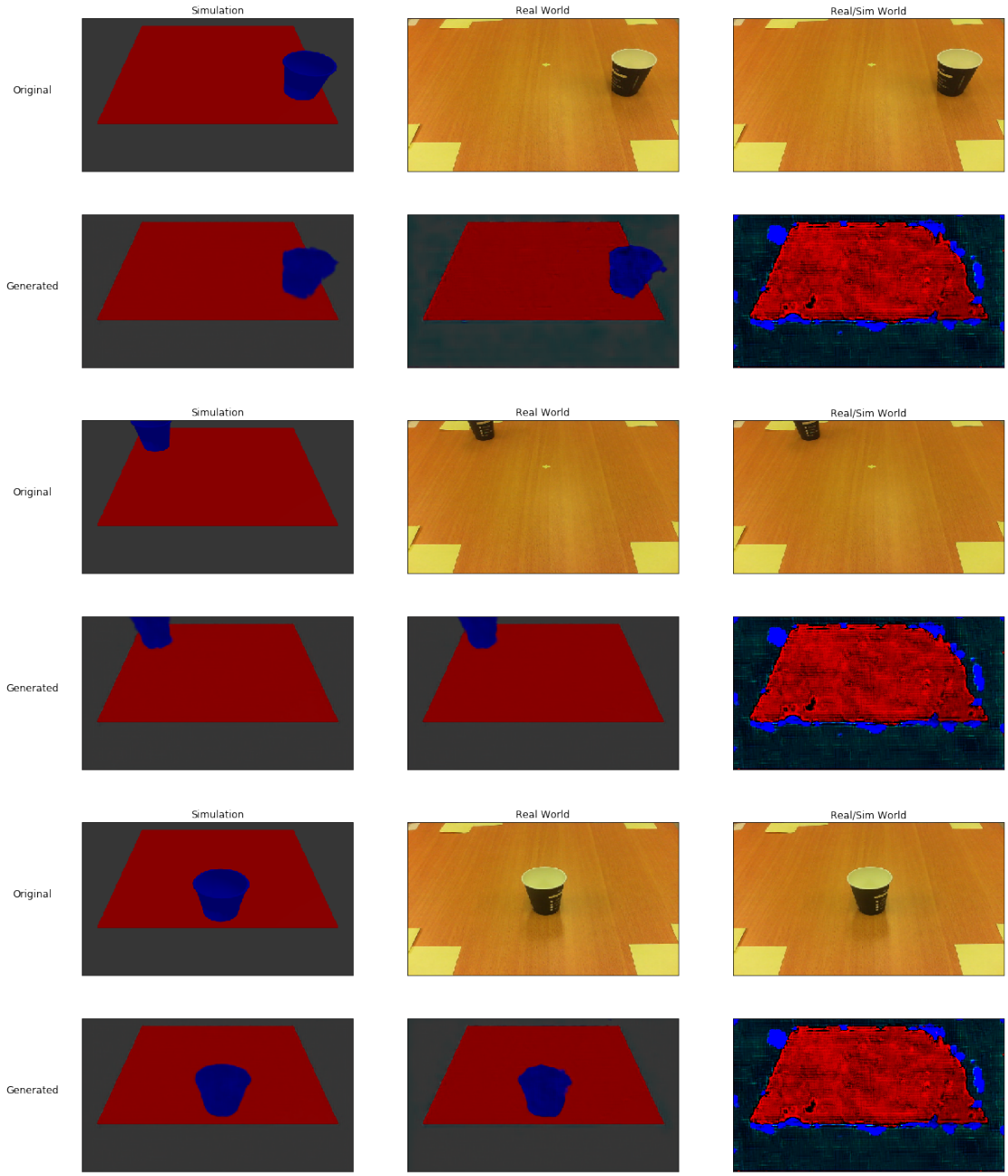


Figure 6.15: Results of the VAEs on the real and simulated images for the scene with stable lighting. The simulated VAE is able to reconstruct the simulated image given a simulated image, and the real world VAE is capable of reconstructing the simulated image given a real world image. In contrast, the simulated VAE is not capable of reconstructing the simulated image, given a real world image.

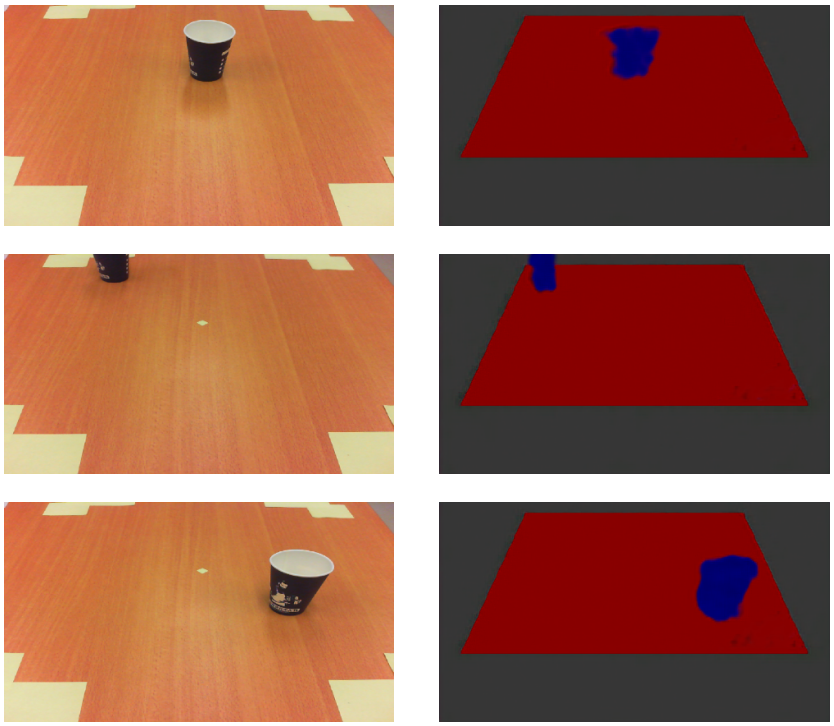


Figure 6.16: Reconstructed images (right) of the real world (left) with the model trained on data from the first day, but tested on a new day. The VAE is capable of reconstructing the scene correctly.

6.4 Conclusion

The VAEs are capable of learning a common representation for the simulated and real world if we look at the results on the test set. However, unless the lighting conditions are very similar, it does not generalise well to new real world data, which is obtained at a different moment than the recorded train and test set. We have tried to remove the aspects that influence these differences, such as the noise from the background by positioning the camera in such a way that only the table and the cup are visible, and the different lighting conditions by changing to an environment that is not next to a window and does not have a metal table that reflects the light differently.

Only removing the background was not sufficient to learn a model that generalises well to new real world data, but also changing the table and moving to another environment with more stable lighting conditions did help. To conclude, in a controlled environment in which the differences between images obtained at different moments are almost the same, VAEs are capable of learning a common representation for the simulated and real world. However, it is not easy to create these circumstances, especially for daily life situations.

Chapter 7

RL Experiments

This chapter describes the experiments and the results for performing Reinforcement Learning (RL) with the latent state obtained from the encoder of the VAE as input. Firstly, a detailed description of the task is given in Sec. 7.1. Secondly, the experiments and their results are discussed in Sec. 7.2 and Sec. 7.3. Finally, a conclusion about the performance of the VAE is drawn in Sec. 7.4.

7.1 RL to Reach a Cup

In this thesis, RL is used to let the robot learn tasks in simulation and perform them in the real world. The task that the robot has to learn is to reach a blue cup on top of a metal table from a fixed starting position as shown in Fig. 7.1. For this experiment, the entire scene is used, meaning that the camera is positioned in such a way that the robot arm, the table, the cup and some background is visible. Since the scope of this thesis is to show that the latent state can be used as input for RL, the task is kept simple by not taking collision with the table or itself into account. This is done by giving the table no collision in simulation, such that the robot can just hit the table without any consequences in simulation.

Since the task of this experiment is to reach a certain goal position, the reward function of this task is the negative distance between the end effector of the robot and the goal position:

$$reward = -||ee - goal||_2, \tag{7.1}$$

where ee is the position of the end effector and $goal$ the position of the cup.

Training the policy is done with a Ornstein-Uhlenbeck process [Uhlenbeck and Ornstein, 1930] as noise with $\mu = 0$ and $\sigma = 0.2$, and a memory limit of $1e6$. For all other parameters, the default values from Stable Baselines [Hill et al., 2018] are used.

The experiments are evaluated by the time to train in hours, the average distance (error) between the end effector of the robot and the centre of the cup in centimetres, and the accuracy, which is the percentage that the robot reached the cup within a range of 10cm. To compute the average error and the accuracy, the robot performs the learned policy 100 times. The policy is only tested in simulation and not in the real world, since the results of the VAE were not sufficient to reconstruct the scenes with the robot in the real world. Any tests with the policy in the real world would thus result in obtaining a random latent state as input, which the policy has probably not seen, resulting in unknown behaviours.

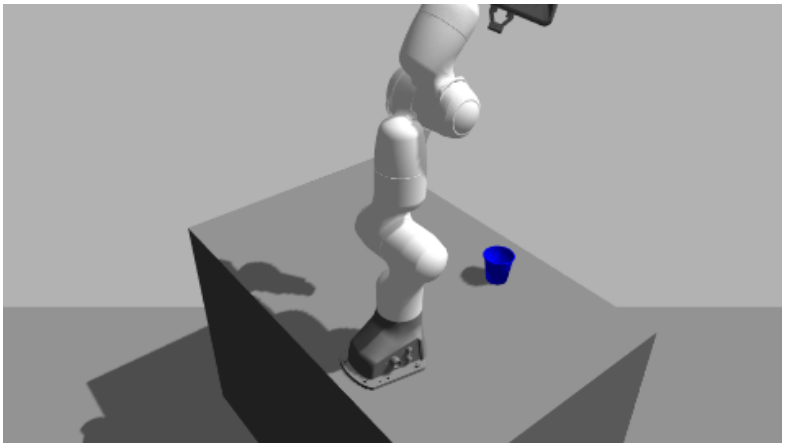


Figure 7.1: The starting position of the robot from which it should reach the cup.

7.2 Reach a Cup at a Fixed Position

In this experiment, the robot has to reach a cup that is placed at a fixed position: 35cm in front of the robot. This experiment is used to show that the setup for training the robot with RL is working and tries to answer the question: *Can the robot learn to go to a fixed target position, given the latent state and current joint values as input?* While the latent state is given as input, it does not actually need it. The robot should learn to always go to the same joint values. We expect that the robot is capable of learning this, since it only needs to predict the same actions as seen during demonstration. Firstly, the robot is trained for $1,000$ epochs on 10 expert demonstrations reaching the cup. Next, the robot continues to learn how to reach the cup using RL for $2,500$ steps.

7.2.1 Results

The results of reaching a cup at a fixed position in simulation are shown in Tab. 7.1. The imitation policy is only pretrained on the expert demonstrations and has thus not been trained using RL. As can be seen from the results, the RL policy outperforms the imitation policy for reaching a cup at a fixed position. Compared to using the position of the cup as input state instead of the latent state, the latent state performs slightly worse. This can be explained by the fact that the latent state can be slightly different each time, since a small adjustment in the captured image, for example the arm not being at the precise start position, might influence the latent state.

		Time to train (h)	Avg error (cm)	Accuracy (%)
With cup position	Imitation	< 0.1	**2.63**	98
	RL	1.27	3.34	**100**
With latent state	Imitation	< 0.1	5.37	89
	RL	1.37	**4.67**	**96**

Table 7.1: Results for reaching a cup at a fixed position in simulation. The RL policy outperforms the imitation policy. Having the cup position as input slightly outperforms the latent state.

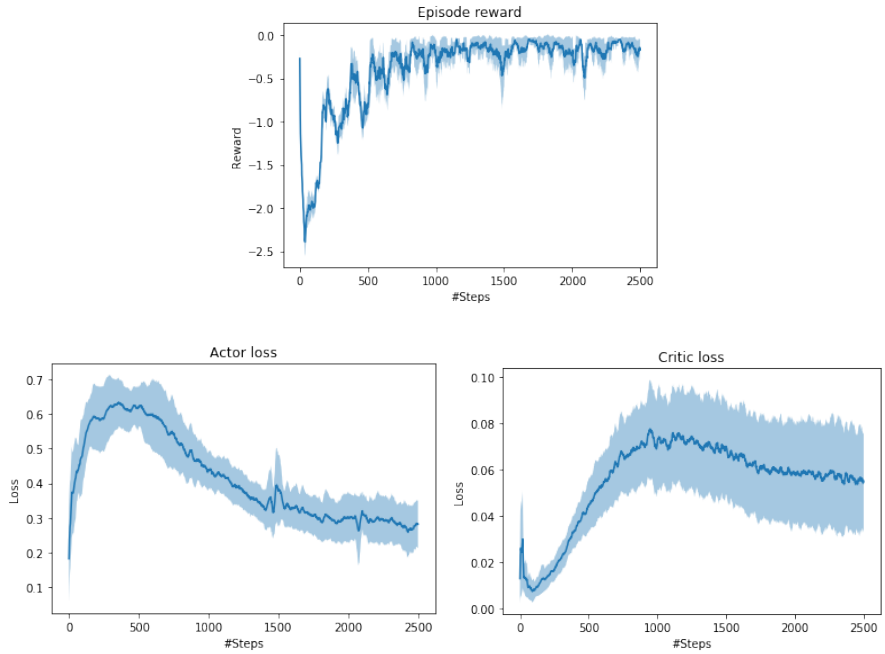


Figure 7.2: The episode reward and the actor/critic losses over time during training for reaching a cup at a fixed position averaged over three training runs. After 600 steps, the losses start to decrease and the episode reward gets more stable towards 0.

Figure 7.2 shows the episode reward, the actor loss and the critic loss over time during training for reaching a cup at a fixed position averaged over three training runs. As can be seen, the losses start increasing, since the robot starts to take actions with noise while initialised with the imitation policy. This can also be seen from the episode reward: the rewards are heavily fluctuating in the beginning. After 600 steps, the losses start to decrease and the episode reward gets more stable towards 0.

7.3 Reach a Cup at Ten Positions

In the next experiment, the robot has to reach a cup that is placed at ten different positions. It should now use the latent state for determining where the cup is placed to reach it correctly. Our hypothesis is that the latent state is a sufficient input for RL to determine where the cup is and to learn to move towards it. Firstly, the robot is trained for $5,000$ epochs on 50 expert demonstrations reaching the cup at the ten positions. Next, the robot continues to learn how to reach the cup using RL for $10,000$ steps.

7.3.1 Results

The results of reaching ten cups in simulation are shown in Tab. 7.2. In contrast to reaching a cup at a fixed position, the RL policy performs a lot worse than the imitation policy for reaching ten cups. However, this is the case for both the cup position and the latent state, so this is not due to the latent state, but due to the RL algorithm. Using the latent state as input even slightly outperforms using the cup position as input.

		Time to train (h)	Avg error (cm)	Accuracy (%)
With cup position	Imitation	< 0.1	**12.10**	**80**
	RL $(10,000)$	4.75	27.41	44
With latent state	Imitation	< 0.1	**12.48**	**82**
	RL $(10,000)$	3.53	28.48	58
	RL $(50,000)$	27.66	20.79	74

Table 7.2: Results for reaching ten cups in simulation. The imitation policy outperforms the RL policy, but using the latent state as input slightly outperforms the cup position.

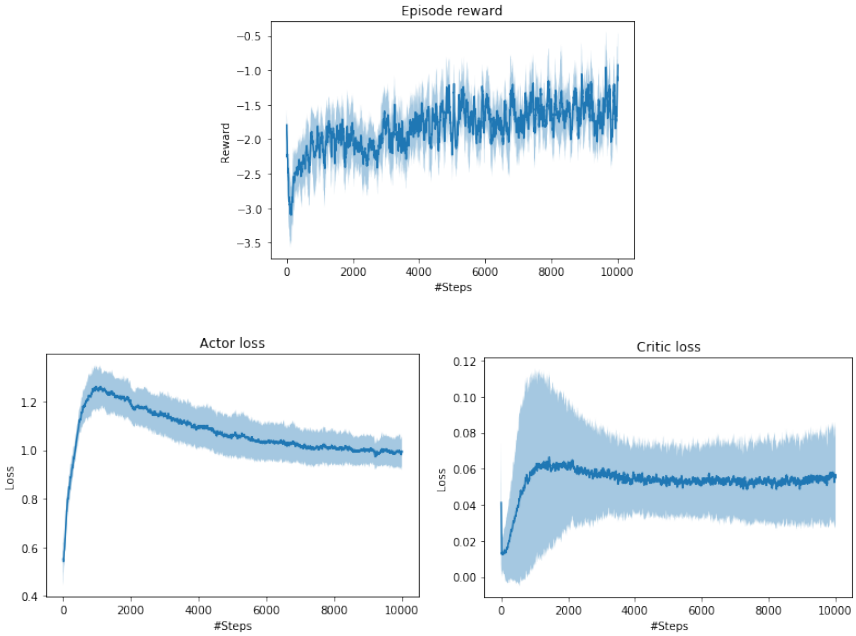


Figure 7.3: The episode reward and the actor/critic losses over time during training for reaching a cup at ten positions averaged over three training runs. Both the reward and the losses seem to be not yet at an optimum, suggesting that it can still learn more when training longer.

Figure 7.3 shows the episode reward, the actor loss and the critic loss over time during training for reaching a cup at ten positions averaged over three training runs. Similar to the training process for reaching one cup, the losses first start increasing before decreasing. However, the critic loss does not decrease yet and the actor loss and episode reward are not yet at an optimum. This can be explained by the fact that it has not had enough time to train yet. Figure 7.4 shows the episode reward, the actor loss and the critic loss over time during training one run for $50,000$ steps. As can be seen, after $50,000$ steps, the losses are lower compared to the $10,000$ steps training runs, but the reward is about the same. Despite the same reward, the policy that was trained for $50,000$ steps heavily outperforms the policy that was trained for $10,000$ steps as shown in Tab. 7.2. However, training this policy took almost 28 hours, compared to the 3.53 hours for $10,000$ steps.

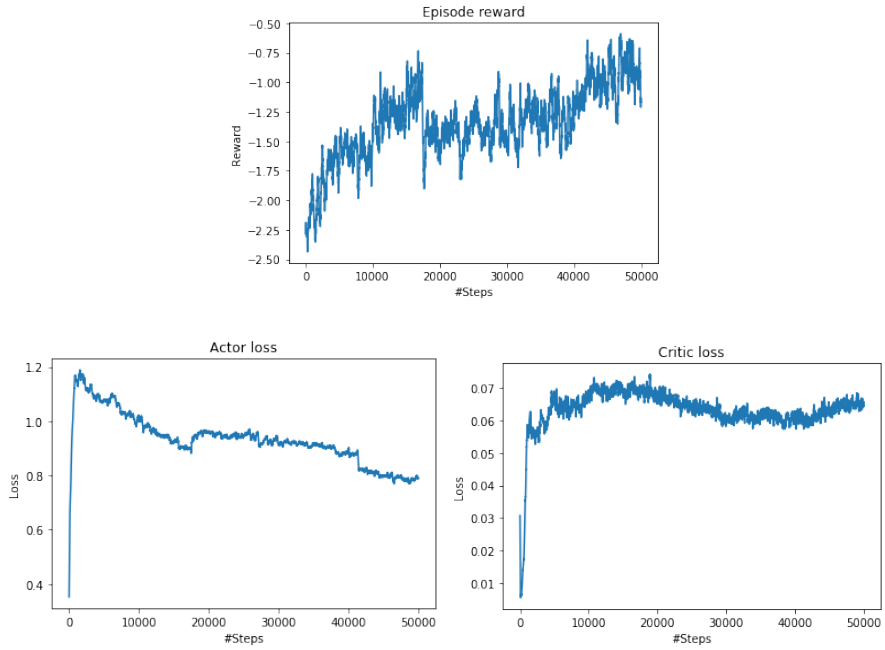


Figure 7.4: The episode reward and the actor/critic losses over time during training for reaching a cup at ten positions for $50,000$ steps. Both the reward and the losses seem to be not yet at an optimum, suggesting that it can still learn more when training longer.

7.4 Conclusion

The experiments have shown that the latent state obtained from the encoder of the VAE, can be used as input for RL. Compared to using the position of the cup directly, the performance of using the latent state as input is the same. This shows that all information needed to reach the cup, such as the position of the cup, is captured in the latent state. However, training a policy for reaching a cup takes a long time in our setting within ROS. Furthermore, the imitation policy still outperforms the RL policy for more complex tasks.

All these experiments have been tested in simulation and no tests with the real robot are performed. No conclusion can thus be drawn about the capability of using the latent state obtained using the encoder of a VAE to bridge the visual reality gap in order to transfer a policy trained in simulation to the real robot. However, we can conclude that the latent state can be used as input state for RL. Thus, if this latent state can be improved in such a way that the same states in simulation and the real world are consistently represented as the same latent state, experiments with transferring the policy trained in simulation to the real robot can be conducted.

Chapter 8

Discussion and Conclusion

Since the use of robots in real life situations is still limited and adjusting a robot to new situations can be time-consuming, we have decided to look into using Reinforcement Learning (RL) for robot learning. However, RL on a real robot can be time-consuming and even dangerous, thus learning in simulation could be used instead. Unfortunately, transferring a learned policy from the simulator to the real world does often not result in the same behaviour in the real world. Therefore, we looked into using Variational Auto Encoders (VAE) to bridge the visual reality gap in order to perform RL with robots. This chapter discusses the results of using VAEs to bridge the visual reality gap in Sec. 8.1. Next, recommendations for future work are provided in Sec. 8.2. Finally, a conclusion is drawn in Sec. 8.3.

8.1 Discussion

The conducted experiments have shown that in a controlled environment in which the differences between images obtained at different moments are almost the same, VAEs are capable of learning a common representation for the simulated and real world. However, it is not easy to create these circumstances, especially for daily life situations. The RL experiments have shown that the latent state, obtained using the encoder of the VAE, can be used as the input state for RL to learn a simple task, such as reaching a cup. However, training such a policy can take days while still being outperformed by the imitation policy. But, this is independent of the latent state, since training such a policy with the position of the cup as input instead of the latent state has the same performance and training time.

The experiments have also shown the limitations of using a VAE to obtain a common representation for the simulated and real world. Firstly, the conditions of the real world environment should stay the same, which is very difficult when both the workspace of the robot and the background are in the image, or when the lighting conditions vary. Furthermore, the reconstruction of the original image with the VAE is not perfect for the entire scene. Specifically, the arm was not always correctly reconstructed. This is not a problem for our experiments, since the current joint values can be obtained and the robot does not have to avoid obstacles. However, more complex tasks that require hand-eye coordination might need a better reconstruction. Another drawback of using VAEs is the lack of the ability to generalise to unknown objects. VAEs tend not to reconstruct objects that are not observed often, since it would been seen as noise. This might lead to problems when unknown objects occur. For example, avoiding a human arm in our setting would not be possible if it has never seen that arm before, since it would most likely not be reconstructed. This is the nature of VAEs and one should think about the use cases for

the robot before using this method.

In addition to the limitations imposed by the VAE, there are also some limitations regarding the RL part. Our method requires some expert knowledge about the task, since demonstrations are used to speed up the learning process. Without these demonstrations, learning a simple task such as reaching a cup would take a lot more time [Zhang and Ma, 2018]. Furthermore, the current task is a relatively simple task compared to other tasks that robots can perform, and it is already solvable using other techniques such as an object detector in combination with a motion planner. Moreover, no collision avoidance is learned in our experiments, so a motion planner would still be needed when testing the policy on the real robot to prevent it from hitting anything. This is something that could be added within the reward function during training. We planned to do this using MoveIt [Sucan and Chitta, 2013], such that MoveIt could provide us with feedback if a desired joint value would be valid or not, which could be used in the reward function to learn to avoid taking invalid actions. However, we did not get this to work, since MoveIt crashes after a while for no clear reason, so we decided not to learn any collision avoidance. Lastly, training tasks using RL still take a long time and one should currently still wonder if that time is worth saving the manual engineering and fine tuning. However, RL is still a large active research area with a lot of new techniques still under development, including methods to speed up the learning process such as [Jain and Tulabandhula, 2017] and [Stooke and Abbeel, 2018].

8.2 Recommendations

This thesis has shown that using VAEs to obtain a common representation for the simulated and real world is still complicated. The next step to bridge the visual reality gap in order to perform RL with robots would thus be to improve the VAE. One way of improving the VAE is by using a conditional Auto Encoder such as DIVA [Ilse et al., 2019] or Variational Fair Auto Encoders [Louizos et al., 2015] to condition the Auto Encoder on the day of the recorded data set. In this way, the variation caused by the different days can be removed.

Another way to improve the VAE is by continuing to train the simulation VAE during training of the RL part. In this way, more data can be collected while already learning a policy. This could, however, cause a bias towards the states that are visited more often during training, but these are also the states that the robot will probably encounter the most, so good representations of these states are desired. Furthermore, the depth information of the images obtained using a Kinect is not used in this thesis. To improve the information that can be extracted from the Kinect, the depth information can be used to better estimate the position of the objects in the environment.

Besides obtaining more or different data, Data Augmentation [Shorten and Khoshgoftaar, 2019] can be used to generate more training images with different lighting conditions to learn a more robust VAE that performs well under different lighting conditions. Another way to learn a model that is invariant to different lighting conditions is by normalising the colours in order to get colour constancy [Agarwal et al., 2006], which means that the same objects have the same colours under different illumination conditions. For example, the Gray World algorithm [Buchsbaum, 1980] assumes that the colour in each channel averages to grey over the entire image and thus colour constancy can be achieved by dividing each channel by its average value.

Aside from improving the latent space obtained with the VAEs, the RL part can be improved by extending the current tasks to more complex tasks, such as including collision avoindance or grabbing the cup instead of only reaching it. Also, different starting positions can be used to let the robot learn how to reach the cup from different positions.

Finally, one can look into RL methods that try to speed up the learning procedure for example by optimising existing (deep) RL algorithms for a combination of CPUs and GPUs [Stooke and Abbeel, 2018] or by training in parallel [Liu et al., 2018], but this also requires the simulator to be able to run in parallel.

8.3 Conclusion

From the experiments that are performed, we can conclude that a common latent space for the simulated and real world can be learned using VAEs. The generated images from the common latent space look alike and only differ slightly between the simulated and real world. However, the VAE does not generalise well to slightly different data, so the environment should be exactly the same as at the moment that the data set was recorded. This also includes the background and lighting, which is difficult to keep the same since the surrounding of the environment can change. The RL experiments have shown that this latent space can be used to let a robot learn to reach a cup using RL in simulation. Compared to using the position of the cup directly, the performance of using the latent state as input is the same. This shows that all information needed to reach the cup, such as the position of the cup, is captured in the latent state. Since the results of the latent state were not sufficient to reconstruct the scenes with the robot in the real world, the trained policy is only tested in simulation and not in the real world. Thus, it is not possible to determine whether the latent space generated by VAEs can be used to bridge the visual reality gap, which would allow policies trained in simulation to be transferred to the real robot.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1–8. MIT Press, 2007. URL `http://papers.nips.cc/paper/3151-an-application-of-reinforcement-learning-to-aerobatic-helicopter-flight.pdf`.

Vivek Agarwal, Besma R Abidi, Andreas Koschan, and Mongi A Abidi. An overview of color constancy algorithms. *Journal of Pattern Recognition Research*, 1(1):42–54, 2006.

Dana H Ballard. Modular learning in neural networks. In *AAAI*, pages 279–284, 1987.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Gershon Buchsbaum. A spatial processor model for object colour perception. *Journal of the Franklin institute*, 310(1):1–26, 1980.

François Chollet et al. Keras. `https://keras.io`, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

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

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1480–1490. JMLR. org, 2017.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

Maximilian Ilse, Jakub M. Tomczak, Christos Louizos, and Max Welling. DIVA: domain invariant variational autoencoders. *CoRR*, abs/1905.10427, 2019. URL `http://arxiv.org/abs/1905.10427`.

Tadanobu Inoue, Subhajit Choudhury, Giovanni De Magistris, and Sakyasingha Dasgupta. Transfer learning from synthetic to real images using variational autoencoders for precise position detection. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2725–2729. IEEE, 2018.

Vikas Jain and Theja Tulabandhula. Faster reinforcement learning using active simulators. *CoRR*, abs/1703.07853, 2017. URL `http://arxiv.org/abs/1703.07853`.

Stephen James, Andrew J Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *arXiv preprint arXiv:1707.02267*, 2017.

J. L. W. V. Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(1):175–193, Dec 1906. ISSN 1871-2509. doi: 10.1007/BF02418571. URL `https://doi.org/10.1007/BF02418571`.

Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016. URL `http://arxiv.org/abs/1603.02199`.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Qihao Liu, Xiaofeng Liu, and Guoping Cai. Control with distributed deep reinforcement learning: Learn a better policy. *arXiv preprint arXiv:1811.10264*, 2018.

Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. *arXiv preprint arXiv:1511.00830*, 2015.

Richard Meyes, Hasan Tercan, Simon Roggendorf, Thomas Thiele, Christian Büscher, Markus Obdenbusch, Christian Brecher, Sabina Jeschke, and Tobias Meisen. Motion planning for industrial robots using reinforcement learning. *Procedia CIRP*, 63:107–112, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016a.

Andrei A Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016b.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

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

Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

Ioan A Sucan and Sachin Chitta. Moveit! *Online at http://moveit. ros. org*, 2013.

Lei Tai and Ming Liu. Deep-learning in mobile robotics - from perception to control systems: A survey on why and why not. *CoRR*, abs/1612.07139, 2016. URL `http://arxiv.org/abs/1612.07139`.

Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

Thiemo Wiedemeyer. IAI Kinect2. `https://github.com/code-iai/iai_kinect2`, 2014 – 2015. Accessed June 12, 2015.

Fangyi Zhang, Jürgen Leitner, Michael Milford, and Peter Corke. Sim-to-real transfer of visuomotor policies for reaching in clutter: Domain randomization and adaptation with modular networks. *world*, 7(8), 2017.

Xiaoqin Zhang and Huimin Ma. Pretraining deep actor-critic reinforcement learning algorithms with expert demonstrations. *arXiv preprint arXiv:1801.10459*, 2018.

Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018. URL `http://arxiv.org/abs/1807.05511`.