Learning to Generalise through Features

by

Dmitry Grebenyuk

ORCID: 0000-0003-3799-104X

A thesis submitted in total fulfillment for the
degree of Master of Philosophy

in the
Melbourne School of Computing and Information Systems
Melbourne School of Engineering
THE UNIVERSITY OF MELBOURNE

October 2020

THE UNIVERSITY OF MELBOURNE

Abstract

Melbourne School of Computing and Information Systems
Melbourne School of Engineering

Master of Philosophy

by Dmitry Grebenyuk
ORCID: 0000-0003-3799-104X

A Markov decision process (MDP) cannot be used for learning end-to-end control policies in Reinforcement Learning when the dimension of the feature vectors changes from one trial to the next. For example, this difference is present in an environment where the number of blocks to manipulate can vary. Because we cannot learn a different policy for each number of blocks, we suggest framing the problem as a POMDP instead of the MDP. It allows us to construct a constant observation space for a dynamic state space. There are two ways we can achieve such construction.

First, we can design a hand-crafted set of observations for a particular problem. However, that set cannot be readily transferred to another problem, and it often requires domain-dependent knowledge.

On the other hand, a set of observations can be deduced from visual observations. This approach is universal, and it allows us to easily incorporate the geometry of the problem into the observations, which can be challenging to hard-code in the former method.

In this Thesis, we examine both of these methods. Our goal is to learn policies that can be generalised to new tasks. First, we show that a more general observation space can improve the performance of policies tested in untrained tasks. Second, we show that meaningful feature vectors can be obtained from visual observations. If properly regularised, these vectors can reflect the spacial structure of the state space and used for planning. Using these vectors, we construct an auto-generated reward function, able to learn working policies.

Declaration of Authorship

I, Dmitry Grebenyuk, declare that this thesis titled, 'Learning to Generalise through Features' and the work presented in it are my own. I confirm that:

- The thesis comprises only my original work towards the Master of Philosophy except where indicated in the preface;
- due acknowledgement has been made in the text to all other material used; and
- the thesis is fewer than the maximum word limit in length, exclusive of tables, maps, bibliographies and appendices as approved by the Research Higher Degrees Committee.

Signed: _____

Date: _____

Preface

All the chapters of this thesis contain unpublished material not submitted for publication.

The research was funded by the Australian Government Research Training Program (RTP) Scholarship.

Acknowledgements

I want to thank my primary supervisors, Nir Lipovetzky and Miquel Ramirez Javega, for their deep patience and infinite calm. Without their invaluable advice, this research would be impossible.

I also want to thank Kris Ehinger, who helped me greatly with writing this thesis.

Finally, I want to thank the Australian government, which gave me the Research Training Program (RTP) Scholarship. Without this money, my journey to Australia would be impossible.

Contents

List of Figures

List of Tables

Abbreviations

AE	**A**uto**E**ncoder
CDPAE	**C**omprehensive **D**istance-**P**reserving **A**uto**E**ncoder
CNN	**C**onvolutional **N**eural **N**etwork
DDPG	**D**eep **D**eterministic **P**olicy **G**radients
DQN	**D**eep **Q**-**N**etwork
LSTM	**L**ong **S**hort-**T**erm **M**emory
MDP	**M**arkov **D**ecision **P**rocess
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
PPO	**P**roximal **P**olicy **O**ptimisation
RL	**R**einforcement **L**earning
TRPO	**T**rust **R**egion **P**olicy **O**ptimisation
UPN	**U**niversal **P**lanning **N**etworks
VAE	**V**ariational **A**uto**E**ncoder

Constants

Symbols

a	action
A	advantage function
$\mathcal{A}$	action space
$\mathcal{D}$	experience replay buffer
D_{KL}	Kullback–Leibler divergence
H	horizon
J	expected discounted reward
h	latent action
$\mathcal{H}$	latent action space
O	observation function
Q	action-value function
r	reward
R	reward function
s	state
$\tilde{S}_{a,o}$	sensing sphere
$\mathcal{S}$	state space
t	time step
T	state-transition function
V	value function
X_{goal}	goal coordinates
z	latent state
$\mathcal{Z}$	latent space
γ	discount factor
θ, ϕ	neural network weights

Ω	observation space
π	policy
χ	configuration

Chapter 1

Introduction

1.1 Motivation

Although it is not a difficult task for humans to pick or place arbitrary objects in open, unknown environments or under uncontrolled conditions, such manipulation tasks are still an open challenge in robotics. Traditionally, in the planning literature, such movements were precomputed or solved by probabilistic roadmaps [1], [2] or rapidly-exploring random trees [3], [4], which take random samples from the configuration space of the robot.

However, these algorithms treat all the surrounding objects as obstacles. Thus, they re-sample the problem whenever any object has moved, which can be a considerable computational overhead in high-dimensional configuration spaces. Additionally, these algorithms cannot be applied if the configuration space is unknown, which is usually the case in complicated, real-world tasks.

To overcome this problem, Reinforcement Learning (RL) can be used. There is significant progress in learning end-to-end policies (behaviours) in continuous control tasks such as robot locomotion [5] or dexterous in-hand manipulations [6] and recent attempts in multi-tasking [7].

Although many of these learned policies have high complexity, the authors usually focus on highly constrained, isolated tasks in a controlled environment. Thus, their policies

cannot be readily transferred to another, although similar, problem or to another environment. However, it is also important to focus on the robot's ability to handle a variety of tasks or operate in a range of situations, i.e. to focus on generalisation.

However, the current RL algorithms can be one of the problems affecting generalisation. These algorithms can only be employed if their input observations, called feature vectors, are of fixed size. At the same time, the feature vectors, being the sole source of information about the current state, are traditionally defined as points in a robot's configuration space. Such learning setup cannot be used in unbounded problems because the number of elements can vary in the problem's configuration space. Moreover, the configuration space can be unknown or too complex to define mathematically.

Therefore, there is a recent trend to re-define RL learning problems as a Partially Observable Markov Decision Process (POMDP) because, in this framework, the configuration space, called state space, is unknown and hidden. In this case, the RL agent operates in the observation space that partially observes the state space. Therefore, because the observation space can reflect only the essential aspects of the state space, feature vectors drawn from it can have fixed size even in unbounded problems.

For this purpose, feature vectors can be hand-crafted or deduced from visual observations. However, in the former approach, it can be a daunting task to design a proper observation space that both incorporates the geometry of the problem and can be readily transferred to another task. On the other hand, in the latter approach, feature vectors can be simply defined as the intensities of raw image pixels. However, although it can be expedient, such observation space does not reflect the structure of the hidden state space, which can impede planning.

On the other hand, in recent years, it has been shown that the raw images could be compressed to their latent representations using Variational Autoencoders (VAEs). These representations may preserve some properties and approximate metric structure when mapped into latent space, which can reflect the fundamental aspects of the environment. However, the right space organisation is still an open question, which we examine as one of our research questions.

The second problem affecting the transferability of modern RL algorithms can be attributed to their use of hand-crafted reward functions. These functions are designed to

guide the agent to obtain desirable policies. However, because it is hard to create an efficient function that can be applied to unbounded problems, these functions usually work in isolated tasks or controlled environments. Thus, an auto-generated goal-conditioned reward function can accelerate the adoption of RL algorithms and their transferability. Therefore, our second research question is to generate an observation space from visual observations, reflecting the spatial structure of the hidden state space.

1.2 Literature Review

End-to-end Reinforcement Learning (RL) is a dynamic field of research, which produced a variety of benchmarks such as Atari games [8] or OpenAI Gym [9], where all the environments are provided with hand-crafted reward functions. In OpenAI Gym, an agent directly observes the state space, which is defined as a set of hand-crafted features. On the other hand, in Atari games, the state space is given as a set of screen frames. Both of these benchmarks offer highly constrained, isolated tasks in controlled environments. Therefore, they cannot be used in our experiments to examine policy generality.

Control problems can be categorised in many ways. We are primarily interested in:

- learning from hand-crafted features
- learning from visual observations
- learning from hand-crafted reward functions
- learning from auto-generated reward functions, i.e. self-supervised learning or imitation learning.

Learning from Hand-Crafted Features

Hand-crafted features are usually defined as direct signals from a robot's sensors. For example, Haarnoja et al. [5] learned policies for continuous locomotion tasks in OpenAI Gym using an extension to the soft actor-critic algorithm with better sample efficiency.

Andrychowicz et al. [6] used the PPO algorithm to learn dexterous in-hand manipulation policies that could perform vision-based object reorientation. The policy was learned

for the observation space given as the coordinates of the finger's markers using a hand-crafted reward. Although a convolutional neural network (CNN) was trained to map the images into the fingers' coordinates, the authors used very constrained environments and operated in the state space.

Additionally, Neither Haarnoja et al. [5] nor Andrychowicz et al. [6] attempted to examine the generality of their policies.

Learning from Visual Observations

On the other hand, there is significant progress in learning policies from visual observations in recent years.

By using a Deep Q-Network (DQN), Mnih et al. [8], [10] demonstrated that it is possible to play Atari games on the human level using only screen pixels. However, the authors used a series of frames as observations to keep the information about previous states, which effectively reduced the problem to an MDP.

Additionally, Hausknecht et al. [11] investigated the effects of adding recurrency to the DQN by replacing the first post-convolutional fully-connected layer with a recurrent Long Short-Term Memory (LSTM). They showed that their algorithm had better performance in Atari games when the environment is defined as a POMDP instead of an MDP, i.e. when the observations were comprised of one frame at each time step. However, in both of the articles, the rewards were predefined, and the policies were learned from raw pixels.

Nasiriany et al. [12] suggested planning over model-free goal-conditioned polices representing sub-goals. The authors observed that the learned policies could only be used for short-term planning, because the leaning produced overly greedy policies.

Another way to obtain a general policy was presented in Action Schema Networks (ASNet) [13], in which the authors used supervised learning to learn an action-proposition representation of the problem. This presentation could automatically generalise to any problem from a given planning domain and could plan in this domain using a heuristic.

Imitation Learning

On the other hand, instead of crafting a reward function, it can be expedient to learn correct behaviour from human-made demonstrations.

Lynch et al. [7] proposed to learn from teleoperated play data. The authors encoded a large collection of unscripted robot play data into latent space, where each point in the space represented a single trajectory. For this purpose, a vanilla Variational Autoencoder (VAE) was used. The decoder was goal conditioned. That is, it decoded a trajectory that had the closest final state to the goal state, which was used as a policy.

Zhang et al. [14] used a virtual reality device to teleoperate a robot to collect demonstrations on ten manipulation tasks. The authors used raw image pixels that were processed by a Convolutional Neural Network (CNN) into the state space. Then, the policies were learned using a standard imitation learning loss function.

Losey et al. [15] encoded high-DoF robot actions into intuitive, human-controllable, and low-DoF latent actions, which can be directly coordinated with a handheld joystick. The authors used a vanilla autoencoder, trained from teleoperated task demonstrations.

To encode motions into the latent space, Dwibedi et al. [16] encoded multiple video frames simultaneously, which encouraged the network to learn similarities between the videos. First, each frame's raw pixels were encoded by a CNN. After that, multiple frames were connected by a temporal convolution [17].

In Universal Planning Networks (UPN) proposed by Srinivas et al. [18], the authors used a vanilla autoencoder to map video frames into the latent space. The authors suggested planning in this space by imitation human-made demonstrations.

However, in the works discussed above, the spatial structure of the spaces did not affect resulting policies.

Zhan et al. [19] proposed Comprehensive Distance-Preserving Autoencoders (CDPAE) to address the problem of unsupervised cross-modal retrieval. Besides pairwise distances, the CDPAE also considered heterogeneous distances of representations extracted from cross-media spaces (e. g. photos, and videos). That is, putting aside the authors' terminology, the latent space was organised in a way to place similar images close to each other, e.g. dogs to dogs and cats to cats. The CDPAE used the cosine similarity distance to measure the similarity of features in the same media spaces.

On the other hand, Ghosh et al. [20] aimed to learn actionable representations with goal-conditioned policies. That is, the authors aimed to capture the factors that are important for decision making. Thus, the latent space was organised in a way that

similar goal-conditioned policies were placed close to each other. In contrast to our approach, the policies were decoded into the latent space, where the distance between policy distributions was measured by KL-divergence.

Self-Supervised Learning

In contrast to the standard RL technique, where each new skill requires a manually-designed reward function, Pong et al. [21] suggested setting self-supervised goals, enabling an agent to propose its own goals and acquire skills to achieve these goals. The authors proposed a new objective function for learning to model the uniform distribution over states when only data collected by an autonomous goal-conditioned policy was given. The experiments were made from visual observations using a vanilla VAE.

Nair et al. [22] presented another self-supervised learning approach. The authors trained a vanilla VAE using data generated by an exploration policy. After that, new goals were sampled from the VAE's latent space, which was used to train a goal-conditioned policy with reward measuring Mahalanobis distance in the latent space.

Hafner et al. [23] proposed a model-based agent that learns the environment's dynamics from pixels and chooses actions through online planning in a compact latent space. Policy planning was done by model-predictive control [24]. That is, the plan was re-generated each time step. Notably, the latent space, representing the environment's dynamics, was comprised of both stochastic and deterministic parts, learned by a VAE.

However, in all the discussed work, the authors did not try to construct a latent space reflecting the spatial structure of the state space or examine the generality of their policies.

1.3 Aim, Scope, and Contributions

In this thesis, we focus on two research questions. In the literature, the authors usually focus on highly constrained, isolated tasks in controlled environments. Thus, their policies cannot be readily transferred to another, although similar, problem. Therefore, our first question is to explicitly construct an observation space that can be used for learning policies when the number of elements in the state space can vary. Particularly, our goal is to learn end-to-end policies for picking and placing objects when the number

of objects can vary. We are interested in policies that can work with the number of objects that are not used in training. More broadly, we are interested in examining the connection between the policy's generalisation ability and its observation space.

To solve this problem, we propose a new learning environment, in which the number of elements in the state space can vary. We frame this environment on a POMDP, for which we construct three observation spaces of fixed size. We show that naively constructed observation space can impede the policy's generalisation ability.

Our second research question is influenced by the recent advancement in self-supervised learning. Our goal is to generate the latent space from visual observations using a VAE. However, in contrast to the state of the art, we are interested in the latent space that reflects the spatial structure of the hidden state space, from which visual observations are taken. In contrast to current practice, we use visual observations taken from two cameras instead of one. The cameras are placed perpendicular to each other.

Additionally, we use the latent space to construct a goal-conditioned reward function. However, in contrast to the state of the art, we are interested in a reward function that can be used with RL algorithms that are not goal-conditioned. Particularly, we focus on learning polices using visual observations and the suggested goal-conditioned reward. More broadly, our goal is to show that the same problem setup is general enough to be applied to a variety of problems.

As our contribution, we suggest a new auto-generated goal-conditioned reward function measuring the distance to the goal in the latent space. We show that VAEs with the vanilla regularisation cannot be used in such rewards because vanilla regularisation cannot guarantee that the latent space reflects the structure of the hidden state space. Thus, we propose and examine new regularisation.

1.4 Thesis Structure

The structure of the thesis is organised as follows. In Chapter 2, we discuss the required background knowledge. More particularly:

- In Section 2.1, we introduce MDPs and POMDPs, as well as, we discuss their major differences.

- Section 2.2 is dedicated to Reinforcement Learning. We commence discussing Q-Learning in Section 2.2.1 and its machine learning extension, Deep Q-Networks, in Section 2.2.2. After that, Policy Gradient is introduced in Section 2.2.3, which is used in the DDPG algorithm in Section 2.2.4. Then, we conclude with the notion of trust regions in Section 2.2.5 and the PPO algorithm in Section 2.2.6.
- Finally, in Section 2.3, we discuss autoencoders in Section 2.3.1 and VAEs in Section 2.3.2.

In Chapter 3, we concentrate on constructing observation spaces when the number of elements in the state space can vary. After a brief introduction to the chapter in Section 3.1, we define a new Kuka environment in Section 3.2. Further, we propose four observation spaces in Section 3.3 and evaluation tasks for picking and placing objects in Section 3.4, which are used in our experiments in Section 3.5.

Chapter 4 is dedicated to generating observation spaces from visual observations and learning policies using goal-conditioned reward functions. Firstly, we briefly introduce the chapter in Section 4.1. Section 4.2 focuses on the state-similarity reward function, which requires a proper regularisation of the latent space. We discuss this regularisation in Section 4.3. After that, we extend the Kuka environment and introduce a new Reacher environment in Section 4.4. Finally, we use these environments in our experiments in Section 4.5.

In Chapter 5, we summarise the contributions of the thesis and outline possible future directions.

Chapter 2

Background

We begin the chapter with a review of a Partially Observable Markov Decision Process (POMDP). In contrast to a Markov Decision Process (MDP), a POMDP separates the state space, which is hidden, from the observation space. This mathematical framework is useful in situations when the structure of the state space (i.e. the environment) is not observable by the agent, which in turn means that the Markov property no longer holds. In such situations, it is desirable to use visual observations as the observation space.

Because we concentrate on learning general policies for manipulation problems using Reinforcement Learning (RL), we frame our environments on a POMDP. Thus, later, we review relevant RL algorithms. We show that three of those algorithms can work with continuous control signals and continuous state spaces, which are desirable in manipulation problems.

However, if raw pixels are used as feature vectors in Reinforcement Learning, it is hard to deduce any similarity between two images and their corresponding hidden state, which can affect planning. Therefore, in recent years, there is a growing interest in encoding images into their compact representation (i.e. the members of a latent space) employing autoencoders. Because we extensively use this technique, we review autoencoders and the structure of their latent space at the end of the chapter.

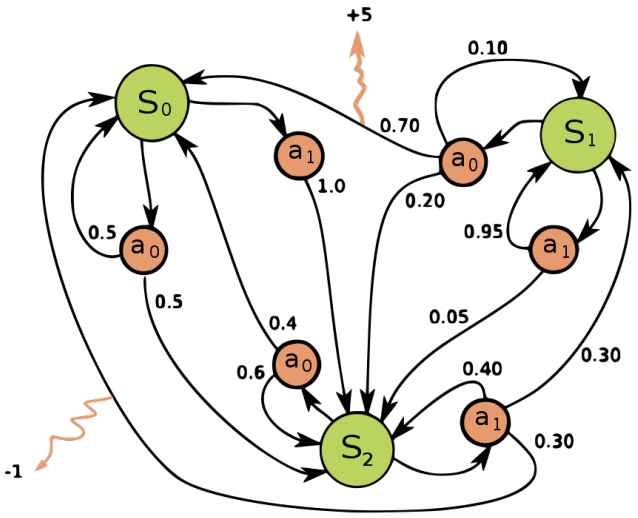

FIGURE 2.1: An example of an MDP with three states (green circles), two actions (orange circles), and two rewards (orange arrows) by Waldoalvarez, licensed under CC BY-SA 4.0.

2.1 POMDP

In general, most AI problems should be framed on a mathematical framework before they can be solved. In Reinforcement Learning (RL), such a framework is a Markov Decision Process (MDP), formally introduced to the field by Kaelbling et al. [25].

The MDP is convenient to model complex problems, in which an *agent* (e.g. a learning algorithm) observes an *environment*, receiving the current state, and takes *actions*. The actions are influenced by rewards. Is some cases, those rewards can be infrequent or delayed, which makes it hard to identify the actions positively contributing to the reward. For example, a chess game can be won or lost many moves before its end. An example of the MDP can be found in Figure 2.1.

Mathematically, an MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R)$, where:

- $\mathcal{S}$ is a set of states. For robotic control, it can be represented by raw images or joint angles and velocities.
- $\mathcal{A}$ is a set of actions. It represents all the actions that the agent can take.
- $T : S \times A \to \Pi(\mathcal{S})$ is the *state-transition function* that estimates the transition probabilities between two states $s, s' \in \mathcal{S}$ when an action $a \in \mathcal{A}$ is taken.

- $R : S \times A \to \mathbb{R}$ is the *reward function* providing the expected immediate reward gained by the agent for taking an action $a \in \mathcal{A}$ in a state $s \in \mathcal{S}$.

To be framed on the MDP, the problem has to hold the Markov property. We say that a stochastic process has the Markov property if the conditional probability distribution of future states of the process depends only on the present state, not on the sequence of events that preceded it:

$$P(X_n = s_n \mid X_{n-1} = s_{n-1}, \ldots, X_0 = s_0) = P(X_n = s_n \mid X_{n-1} = s_{n-1}) \,.$$

The agent's objective is to maximise expected discounted future reward

$$\mathbb{E}_\pi \left[\sum_{t=0}^{H} \gamma^t r_t \right],$$

where $\gamma \in [0, 1]$ is a discount factor, r is a reward, $H \in \mathbb{N}$ is planning horizon, and the actions are drawn according to a *policy* $\pi : \mathcal{S} \to \mathcal{A}$. A policy maximising the future reward is called *optimal*.

When we frame a learning problem on an MDP, we assume that the agent can perfectly observe the state space $\mathcal{S}$, which is usually not the case in real manipulation problems. Thus, the Markov property does not hold. In such cases, the common practice is to approximate the problem as an MDP.

To improve the prediction about the future, a Partially Observable Markov Decision Process (POMDP) can be used [11], [25]. The POMDP separates the state space $\mathcal{S}$, which is hidden, from the observation space Ω that has partial information about the state space. Thus, in contrast to MDPs, the agent is unable to observe the current state. Instead, the state is deduced from observations taken after the current action has been taken.

Mathematically, the POMDP extends the MDP and defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O)$, where:

- $\mathcal{S}, \mathcal{A}, T, R$ are the same as in the MDP.
- Ω is a set of observations.

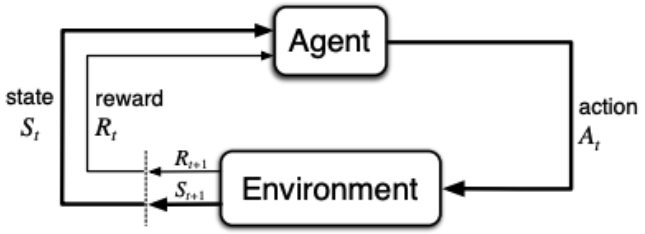

FIGURE 2.2: Agent–environment interactions in reinforcement learning from Sutton and Barto [26].

- $O : S \times A \rightarrow \Pi(\Omega)$ is the *observation function* that defined the probability of taking observation $o \in \Omega$ when the agent took an action $a \in \mathcal{A}$ and landed in a state $s \in \mathcal{S}$.

Usually, in manipulation problems, the POMDP has unknown state-transition and observation functions as well as large continuous action and state spaces. In such cases, it can be infeasible to find the optimal policy analytically. Thus, an approximate policy should be learned by RL. Now we proceed to describe the existing learning algorithms briefly.

2.2 Reinforcement Learning

In this section, we review existing reinforcement learning algorithms beginning with a historical review. First, we introduce the model of *agent-environment interactions*.

Suppose, there is a robot (i.e. the *agent*) with a task to cross a room full of mines, and the robot can only move one tile at a time. Those movements represent a set of all possible action $\mathcal{A}$ in our *environment* (Figure 2.2). After each action, the agent receives the robot's new location in the room that represents a state $s \in \mathcal{S}$ in our environment, often called a *feature vector*. Additionally, the agent receives a reward R assessing the taken actions. Based on that data, the robot takes a new action. The process is repeated until the agent reached the goal state or ran out of time, concluding a *trial*. The objective of RL is to learn a policy π that assigns the best action $a \in \mathcal{A}$ to each state $s \in \mathcal{S}$; that is, $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

2.2.1 Q-Learning

Q-Learning [27] is a seminal, model-free Reinforcement Learning algorithm, based on the agent-environment interactions, where the agent estimates an *action-value function* $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The function $Q[s, a]$ is defined as a lookup table in which the maximum expected future rewards are calculated for each action a at each state s. The policy is defined as

$$\pi(s) = \arg\max_a Q[s, a] \,.$$

Initially, values in the action-value table are filled randomly, but they are updated after each trial, gradually converging to the optimal policy. The learning rule is

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right), \tag{2.1}$$

where r_t is the reward, $\alpha \in [0, 1]$ is the learning rate, and $\gamma \in [0, 1]$ is a discount factor. Actions a on each trial can be drawn from the current policy or randomly, which is influenced by exploration vs exploitation trade-off.

However, due to the use of a lookup table, the algorithm is limited to small discreet action or state spaces. Later, we discuss several extension that can work in continuous spaces.

2.2.2 Deep Q-Network

Deep Q-Network (DQN) [8] is a machine learning adaptation of the Q-Learning algorithm. DQN uses a neural network with weights θ to approximate an action-value function $Q(s, a; \theta)$, called a Q-Network. Thus, such a network can be used in continuous state spaces $\mathcal{S}$. However, actions are still drawn from Equation 2.2.1, which limits the use of DQN to small discreet action spaces $\mathcal{A}$.

Framed on the MDP, the agent's learning objective is to maximise the discounted future reward. One of DQN's novel contribution is in stabilising the training process. Instead of updating the policy after each trial, it is updated on a batch of trials drawn from an *experience replay buffer* $\mathcal{D}$. A learning update at each iteration k is made by minimising

a temporal difference (TD) error $\mathcal{L}(\theta_k)$ such that

$$\mathcal{L}(\theta_k) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_k^-) - Q(s, a; \theta_k)\right)^2\right], \tag{2.2}$$

where θ_k, θ_k^- are the current and previous weights of a Q-Network. The minimisation is made by gradient descent with a learning rate α such that

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta_k)\,. \tag{2.3}$$

Although DQN was initially designed for playing Atari games from raw pixels, the learning problem was defined as an MDP, where each state is combined of four consecutive frames. It was shown [11] that DQN performance could be diminished if one image is used instead of four.

Additionally, there are two notable extensions: Double DQN [28], [29] and Dueling DQN [30] that we do not review in this thesis. Both of them, as well as vanilla DQN, are limited to small discrete action spaces; therefore, they cannot be used in manipulation problems having continuous spaces. In following Section 2.2.3, we introduce Policy Gradient method that does not have such a limitation.

2.2.3 Policy Gradient

Policy Gradient (PG) was initially suggested in REINFORCE paper [31] and further developed by Sutton et al. [32]. It adopts a different approach to the learning problem. Instead of optimising an action-value function Q, Policy Gradient directly optimises a policy distribution $\pi_\theta(a|s)$ over multiple trajectories τ to maximise expected discounted rewards $J(\pi_\theta)$; that is,

$$\underset{\theta}{\text{maximise}}\; J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{H} \gamma^t r_t\right] = \mathbb{E}_{\tau\sim\pi_\theta}\Big[R(\tau)\Big], \tag{2.4}$$

where H is the learning horizon. Thus, Policy Gradient can be expressed as follows

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot A^{\pi_\theta}(s_t, a_t)\right], \tag{2.5}$$

where $A^{\pi_\theta}(s_t, a_t)$ is an *advantage function*. In the case of REINFORCE, it is presented as $R(\tau)$, in a more general case, it is

$$A^{\pi_\theta}(s_t, a_t) = \sum_{t'=t}^{H} R(s_{t'}, a_{t'}, s_{t+1}) - b(s_t)\,,$$

where $b(s_t)$ is a baseline. Thus, the advantage function can be further rewritten as

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\,,$$

where $V^{\pi_\theta}(s) = \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \cdot Q^{\pi_\theta}(s, a)$ is a *value function*. The policy updates are made by gradient ascent

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)\,. \tag{2.6}$$

In that case, the policy is stochastic. That is, it returns a probability distribution, which can be suitable for continuous action spaces.

However, the convergence of the Policy Gradient method can be unstable due to the high variance of expected rewards $R(\tau)$. This problem is inherited into expected rewards because one unlucky action can affect the whole trajectory and the length these trajectories may vary. Moreover, this problem is still an active field of research yielding a variety of Policy Gradient algorithms such as A3C [10], A2C [10], DPG [33], D4PG [34], ACER [35], ACTKR [36], SAC [37], and TD3 [38]. However, we solely concentrate on DDPG (2.2.4), TRPO (2.2.5), and PPO (2.2.6), which are capable of operating in continuous action and state spaces.

2.2.4 Deep Deterministic Policy Gradient

The training of Policy Gradient models can be unstable because of the high variance inherent in expected rewards $R(\tau)$. This problem is still an active field of research yielded a variety of algorithms. In this section, we concentrate on Deep Deterministic Policy Gradient (DDPG) [39] designed to address that problem. The additional advantage of DDPG that it can operate in continuous action and state spaces, which is required in our research. DDPG improves training stability by combining the value-based (2.2.2) and policy-based algorithms (2.2.3) into an Actor-Critic method.

Policy Gradient Equation 2.5 can be rewritten as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \nabla_\pi Q(s_t, \pi_\theta(s))\right]. \tag{2.7}$$

Both parts can be estimated separately, which leads us to the Actor-Critic method, where:

- First, the critic network estimates the action-value function $Q(s_t, \pi_\theta(s))$ using the deep Q-Learning.
- After that, the actor network updated the policy distribution $\pi_\theta(a|s)$ using the Policy Gradient.

However, DDPG has another source of instability. Because the policy distribution can be flat in one region but steep in another, gradient updates can collapse into a local minimum. This problem is addressed in following Section 2.2.5.

2.2.5 Trust Region Policy Optimisation

Finding a suitable learning rate for gradient updates can be a daunting task. The policy can be flat in one region but steep in another. Thus, the same change in parameter space θ can lead to different policy updates in different regions because of the policy's curvature. Thus, gradient updates can collapse into a local minimum, while climbing to the global maximum.

To constrain policy changes to a safe region, Trust Region Policy Optimisation (TRPO) [40] was suggested. It uses natural policy gradients. TRPO uses natural policy gradients to maximise a surrogate loss that estimates the lower bound of the expected discounted reward $J(\pi)$:

$$J(\pi) \geq \mathcal{L}_\pi(\pi_{\text{old}}) - C\mathbb{E}_{s\sim d^\pi}\left[D_{\text{KL}}(\pi||\pi_{\text{old}})[s]\right], \tag{2.8}$$

where C is a normalisation constant, D_{KL} is a Kullback–Leibler divergence such that

$$D_{\text{KL}}(P\,||\,Q) = \mathbb{E}_x \log \frac{P(x)}{Q(x)}, \tag{2.9}$$

d^π is a discounted future state distribution such that

$$d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi)\,, \tag{2.10}$$

and $\mathcal{L}_\pi(\pi)$ is an importance sampling loss such as

$$\mathcal{L}_\pi(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)}A^\pi(s_t,a_t)\right], \tag{2.11}$$

where A^π is an advantage function as in Section 2.2.3.

Although TRPO shows improved learning stability, the calculation of KL-divergences requires more computational resources due to the use of second derivatives, which limits the scale of possible learning problems. In following Section 2.2.6, we discuss PPO that does not use KL-divergences.

2.2.6 Proximal Policy Optimisation

In practice, natural policy gradient involves the calculation of second-order derivatives, which limits the scale of possible learning problems. Proximal Policy Optimisation (PPO) [41] addresses this problem. Instead of imposing a trust region, PPO clips the loss function such that

$$\mathcal{L}^{\text{CLIP}}_{\theta_k}(\theta) = \mathbb{E}_{\tau\sim\pi}\left[\min\left(r_t(\theta)\hat{A}^{\pi_k}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_k}_t\right)\right], \tag{2.12}$$

where $r_t(\theta)$ is a ratio between new and old policie distributions

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}\,.$$

Therefore, the loss function $\mathcal{L}^{\text{CLIP}}_{\theta_k}(\theta)$ can be solved by first-order optimisers.

In our research, we use PPO as our primary learning algorithm. However, we deviate from common practice and frame our learning problems as POMDPs instead of MDPs. That is, instead of learning the policy distribution $\pi_\theta(a|s)$, we learn the distribution $\pi_\theta(a|o)$, where $o \in \Omega$.

Therefore, to frame the problem as the POMDP, we need to define the observation space Ω, which we constructed from visual observation employing autoencoders. Thus, in following Section 2.3, we give their formal introduction.

2.3 Autoencoders

Autoencoders (AEs) have been part of the state of the art for decades [42], and they are closely related to compression algorithms such as Principal Component Analysis (PCA) [43]. Initially, AEs were proposed for dimensionality reduction and feature learning. However, unlike PCA, they can learn non-linear representations due to the use of a non-linear activation function.

Recently, Generative Adversarial Networks (GANs) [44], based on Variational AEs, have sparked a new line of research in Reinforcement Learning (RL). It is desirable in manipulation problems to use visual observations because they contain geometrical information about the environment, which could be problematic to hard-code if we used hand-made features. Although RL algorithms can successfully learn policies from raw pixels [8], [11], the states constructed from them have no notion of similarity or closeness, which can negatively affect planning. Thus, it was proposed [18] to use AEs for deducing useful features from raw images, because they are designed to work with unstructured data. In Chapter 4, we used Variational AEs to construct observation spaces in POMDP frameworks from raw images. Now we proceed to introduce vanilla and Variational AEs.

2.3.1 Vanilla Autoencoder

Autoencoders (AEs) [45] aim to copy their inputs to their outputs. It is done by compressing the input into a latent representation, from which the output is reconstructed. That is, AEs encode input values $x \in \mathcal{X}$ into a latent space $\mathcal{Z}$, i.e. $f : \mathcal{X} \to \mathcal{Z}$. Then, the latent representation $z \in \mathcal{Z}$ can be decoded back to its input value x such that $g : \mathcal{Z} \to \mathcal{X}$. Thus, AEs represent a form of compression, where the latent space serves as a compact representation of the input space.

For any input space $\mathcal{X}$, the functions f and g can be approximated by a neural network with weights θ. Learning updates are made by minimising a mean squared error (MSE)

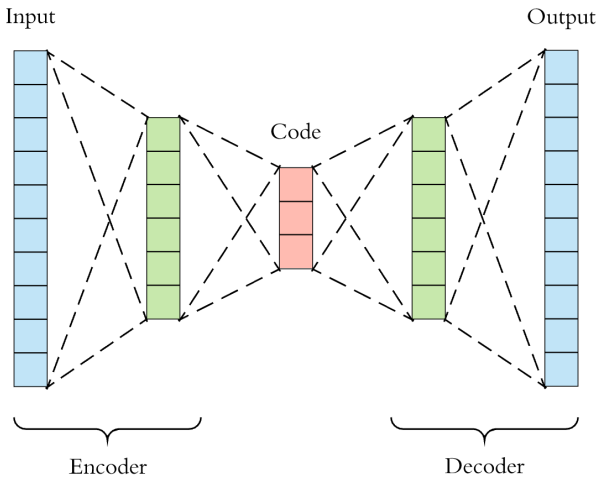

FIGURE 2.3: The structure of an autoencoder.

loss function

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=0}^{N}\left|\left|x_i - g_\theta(f_\theta(x_i))\right|\right|^2. \tag{2.13}$$

The learning objective is to reconstruct the input as accurate as possible.

However, the loss $\mathcal{L}(\theta)$ makes no restrictions on the structure of latent space. We cannot guarantee that two states close in $\mathcal{X}$ will be close in $\mathcal{Z}$ as well. Thus, it can be challenging to separate the unique features of the input space.

On the other hand, unique features can be overly separated. That is, the space $\mathcal{Z}$ can form clusters with meaningless states between them. Thus, the learned latent space $\mathcal{Z}$ might not be convex or continuous when it is desired. Therefore, such spaces cannot be used as observation spaces in POMDPs or for planning.

Although there are many kinds of AEs such as Sparse Autoencoders (SAEs) [46], Denoising Autoencoders (DAEs) [47], or Contractive Autoencoders (CAEs) [48], only for Variational Autoencoders (VAEs) [49], it can be guaranteed that the latent space is convex and continuous. Thus, we will use VAEs in Chapter 4, and we focus on VAEs in the following section.

2.3.2 Variational Autoencoder

Although Variational autoencoders (VAEs) [49] have the architecture similar to vanilla AEs, they can regularise the latent space $\mathcal{Z}$. Instead of learning a deterministic function $f : \mathcal{X} \to Z$, a VAE learns a latent distribution $p(z|x)$, from which latent states $z \in \mathcal{Z}$

can be sampled. Then, these states can be decoded back by a deterministic function $g : \mathcal{Z} \to \mathcal{X}$.

With Bayes' theorem, we can define a variational inference problem:

$$p(z|x) = \frac{p(x|z)\, p(z)}{p(x)}\,, \tag{2.14}$$

where $p(x|z)$ is assumed to be unknown normal distribution and $p(z)$ is $\mathcal{N}(0, I)$.

The distribution $p(z|x)$ can be approximated by a Gaussian distribution $q_x(z)$ such that

$$q_x(z) = \mathcal{N}(\mu(x), \sigma(x))\,.$$

The functions $\mu(x)$ and $\sigma(x)$ are unknown deterministic functions, which can be learned by a neural network with weights θ. Thus, by using the so-called reparametrisation trick, we define a distribution $q_x^\theta(z)$ as

$$q_x^\theta(z) = \mu_\theta(x) + \sigma_\theta(x) \cdot \mathcal{N}(0, I)\,. \tag{2.15}$$

Additionally, the unknown distribution $p(x|z)$ can be approximated by the deterministic function $g_\theta(z)$.

Finally, we can construct a loss function $\mathcal{L}(\theta)$ by minimising the Kullback-Leibler divergence such as

$$\begin{aligned}
\mathcal{L}(\theta) &= \arg\min_\theta D_{\mathrm{KL}}\Big(q_x^\theta(z)\,||\,p(z|x)\Big) \\
&= \arg\max_\theta \left(\mathbb{E}_{z\sim q_x(z)}\Big[\log p(x|z)\Big] - D_{\mathrm{KL}}\Big(q_x^\theta(z)\,||\,p(z)\Big)\right) \\
&= \arg\max_\theta \left(\mathbb{E}_{z\sim q_x(z)}\Big[-\Big|\Big|x - g_\theta(z)\Big|\Big|^2\Big] - D_{\mathrm{KL}}\Big(q_x^\theta(z)\,||\,p(z)\Big)\right).
\end{aligned} \tag{2.16}$$

The equation represents a trade-off between minimising a reconstruction error and staying close to the prior distribution, i.e. a normal distribution. By requiring to stay close to the normal distribution, which is convex and continuous, we obtain a guarantee that our distribution $q_x^\theta(z)$ is convex and continuous as well.

However, in any latent space regularised with Equation 2.16, two states close in $\mathcal{X}$ can be apart in $\mathcal{Z}$. More generally, we cannot guarantee that a triangle inequality holds. This problem is one of our research questions, which we address in Chapter 4.

2.4 Summary

Traditionally, a learning problem is framed as an MDP; however, there is a recent trend in using POMDPs instead. This mathematical framework is useful in situations when the structure of the state space is complicated, and the use of visual observations is desired.

Thus, if a manipulation problem is framed on a POMDP, the problem usually has continuous action and state spaces. In such cases, a policy can be learned by PPO. Although it is possible to learn the policy from feature vectors as raw pixels, it can be challenging to deduce a meaningful structure of the hidden state space, which can affect planning. Therefore, it can be useful to decode images into a latent space reflecting the structure of the state space before using them in PPO. In this case, the latent space should be properly regularised.

Chapter 3

Learning from Hand-Crafted Feature Vectors

An observation space, decoded as a set of feature vectors, can be either hand-crafted or deduced from visual observations. In this chapter, we concentrate on the former approach.

Traditionally, manipulation problems are framed on an MDP with a hand-crafted state space. However, this approach has a significant drawback: a state space has to be engineered for each environment, which impedes its applicability and usually requires domain-dependent knowledge.

Another issue is a limitation of current machine learning algorithms. The size of feature vectors has to be constant in every instance of the environment, which can not be the case if the feature vectors are drawn from the state space. For example, in an environment where each trial can have a different number of blocks to manipulate, the size of feature vectors is not constant.

In this chapter, we show that an environment framed on a POMDP instead of an MDP can overcome that limitation. We present and examine several constant-size observation spaces for an environment where the number of blocks can vary. Further, we concentrate our efforts on learning policies that can be applied to untrained tasks.

3.1 Introduction

First, we introduce several definitions that we will use extensively throughout the chapter.

Definition 3.1. An *environment* is a union of all the possible POMDPs representing a learning problem, where the state and action spaces are fixed, and only the initial and goal states can change. On each trial, the environment generates a new instance. We use this term in Reinforcement Learning sense.

Definition 3.2. An *instance* of the environment is a single POMDP that has a single initial and goal state.

Definition 3.3. We say that an observation is *instance-invariant* if it is represented by a feature vector that has a constant size. When the observation is encoded as a feature vector which size might not be constant, we say the observation is *instance-dependent*.

Traditionally, in planning problems framed as an MDP, a state space, which can be defined as a set of hand-crafted feature vectors, is usually comprised of the configurations of a manipulator and all objects (Section 1.2). However, this approach has a significant drawback: a state space has to be engineered for each environment, which impedes generality and usually requires domain-dependent knowledge. Also, it can be hard to incorporate the geometry of an environment into feature vectors.

Another issue is a limitation of current machine learning algorithms. The size of feature vectors has to be constant in every instance of an environment. That is, feature vectors have to be instance-invariant. In the MDP, such a constrain holds only if the state space is instance-invariant, which might not be the case in complicated problems. For example, for an environment in which each trial can have a different number of blocks to manipulate, the size of feature vectors is not constant.

In this chapter, we relax the assumption that the state space is instance-invariant. Our main contribution that we construct and examine instance-invariant observation spaces for an instance-dependent state space. In Section 3.2, we formally introduce our environment framed as a POMDP. Then, in Section 3.3, we discuss several possible ways of construction instance-independent observation spaces, which we test in our environment in Section 3.5.

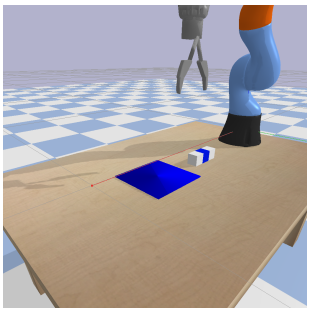

FIGURE 3.1: A frame of the Kuka environment.

3.2 Framework Formulation

We created a virtual environment simulating a Kuka IIWA robotic arm [50]. The manipulator has seven degrees of freedom. An agent can control the configuration χ_{gr} of a 2-finger gripper such that

$$\chi_{\text{gr}} = x \times y \times z \times \alpha \in \mathbb{S} \times \mathbb{R}^3, \tag{3.1}$$

where x, y, z are the gripper's Cartesian coordinates in the world frame, and α (yaw) is a Euler angle. To simplify the problem, we always keep the arm vertically; thus, Euler angles β (roll) and γ (pitch) are constant. Additionally, the configurations of all the arm's other joints are set by inverse kinematics.

We mounted the arm on the top of a table (Fig 3.1). The table has enough space to accommodate any configuration and number of blocks. In our experiments, we used cube-shaped rigid blocks, having the same size. All the blocks are affected by gravity. Each block with the configuration χ_o is defined by the set of coordinates x, y, z and orientation quaternion q:

$$\chi_o = x \times y \times z \times q \in \mathbb{R}^7. \tag{3.2}$$

The agent operates the arm by setting shifts

$$\Delta = \delta x \times \delta y \times \delta z \times \delta\alpha \in \mathbb{R}^4. \tag{3.3}$$

We consider a very general case in which the shifts are not bound. Our goal is to learn the physical bounds at each pose implicitly, which would free the designers of robotic tasks from complex specifications.

Additionally, without the loss of generality, we scripted a grasping movement to keep the fingers' control out of the action space. That is, when the arm is in the proximity of a goal block, the script closes the arm's fingers.

Therefore, an instance of the environment with n rigid blocks has the configuration space $\mathcal{X}$, where $\mathcal{X} = \{\chi \,|\, \forall\, \chi \in \mathbb{R}^{4+7 \cdot n}, \forall n \in \mathbb{N}_0\}$ is a union of all possible configurations χ such that

$$\chi = \chi_{\text{gr}} \times \chi_o^1 \times ... \times \chi_o^n \in \mathbb{S} \times \mathbb{R}^{3+7 \cdot n}. \tag{3.4}$$

We frame the environment on a union of POMDPs with tuples $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O)$, where a single initial $s_{\text{init}} \in \mathcal{S}$ and goal $X_{\text{goal}} = \chi_{\text{gr}}$ state is assigned to each POMDP.

The state space is $\mathcal{S} = \mathcal{X}$. It is not instance-invariant because each instance can have a different number of blocks. At time step t, an instance containing the robotic arm and n objects has a state s_t in the configuration space $\mathcal{X}$.

Time is discretised in the following manner. When the agent has requested an action a_t, the environment terminates the time step t. Then it applies the action and simulates the changes. After that, the environment starts a new time step $t+1$ and returns a new state s_{t+1}. Then it freezes the simulation until it receives a new action a_{t+1}.

Later, we specify action $\mathcal{A}$ and observations Ω spaces as well as the reward function R. Other aspects of the framework are the same as in Section 2.1.

3.2.1 Action Space

The action space $\mathcal{A}$ contains two actions $\{\mathcal{A}_1, \mathcal{A}_2\}$:

- **The arm's translation.** To translate the arm, preserving its orientation, the agent can apply a shift $\Delta_1 = (\delta x, \delta y, \delta z, 0) \in \mathbb{R}^3$. Then, the next time step $t+1$, the arm will move to the state with coordinates $u_{t+1} = u_t + \Delta_1$, where u_t is the arm's coordinate in the previous time step t. Thus, $\mathcal{A}_1 = \{\Delta_1 \,|\, \forall\, \delta x, \delta y, \delta z \in \mathbb{R}\}$.
- **The arm's rotation.** To rotate the arm (yaw), preserving its spatial position, the agent can apply a shift $\Delta_2 = (0, 0, 0, \delta\alpha) \in \mathbb{R}$. Then, the next time step $t+1$, the arm will change its orientation by $q_{t+1} = q_t + \Delta_2$, where q_t is the arm's orientation in the previous time step t. Thus, $\mathcal{A}_2 = \{\Delta_1 \,|\, \forall\, \delta\alpha \in \mathbb{R}\}$.

Both actions can be applied at the same time step; thus, the action space $\mathcal{A}$ is equal to $\mathcal{A}_1 \cup \mathcal{A}_2$.

3.2.2 Reward Function

We define reward R as a function at a time step t:

$$R : X_{\text{goal}} \times X_{\text{grip}} \times \chi_{o_1} \times ... \times \chi_{o_n} \to \mathbb{R}\,, \tag{3.5}$$

where $X_{\text{goal}} = \chi_{\text{gr}}/q$ and $X_{\text{grip}} = \chi_{\text{gr}}/\alpha$ are the world coordinates of the goal and the gripper.

The function we use in our experiments is

$$R_t = -a|X_{\text{goal}} - X^t_{\text{grip}}|^2 - b\lambda \sum_{i=1}^{n} |\chi^t_{o_i} - \chi^0_{o_1}|^2 + \text{C}\,. \tag{3.6}$$

The first term guides the agent to the goal. The constant C is used to reward a completed task. The third term is added to discourage the agent from moving the surrounding blocks, where the parameter λ regulates its strength. Constants a, b are normalisation coefficients.

In this chapter, we use a hand-crafted reward function. However, in Chapter 4, we propose a method to generate rewards automatically. Further, we define several observation spaces Ω for our instance-dependent state space $\mathcal{S}$.

3.3 Observation Space

In this chapter, we examine the generalisation abilities of instance-independent observation spaces Ω derived from an instance-dependent state space $\mathcal{S}$. Thus, in this section, we define three possible observation spaces. Two of them are instance-invariant and one instance-dependent. Later, we omit t for simplicity.

3.3.1 Instance-Dependent Observation Space

As a baseline, we define an instance-dependent observation space Ω that is equal to the state space $\mathcal{S}$, which effectively reduces our problem to an MDP. The agent receives the goal's coordinates X_{goal} in the world frame and a state vector $\chi = s \in S$. Therefore, it is a deterministic perfect information problem.

Additionally, we augment the state vector by adding the goal coordinates X_{goal}. Thus, the observation space Ω is a union of all possible configurations of

$$X_{\text{goal}} \times \chi \in \mathbb{S}^2 \times \mathbb{R}^{6+7n}\,.$$

Thus, the size of the space depends on the number of blocks n. In our experiments, we label this space as *all in obs*.

3.3.2 Instance-Invariant Observation Space

Later, we construct several instance-invariant observation spaces.

3.3.2.1 No Blocks in Observations

A straightforward approach can be to exclude all the block from observations because their number can vary. Thus, the state space $\mathcal{S}$ is partially observable, and the agent receives the goal's coordinates X_{goal} in the world frame and the gripper's configuration χ_{gr}. Therefore, it is a deterministic imperfect information problem.

The observation space Ω is a union of all possible configurations of

$$X_{\text{goal}} \times \chi_{\text{gr}} \in \mathbb{S}^2 \times \mathbb{R}^6\,.$$

Thus, the size of the space is constant. In our experiments, we label this space as *none in obs*.

However, such an observation space is ignorant of the location of objects, which impairs the agent's ability to manipulate these objects. In the next section, we propose another observation space without such limitations.

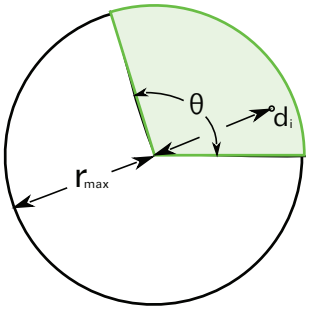

FIGURE 3.2: An illustration of a sensing sphere with the radius $r_{\max}$. The gripper is situated at the origin. The green region represents a sensing sector. For simplicity, the angle ϕ is omitted. The sensing sector returns the distance d_i to the closest object in that sector. If there is no object, the sector returns $r_{\max}$.

3.3.2.2 Sensing Sphere

Because our goal is to construct an observation space Ω that has information about any number of blocks yet constant in size, we propose only to keep the closest blocks to the arm. Thus, the agent has information about the environment within a sensing radius $r_{\max}$ and ignorant of the world beyond. For 3D problems, the sensing radius can be viewed as a sensing sphere, hence the name.

We place the sphere's origin is at the gripper's centre of mass. To keep a constant number of tracking blocks, we further subdivide the sphere into $a \times o$ sectors, where we keep the distance to the closes block b_p in each sector. Additionally, if there is no block within that volume of space v, we keep the sensing distance $r_{\max}$. We define a as the number of meridians and o as the number of parallels on the sphere, which intersections form 3D sectors. Thus, each sector has $\theta = 2\pi/a$ and $\phi = 2\pi/o$. Figure 3.2 illustrates our idea.

Mathematically, for each object $b_p \in v$ and a distance $d_i \equiv d_i(X_{\text{gr}}, b_p)$, the sensing sphere $\tilde{S}_{a,o}$ is defined as a matrix

$$\tilde{S}_{a,o} = \Big(s_{i,j} = \min(d_1, d_2, ..., d_q, r_{\max}) \Big) \in \mathbb{R}^{a \times o}, \tag{3.7}$$

where $a, o \in \mathbb{N}$ represent the number of sectors in l$a$titudinal and l$o$ngitudinal angles.

Thus, the observation space Ω is constructed as a union of all configurations of

$$\tilde{X}_{\text{goal}} \times \chi_{\text{gr}} \times \tilde{S}_{a,o} \in \mathbb{R}^{3+7+a \times o},$$

where $\tilde{X}_{\text{goal}}$ is the Cartesian coordinates of a goal in the gripper's frame, and the choice of the parameters a, o is instance-independent. In our experiments, we label this space as *sens*.

Additionally, we can exclude the gripper's configuration χ_{gr} from the observations. Because χ_{gr} is the only one variable defined in absolute values, that can make a new observation space more general. Thus, the observation space Ω is constructed as a union of all configurations of

$$\tilde{X}_{\text{goal}} \times \tilde{S}_{a,o} \in \mathbb{R}^{3+a\times o}\,.$$

We label this space as *sens, no gr*.

3.4 Evaluation Tasks

Our goal was to construct an observation space that can be used for learning policies for high-level planning operations in highly constrained scenarios in an instance-dependent environment. That is, our goal was to learn policies for picking and placing objects when the number of blocks could vary. Bearing that in mind, we defined two evaluation tasks: *Pick* and *Place*, discussed in the following sections.

3.4.1 Pick

In the first task, the agent's goal was to pick a blue block without toppling a tower (Figure 3.3-A). This task can be challenging if the learned policy is used in an environment with a tower of different height. Because the tower is made of blocks, it changes the number of object in the environment, hence its state space. Thus, because we cannot learn an individual policy for each state space (i.e. the number of objects), our goal was to create a general policy working in general observation space.

To make the task more challenging, the tower could be adjacent to any sides of the goal block. Also, while we were learning the policy, we kept the tower's height and the manipulator's initial state unchanged. Therefore, in this task, we examined the policy's generalisation ability when the policy was tested in an environment with a different number of blocks.

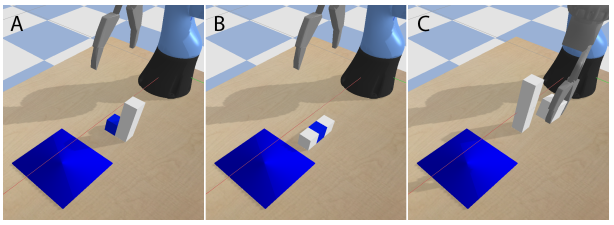

FIGURE 3.3: (A)(B) Possible initial states of the environment used in the task '*Pick*'. (C) A possible initial state of the environment used in the task '*Place*'.

Additionally, we tested each policy on a different initial state (Figure 3.3-B) where two blocks were adjacent to opposite sides of the goal. Because we did not train the policy in such block configurations, it would require more general policy to perform well. We denote that test case as *row*.

Additionally, we used a single initial configuration for the manipulator. That is, the arm's initial position was constant in all trials. In contrast, we varied the coordinates of the blue block. That is, the positions of the goal block and the tower could move across the table.

In our experiments, we label this task as *pick* x *on* y. Here, x denotes the tower's height that we used in training and y the height that we used in testing.

3.4.2 Place

In the second task, the agent's goal was to place a block on the top of a tower (Figure 3.3-C). As before, we could vary the complexity by changing the tower's height. However, by varying the height, we also change the elevation of a goal state, which adds to the complexity of the problem. To isolate and study this additional complexity, we designed two different sets of initial states.

Fig 3.4-A shows the first subset, which is more challenging. We put the arm, holding the red-contoured block, at the same coordinate position each trial, while the tower could move across the table. Therefore, not only the distance to the goal but also the goal's elevation could vary with the tower's height. Thus, when the policy was tested on a tower of another height, it required a more general policy that could elevate the block. In our experiments, we label the first subset as *place* x *on* y*, absolute*, where x denotes the tower's height that we used in training and y denotes the height that we used in testing.

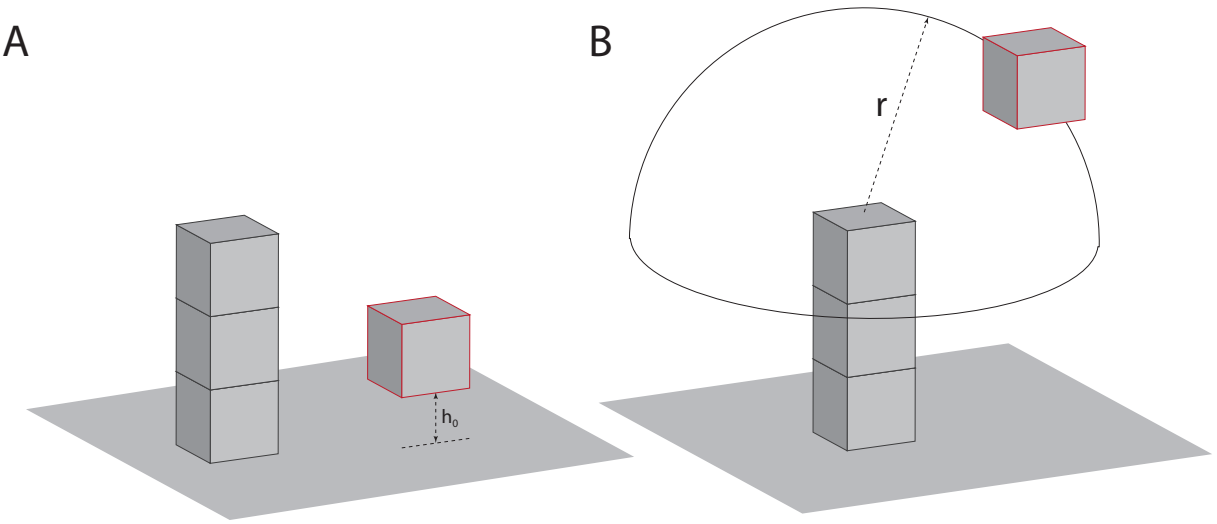

FIGURE 3.4: (A) An illustration of the red-edged block having a single initial state. The manipulator holds the red-edged block, and the block has constant coordinates every trial. (B) An illustration of the red-edged block having a set of initial state. The manipulator holds the red-edged block, and the block is located on the half-sphere with radius r every trial.

Fig 3.4-B shows the second subset of initial states, in which we kept the arm above the goal in the vicinity of the tower on a half-sphere with radius r. It reduced the task's complexity because the elevation height of the goal did not depend on the height of the tower. In our experiments, we label this case as *place x on y, relative*, where x denotes the tower's height that we used in training and y denotes the height that we used in testing.

3.5 Experimental Results

As we mentioned before, our goal was to construct an observation space that can be used for learning policies for high-level planning operations in highly constrained scenarios in an instance-dependent environment. That is, our goal was to learn policies for picking and placing objects when the number of blocks could vary. We were particularly interested in policies that can work in instances with the number of blocks that were not used in training. We were interested in finding the connection between the policy's generalisation ability and the used observation space. Therefore, in this section, we present the performance of the polices equipped with three observation spaces defined in Section 3.3 in the '*Pick*' and '*Place*' tasks, defined in Section 3.4.

We trained all the policies using a reward function R_t such that

$$R_t = -10|X^t_{\text{goal}} - X^t_{\text{grip}}|^2 + C - 100\lambda \sum_{i=1}^{n} |\chi^t_{o_i} - \chi^0_{o_i}|^2, \tag{3.8}$$

where the constant C rewards the agent with extra 50 points if the task has been completed successfully and discourages with -1 point if the task has failed. The third term was added to discourage the agent from moving the surrounding blocks, where the parameter λ regulates its strength. We examined the impact of λ in the '*Pick*' task, but we kept $\lambda = 0$ in the '*Place*' task.

We used four indicators to evaluate each policy over 2000 trials:

- **Success rate** - the fraction of the successful policy runs in the total number of runs. It measures the robustness of a learned policy on a test set.
- **Mean policy execution trace** - the average number of time steps in a policy. The optimal policy should have the length that does not depend on the tower's height.
- **Mean displacement** - the total distance at which the adjacent blocks have been relocated during the execution of a policy. It measures the impact of the policy on the surrounding blocks. The intuition is that while picking or moving a block, the arm could collide with surroundings, hence damaging them. Therefore, a better policy should have a lower displacement.
- **Mean displacement in successful runs** - the average displacement measured for successful runs. Because failed runs can significantly affect the displacement, this indicator, when compared with the mean displacement, shows how destructive is the policy in failed runs.

3.5.1 Learning Setup

As discussed in Chapter 2, there are two first-order learning algorithms: DDPG and PPO that can work with continuous action and state spaces, which is required in our environments. To choose between them, we compared their performance on '*Pick*' task, illustrated in Figure 3.5.

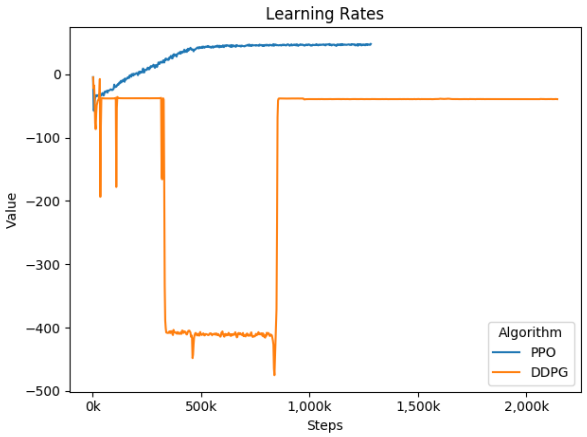

FIGURE 3.5: PPO vs DDPG performance. The cummultive rewards for '*Pick*' task.

On the same task, PPO showed better results in producing a working policy. In contrast to PPO, the training of DDPG was unstable due to the gradient explosion discussed in Section 2.2.4. Thus, we chose to employ PPO in the following experiments.

We used the off-the-shelf PPO and DDPG implementations from `rllib`, a Python package [51], with the following hyperparameters:

- $\tilde{S}_{16,8}$ was used for the sensing sphere. This parameter controls the perceptive ability of the agent.
- The sensing radius was equal to 40 blocks.
- For PPO and DDPG, we used the default parameters suggested by the authors. However, for PPO, we reduced `sample_batch_size` to 40 and `train_batch_size` to 2500. We observed that by reducing those parameters, we increased the training speed yet not impaired the policy convergence.

All the policies were trained on an `m5.24xlarge`[1] cloud instance from AWS Cloud Computing Services. The instance contained 2 Intel Xeon processors with 48 cores and 96 threads. Such an instance performs 3.08 Tflops (trillion floating-point operations per second) in Single float precision General Matrix Multiply (SGEMM) test. It is 8.8 times more powerful than MacBook Pro's Intel Core i5 performance (0.35 Tflops), on which the policies were tested. The Intel Core i5-8259U has 4 cores and 8 threads.

[1] https://aws.amazon.com/ec2/instance-types/m5/

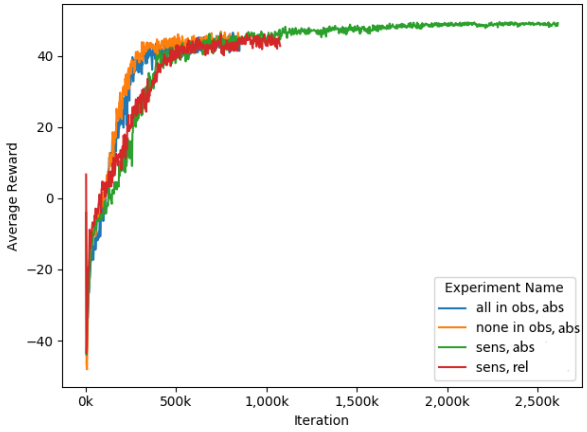

FIGURE 3.6: The cumulative reward for the '*Place*' task with $\lambda = 0$.

Exp Name	Training Time
none in obs, absolute	42m 39s
all in obs, absolute	41m 38s
sens, absolute	2h 17m 26s
sens, relative	29m 42s
sens, relative, no gr	31m 16s

TABLE 3.1: Training times for the '*Place*' task.

3.5.2 Performance Evaluation

This section presents empirical results comparing the performances of three observation spaces in the '*Place*' and '*Pick*' tasks. As a baseline, we used the instance-dependent observation space *all in obs*.

3.5.2.1 Place

We begin with the '*Place*' task. We concentrated on learning policies that can be applied to more general problems. All the policies were trained with a 4-block tower with the reward with $\lambda = 0$[2] (Equation 3.8). The results can be seen in Table 3.2 and in Figure 3.10. Their cumulative rewards can be found in Figure 3.6, along with their training times in Table 3.1.

First, we tested the policies' performance in the environment with the tower of the same height. In Table 3.2, it is labelled as *place 4 on 4*. All three observation spaces showed similar performance, but the sensing sphere with relative initial states (*sens, relative*) was harder to train, which was reflected in its success rate. However, we observed that the performance was comparable with the baseline (*all in obs*) if the gripper's configuration χ_{gr} was excluded from observations (*sens, relative, no gr*). This could be attributed to the fact that the configuration χ_{gr} was only parameter defined in absolute coordinates.

[2]We examined the impact of λ in the task '*Pick*' (Section 3.5.2.2).

Exp Name	Success Rate	Policy Execution Trace	Block Displacement	Displ in successful runs
place 4 on 9				
sens, relative, no gr	$0.8305 \pm 1.99\%$	$9.8760 \pm 2.27\%$	$0.0263 \pm 43.35\%$	$0.0050 \pm 10.00\%$
place 4 on 8				
sens, relative	$\mathbf{0.5940} \pm 3.62\%$	$16.5120 \pm 3.68\%$	$0.7493 \pm 7.74\%$	$0.0124 \pm 39.52\%$
sens, relative, no gr	$\mathbf{0.8915} \pm 1.53\%$	$9.2815 \pm 2.08\%$	$0.0212 \pm 47.17\%$	$0.0045 \pm 24.44\%$
place 4 on 7				
sens, relative	$0.7360 \pm 2.62\%$	$12.3020 \pm 3.69\%$	$0.3270 \pm 12.11\%$	$0.0105 \pm 30.48\%$
sens, relative, no gr	$0.9090 \pm 1.39\%$	$9.3090 \pm 2.08\%$	$0.0202 \pm 42.57\%$	$0.0034 \pm 11.76\%$
place 4 on 6				
sens, relative	$0.8460 \pm 1.87\%$	$10.2285 \pm 3.04\%$	$0.1385 \pm 17.98\%$	$0.0092 \pm 17.39\%$
sens, relative, no gr	$0.9290 \pm 1.22\%$	$8.9585 \pm 1.86\%$	$0.0125 \pm 42.40\%$	$0.0047 \pm 14.89\%$
place 4 on 5				
sens, absolute	$0.4610 \pm 4.75\%$	$16.7880 \pm 3.76\%$	$0.6122 \pm 5.72\%$	$0.0231 \pm 21.21\%$
sens, relative	$0.8950 \pm 1.50\%$	$9.5035 \pm 2.24\%$	$0.0547 \pm 25.41\%$	$0.0117 \pm 13.68\%$
sens, relative, no gr	$0.9350 \pm 1.16\%$	$8.7625 \pm 1.72\%$	$0.0107 \pm 41.12\%$	$0.0056 \pm 12.50\%$
place 4 on 4				
all in obs, absolute	$0.9880 \pm 0.49\%$	$5.5845 \pm 0.80\%$	$0.0066 \pm 6.06\%$	$0.0065 \pm 6.15\%$
none in obs, absolute	$\mathbf{0.9955} \pm 0.29\%$	$\mathbf{5.8480} \pm 0.51\%$	$\mathbf{0.0062} \pm 4.84\%$	$0.0062 \pm 4.84\%$
sens, absolute	$0.9800 \pm 0.62\%$	$5.8230 \pm 1.26\%$	$0.0109 \pm 14.68\%$	$0.0102 \pm 5.88\%$
sens, relative	$0.9105 \pm 1.37\%$	$9.9375 \pm 1.74\%$	$0.0317 \pm 29.34\%$	$0.0115 \pm 13.04\%$
sens, relative, no gr	$0.9310 \pm 1.19\%$	$8.6985 \pm 1.82\%$	$0.0158 \pm 36.08\%$	$0.0082 \pm 24.39\%$
place 4 on 3				
none in obs, absolute	$\mathbf{0.0350} \pm 23.14\%$	$\mathbf{39.0750} \pm 0.59\%$	$\mathbf{0.3873} \pm 0.59\%$	N/A
sens, absolute	$0.9190 \pm 1.27\%$	$9.4490 \pm 2.83\%$	$0.0057 \pm 7.02\%$	$0.0057 \pm 7.02\%$
sens, relative	$0.9110 \pm 1.37\%$	$10.9650 \pm 1.40\%$	$0.0134 \pm 29.85\%$	$0.0094 \pm 7.45\%$
sens, relative, no gr	$0.9140 \pm 1.35\%$	$9.6740 \pm 2.63\%$	$0.0125 \pm 31.20\%$	$0.0081 \pm 9.88\%$
place 4 on 2				
sens, absolute	$0.5675 \pm 3.82\%$	$28.1225 \pm 1.84\%$	$0.0021 \pm 38.10\%$	$0.0023 \pm 13.04\%$
sens, relative	$\mathbf{0.8920} \pm 1.52\%$	$13.4160 \pm 1.82\%$	$0.0081 \pm 8.64\%$	$0.0086 \pm 8.14\%$
sens, relative, no gr	$\mathbf{0.7650} \pm 2.43\%$	$13.9785 \pm 3.71\%$	$0.0134 \pm 32.09\%$	$0.0092 \pm 8.70\%$
place 4 on 1				
sens, relative	$0.7685 \pm 2.41\%$	$20.8295 \pm 2.12\%$	$0.0061 \pm 24.59\%$	$0.0065 \pm 9.23\%$

TABLE 3.2: The experimental results measuring the performance of the learned policies. The task is to place a goal block on the top of a tower. The block displacement represents the mean distance that the surrounding blocks have been moved during the execution of a policy. The displacement in successful runs is the mean block displacement measured in successful runs. The experiment names represent different observation spaces. The notations are: *none in obs* - no blocks in observations; *all in obs* - all the blocks in observations; *sens* - a sensing sphere; *absolute* - a subtask with the gripper's absolute initial configuration; *relative* – a subtask with the gripper's relative initial configuration; *no gr* – no gripper coordinates in the observations. The notation *place 4 on x* means that we trained a policy using a 4-block tower and tested using an x-block tower. We used $\lambda = 0$ in the reward function.

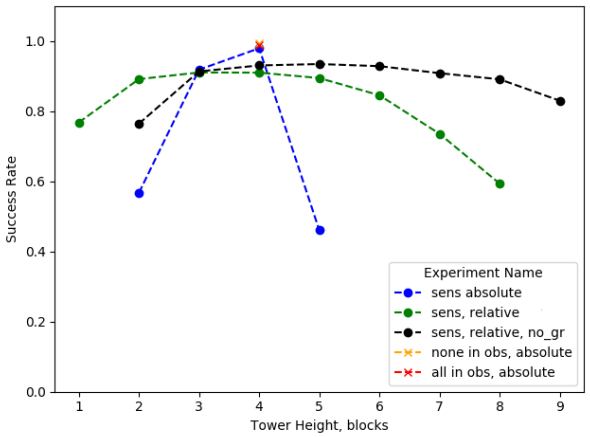

FIGURE 3.7: The experimental results for the task of placing a goal block on the top of a tower. The experiment names represent different observation spaces. The notations are: *none in obs* - no blocks in observations; *all in obs* - all the blocks in observations; *sens* - a sensing sphere; *absolute* - a subtask with the gripper's absolute initial configuration; *relative* – a subtask with the gripper's relative initial configuration; *no gr* – no gripper coordinates in the observations.

Next, we examined the policies' performance on towers with different numbers of blocks. The policy *all in obs* could not be easily applied to them because its observation space contains the coordinates of all the blocks (i.e. instance-dependent). Thus, for example, if we reduce the height from four to three blocks, the observation space will shrink. Unfortunately, RL algorithms require a constant size of the space. Thus, *all in obs* was excluded from further tests.

The policy *none in obs* could be tested on a tower of any height, but we observed a marginal performance, which can be seen in *place 4 on 3*. Therefore, it was excluded from further tests as well.

On the other hand, the policies equipped with a sensing sphere showed a better generalisation ability. For them, we kept only the result with success rate bigger than 50% in the table. Concerning the subtask with a absolute initial height of the gripper, we observed that the policy (*sens, absolute*) could better generalise to the towers of a shorter height. Examining the policy's behaviour, we found that, for the towers taller than the trained height, the policy tended to topple the tower while it was elevating the block, hence the drop in the success rate (as shown in Figure 3.8-A and Video 3).

It can be explained as follows. During the training, the arm's initial state was the same: hovering above the ground at a constant height. Thus, the agent learned to move the arm along the shortest trajectories that could cross a taller tower. However, by design

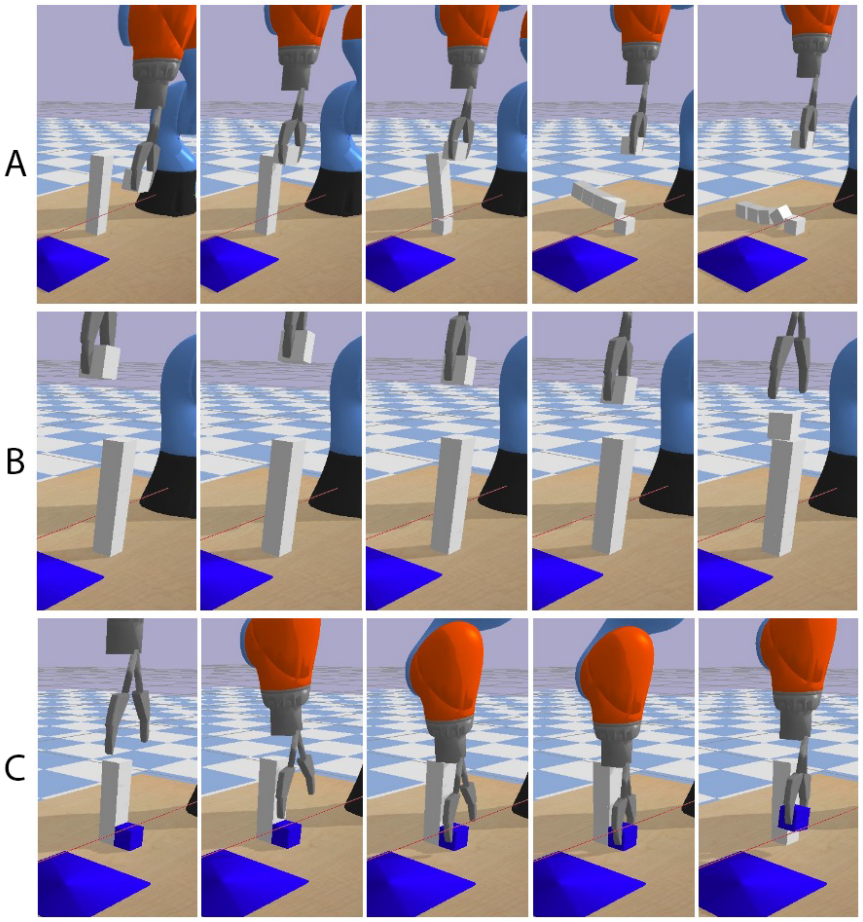

FIGURE 3.8: (A) A demo showing the policy execution of *sens, absolute* on *place 4 on 5* with $\lambda = 0$. (B) A demo showing the policy execution of *sens, relative* on *place 4 on 5* with $\lambda = 0$. (C) A demo showing *sens* policy execution in the task *pick 3 on 4* with $\lambda = 0$.

of the second task, a similar issue could not be observed in the second subtask with a relative initial height of the gripper (*sens, relative*), which performed better in these tests.

Additionally, we improved the success rate for towers taller than the trained one when we excluded the gripper's configuration from the observation space (*sens, relative, no gr*), which was measured in absolute coordinates. However, we observed a diminishing performance for the towers shorter than the trained one. Figure 3.8-B and Video 4 demonstrate its policy execution in *place 4 on 5*.

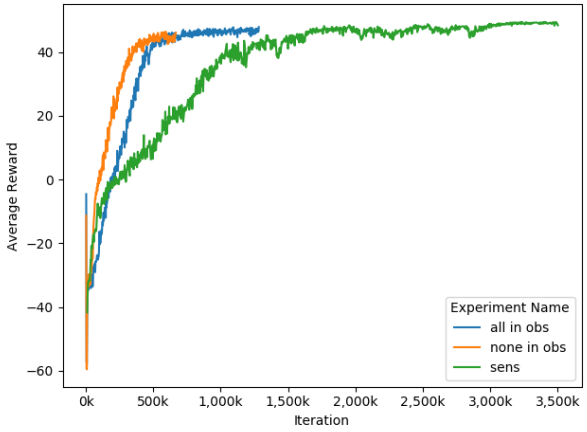

FIGURE 3.9: The cumulative rewards for the '*Pick*' task with $\lambda = 0$.

Exp Name	λ	Training Time
none in obs	0	26m 51s
none in obs	1/16	39m 35s
all in obs	0	51m 2s
all in obs	1/16	1h 8m 17s
sens	0	2h 1m 12s
sens, no gr	0	1h 58m 11s

TABLE 3.3: Training times for the '*Pick*' task.

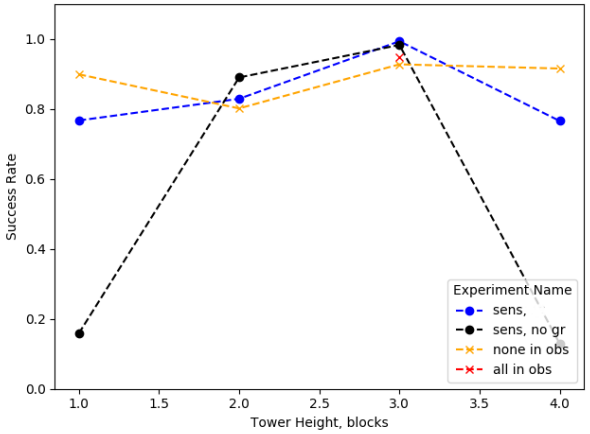

FIGURE 3.10: The experimental results for the task of picking a goal block adjacent to a tower. In this experiment, we treated a trial as successful if the agent had picked the block regardless the state of the tower. The experiment names represent different observation spaces. The notations are: *none in obs* - no blocks in observations; *all in obs* - all the blocks in observations; *sens* - a sensing sphere; *no gr* – no gripper coordinates in the observations. We used $\lambda = 0$ in the reward function.

3.5.2.2 Pick

Second, we examined the '*Pick*' task. We concentrated on learning policies that can be applied to more general problems. Additionally, we measured the impact of the parameter λ in the reward function (Equation 3.8). All the policies were trained with a 3-block tower. The results can be seen in Table 3.4. Their cumulative rewards can be found in Figure 3.9, along with their training times in Table 3.3.

In this task, we treated a trial as successful if the agent picked the block regardless of the final state of the tower. At first, we tested the policies' performance in the environment

Exp Name	Success Rate	Policy Execution Trace	Block Displacement	Displ in successful runs
pick 3 on 4				
none in obs	$0.9160 \pm 1.33\%$	$5.0180 \pm 1.55\%$	$\mathbf{1.6756} \pm 2.04\%$	$1.6835 \pm 2.12\%$
sens	$\mathbf{0.7660} \pm 2.43\%$	$15.7885 \pm 3.86\%$	$1.0557 \pm 4.20\%$	$1.2356 \pm 4.02\%$
sens, no gr	$0.1290 \pm 11.40\%$	$35.3165 \pm 1.35\%$	$0.2623 \pm 10.87\%$	$1.4585 \pm 7.09\%$
pick 3 on 3				
none in obs	$0.9280 \pm 1.22\%$	$5.0420 \pm 1.53\%$	$1.3521 \pm 1.96\%$	$1.3430 \pm 2.03\%$
all in obs	$0.9475 \pm 1.03\%$	$4.7490 \pm 4.02\%$	$1.1409 \pm 2.86\%$	$1.1724 \pm 2.79\%$
sens	$0.9940 \pm 0.34\%$	$4.3355 \pm 1.70\%$	$0.9912 \pm 3.42\%$	$0.9894 \pm 3.44\%$
sens, no gr	$0.9835 \pm 0.57\%$	$9.2715 \pm 2.17\%$	$0.8409 \pm 4.10\%$	$0.8373 \pm 4.16\%$
pick 3 on 2				
none in obs	$0.8020 \pm 2.18\%$	$4.9175 \pm 1.54\%$	$1.0264 \pm 2.18\%$	$0.9953 \pm 2.43\%$
sens	$\mathbf{0.8295} \pm 1.99\%$	$12.4740 \pm 3.83\%$	$\mathbf{0.2832} \pm 6.96\%$	$0.3031 \pm 7.29\%$
sens, no gr	$0.8905 \pm 1.54\%$	$13.1715 \pm 3.46\%$	$0.4061 \pm 6.03\%$	$0.4252 \pm 6.14\%$
pick 3 on row				
none in obs	$0.8995 \pm 1.47\%$	$4.9850 \pm 1.53\%$	$\mathbf{0.4477} \pm 3.19\%$	$0.4404 \pm 3.27\%$
sens	$\mathbf{0.7675} \pm 2.41\%$	$10.3915 \pm 4.32\%$	$0.2562 \pm 4.29\%$	$0.2800 \pm 4.64\%$
sens, no gr	$0.1585 \pm 10.09\%$	$22.6970 \pm 2.99\%$	$0.2579 \pm 7.33\%$	$0.2358 \pm 16.54\%$

TABLE 3.4: The experimental results measuring the performance of the learned policies. The task is to pick a goal block adjacent to a tower. In this experiment, we treated a trial as successful if the agent had picked the block regardless the state of the tower. The block displacement represents the mean distance that the surrounding blocks have been moved during the execution of a policy. The displacement in successful runs is the block displacement measured in successful runs. The experiment names represent different observation spaces. The notations are: *none in obs* - no blocks in observations; *all in obs* - all the blocks in observations; *sens* - a sensing sphere; *no gr* – no gripper coordinates in the observations. The notation *pick 3 on x* means that we trained a policy using a 3-block tower and tested using an x-block tower. We used $\lambda = 0$ in the reward function.

with the tower of the same height, which is labelled as *pick 3 on 3*. The policy with no blocks in the observations, *none in obs*, showed the success rate below the baseline, *all in obs*. However, both of the policies equipped with a sensing sphere performed better than the baseline. We observed no significant difference whether the configuration of the gripper χ_{gr} was in the observations or not.

Next, we examined the policies' performance on towers with different numbers of blocks. As in the task '*Place*', the policy *all in obs* was excluded from further tests. The policy *none in obs* showed a stable success rate over the tower with both bigger and smaller numbers of blocks. However, after careful examination, we observed that the agent achieved this success rate at the expense of toppling the tower because there was no information about the position of the tower in the observations. That also can be seen in more significant block displacements for the policy *none in obs*.

On the other hand, the policy equipped with the sensing sphere without the configuration

of the gripper, *sens, no gr*, showed marginal results, while the policy *sens* performed at the same level as the policy *none in obs*. The policy *sens, no gr* also showed smaller block displacements than the policy *none in obs*. Figure 3.8-C and Video 5 show an example of its execution.

Additionally, we tested the policies on the initial state *row*, where the agent's task was to pick the middle block from a row of blocks. Here, again, policy *none in obs* demonstrated a higher success rate at the expense of a more significant block displacement.

At the end, we examined the impact of the parameter λ on the success rate and block displacements. As discussed in Section 3.2.2, we added the last term in the reward function to facilitate policies that could pick the goal block without disturbing the tower. We observed that by increasing λ, we made training harder, which was reflected in the diminishing success rates. At the same time, the effect on block displacement was minimal. For example, in case of *all in obs*, the success rate dropped by 33%, and the block displacement decreased by 12%. The results can be found in Table 3.5.

Finally, it worth to note that all the policies moved the tower to some extent. This can be attributed to the fact that there was no sufficient geometry information in our observation spaces because we only decoded the centre of mass coordinated and orientations. We will continue the construction of observation spaces in Chapter 4, where we concentrate on visual observations.

3.5.3 Summary

The state space of a real-world environment can be unknown or instance-dependent. In these cases, it is useful to frame the environment on a POMDP instead of an MDP. It allows us to devise a more general instance-independent set of observations that can be used in a broad range of tasks.

However, it can be a daunting task to construct a suitable observation space because it usually requires domain-dependent knowledge. For example, we demonstrated that the policy constructed by excluding all the surrounding blocks, the number of which could vary, showed marginal results in generalising to untrained tower's heights. At the same time, the policy trained with the sensing sphere as an observation space showed better results in the same task.

Exp Name	λ	Success Rate	Policy Execution Trace	Block Displacement	Displ in successful runs
pick 3 on 3					
none in obs	0	**0.9280** $\pm$ 1.22%	5.0420 $\pm$ 1.53%	1.3521 $\pm$ 1.96%	1.3430 $\pm$ 2.03%
all in obs	0	0.9475 $\pm$ 1.03%	4.7490 $\pm$ 4.02%	1.1409 $\pm$ 2.86%	1.1724 $\pm$ 2.79%
sens	0	0.9940 $\pm$ 0.34%	4.3355 $\pm$ 1.70%	0.9912 $\pm$ 3.42%	0.9894 $\pm$ 3.44%
sens, no gr	0	0.9835 $\pm$ 0.57%	9.2715 $\pm$ 2.17%	0.8409 $\pm$ 4.10%	0.8373 $\pm$ 4.16%
none in obs	1/36	0.9440 $\pm$ 1.07%	3.9535 $\pm$ 0.48%	1.2387 $\pm$ 2.52%	1.2234 $\pm$ 2.63%
all in obs	1/36	0.9150 $\pm$ 1.33%	9.4805 $\pm$ 3.93%	1.0337 $\pm$ 3.21%	1.1005 $\pm$ 3.03%
none in obs	1/25	0.4025 $\pm$ 5.34%	3.1970 $\pm$ 9.11%	0.9162 $\pm$ 3.44%	0.8642 $\pm$ 6.41%
all in obs	1/25	0.6450 $\pm$ 3.26%	6.6755 $\pm$ 8.20%	0.8426 $\pm$ 4.00%	0.9022 $\pm$ 4.62%
sens	1/25	**0.8910** $\pm$ 1.54%	4.1590 $\pm$ 2.17%	1.0501 $\pm$ 3.26%	1.0447 $\pm$ 3.47%
none in obs	1/16	0.4515 $\pm$ 4.83%	2.0190 $\pm$ 1.85%	0.7417 $\pm$ 4.02%	0.5884 $\pm$ 8.46%
all in obs	1/16	0.6385 $\pm$ 3.30%	5.7895 $\pm$ 8.62%	1.0017 $\pm$ 3.51%	1.0118 $\pm$ 4.13%
pick 3 on 2					
none in obs	0	0.8020 $\pm$ 2.18%	4.9175 $\pm$ 1.54%	**1.0264** $\pm$ 2.18%	0.9953 $\pm$ 2.43%
sens	0	0.8295 $\pm$ 1.99%	12.4740 $\pm$ 3.83%	0.2832 $\pm$ 6.96%	0.3031 $\pm$ 7.29%
sens, no gr	0	0.8905 $\pm$ 1.54%	13.1715 $\pm$ 3.46%	0.4061 $\pm$ 6.03%	0.4252 $\pm$ 6.14%
sens	1/25	0.7165 $\pm$ 2.76%	14.1890 $\pm$ 0.60%	0.5665 $\pm$ 4.01%	0.5957 $\pm$ 4.73%

TABLE 3.5: The experimental results measuring the impact of λ in the reward function. The task is to pick a goal block adjacent to a tower. In this experiment, we treated a trial as successful if the agent had picked the block regardless the state of the tower. The block displacement represents the mean distance that the surrounding blocks have been moved during the execution of a policy. The displacement in successful runs is the block displacement measured in successful runs. The experiment names represent different observation spaces. The notations are: *none in obs* - no blocks in observations; *all in obs* - all the blocks in observations; *sens* - a sensing sphere; *no gr* – no gripper coordinates in the observations. The notation *pick 3 on x* means that we trained a policy using a 3-block tower and tested using an x-block tower.

However, this approach has several limitations. Due to the nature of the observation space, the sensing sphere cannot be easily transferred to another problem. Also, it does not fully capture the geometry to the problem. Moreover, it can be a daunting task to hard-code geometry into the observation space in a way that this space can be applied to different problems. Therefore, it can be beneficial to use visual observations instead of hand-crafted features, which is our goal in Chapter 4.

Chapter 4

Learning from Visual Observations

Hand-crafted reward functions are common in Reinforcement Learning. However, because a reward designed for one task usually cannot be applied to another, the algorithms cannot be general enough to be transferred to another problem. Thus, it can be expedient to auto-generate rewards from visual observations, which are universal.

We can construct an auto-generated reward if the distance between visual observations can be measured. For this purpose, we can use raw pixels, but such an observation space does not reflect the structure of the hidden state space. On the other hand, we can compress images to their latent representations, which can be used as feature vectors.

In this chapter, we propose a new domain-independent, auto-generated reward function, assessing state-similarity to the goal state. However, the correct distance between two latent representations requires regularisation that can preserve the spatial structure of an environment, which we introduce in this chapter. Additionally, we show that our regularisation can work in a variety of environments, where we concentrate our efforts on learning working policies using our auto-generated reward function.

4.1 Introduction

As discussed in Chapter 3, designing a proper observation space can be a daunting task. Therefore, it would be advantageous to deduce feature vectors from visual observations. For this purpose, we can use raw pixels, but such an observation space does not reflect the structure of the hidden state space. On the other hand, we can compress images to their latent representations, which can be used as feature vectors.

Additionally, in the state of the art, visual observations are usually made by one camera. However, in manipulation problems, the agent usually operates in three dimensions, which cannot be properly captured by one camera. In such cases, it can be possible to extract more useful features if we use two cameras. However, it can bring unexpected irregularities into the latent space; thus, it can impede the quality of feature vectors.

On the other hand, hand-crafted reward functions are common in the state of the art (Section 1.2). Defining a reward function for each task can be problematic and usually requires domain-specific knowledge. Therefore, auto-generated rewards can facilitate the generality of learning algorithms.

In this chapter, we propose a new domain-independent, auto-generated reward function, measuring state-similarity to the goal state. In Section 4.2, we use a Variational Autoencoder (VAE) to map images to latent space, where a similarity metric is defined. However, the correct similarity distance between two latent states requires regularisation that can preserve spatial directions of a real environment, which is not the case for the vanilla regularisation. Thus, we propose a new regularisation in Section 4.3. Additionally, we show that our regularisation can work with visual observations from two cameras.

Finally, we refine our training framework in Section 4.4 and show that an RL agent can learn meaningful policies using the proposed reward. To show the generality of our method, we apply the same regularisation to a new environment, defined in Section 4.4.2.

4.2 State-Similarity Reward Function

It can be problematic to define a hand-crafted reward function for each task because it usually requires domain-specific knowledge. Therefore, to facilitate the generality of learning algorithms, it can be beneficial to use auto-generated rewards.

In practice, we can define a domain-independent reward function $R(s_t, s_g)$, measuring proximity to the goal state s_g, if the distance between two states s_t, s_g can be measured. That is,

$$R(s_t, s_g) = -d(s_t, s_g). \tag{4.1}$$

For example, in a game of the 15-puzzle, such a distance can be the number of misplaced tiles.

However, in practical planning problems, we do not usually have direct information about the structure of the state space $\mathcal{S}$ due to its complexity. The agent often operates within an observation space Ω that is made from visual observations to be universal.

Although the feature vectors can be constructed from a camera's raw pixels by measuring their intensity, it is unclear how to define the distance between such two feature vectors. Fortunately, we can use a VAE to learn a distribution $p(z|o)$ mapping raw images to a latent space $\mathcal{Z} \in \mathbb{R}^n$, hoping that the latent representation will reflect the structure of the state space. That is, neighbouring states in the state space should be close in the latent space as well.

In this case, we can define a new domain-independent reward function based on a distance to the goal state as

$$R(o_t, o_g) = -d(o_t, o_g) \approx -d(z_t, z_g)\Big|_{z \sim p(z|o)} = -\Big|\Big|z_t - z_g\Big|\Big|^2, \tag{4.2}$$

where $z_t, z_g \in \mathcal{Z}$ are drawn from the distribution $p(z|o)$ and $o_t, o_g \in \Omega$.

The approximation holds if and only if the latent representation $\mathcal{Z}$ reflects the structure of the state space $\mathcal{S}$. That is, if the triangle inequality

$$d(z_i, z_j) < d(z_i, z_k) + d(z_k, z_j)$$

is true for any feature vectors z_i, z_j, z_k sampled from the observations o_i, o_j, o_k of states s_i, s_j, s_k such that

$$d(s_i, s_j) < d(s_i, s_k) + d(s_k, s_j).$$

Unfortunately, it is not the case for the VAE's vanilla regularisation because its latent space $\mathcal{Z}$ is approximated as a normal distribution $\mathcal{N}(0, I)$ by KL-divergence $D_{\mathrm{KL}}\Big(q_x^\theta(z) \,||\, \mathcal{N}(0, I)\Big)$. Moreover, the vanilla VAE treats each observation independently; thus, the spatial organisation of feature vectors is not rigorously defined and can be arbitrary. Therefore, in the next section, we introduce a new regularisation in which the triangle inequality holds at least locally.

4.3 Action Regularisation

We assume that each time step, a learning environment, defined as a POMDP with a tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O)$, generates a set of visual observations $o \in \Omega$ after receiving an action $a \in \mathcal{A}$. Thus, each training episode can be represented as a trajectory τ with horizon $H \in \mathbb{N}$ such that

$$\tau = \{o_0, a_0, o_1, ..., o_{t-1}, a_{t-1}, o_t, ..., o_H\}. \tag{4.3}$$

Such trajectories contain spatial information about the structure of the state space $\mathcal{S}$. That is, when the agent moves to a new state, the state-transition function $s' = T(s, a)$, connecting old and new states, depends on action a. For example, in Figure 4.1, the actions (LEFT, UP, RIGHT) and (RIGHT, UP, LEFT) will bring the agent to the same state. Thus, the state space $\mathcal{S}$ can be ordered in a way that neighbouring states can be reached in a single action. That is, the farther the physical distance to a state, the more actions are needed to reach it. Thus, the triangle inequality holds in such spaces.

However, it can be challenging to deduce such a structure for 3D spaces from visual observations, where each observation is a 2D projection. If we obtain a latent space $\mathcal{Z}$ employing a VAE with the vanilla regularisation, the space $\mathcal{Z}$ will be ordered arbitrarily, especially if more than one image or sensor contributes to the observations. Thus, we need a yardstick to regularise the latent space. We suggest employing latent actions in a way similar to how we used the actions to represent the state space above.

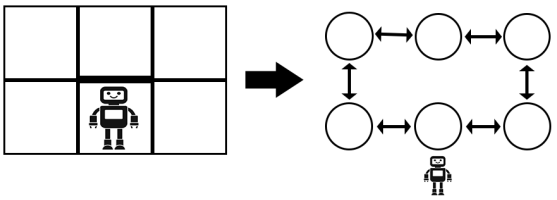

FIGURE 4.1: An example of a state space $\mathcal{S}$ (to the right) modelling the environment to the left, where the further the physical distance to a state (a circle), the further distance in the environment.

Suppose, we have a function $g : \mathcal{A} \to \mathcal{H}$ mapping actions $a \in \mathcal{A} \subset \mathbb{R}^m$ to their latent representations $h \in \mathcal{H} \subseteq \mathcal{Z}$. To keep the spatial structure of the latent space $\mathcal{Z} \subset \mathbb{R}^n$ close to the one of the state space $\mathcal{S}$, we additionally require that the angles between two action vectors a_i, a_j be preserved in the latent action space $\mathcal{H}$, which is possible if it is bigger or equal to the action space $\mathcal{A}$, i.e. $m < n$. For example, if we want to go left in the environment, we have to reserve a direction in the latent space, and this direction has to be orthogonal to the direction of up and down. Mathematically, it can be achieved by cosine similarity such as

$$\frac{a_i \cdot a_j}{||a_i|| \cdot ||a_j||} = \frac{h_i \cdot h_j}{||h_i|| \cdot ||h_j||} \,. \tag{4.4}$$

The function g can be learned using a VAE if the loss function is modified. We keep the original reconstruction error as in Section 2.3.2, but we propose a new regularisation term to keep the angles between a_i, a_j and h_i, h_j. Instead of KL-divergence, we minimise an absolute difference between cosine similarities. Thus, a loss function $\mathcal{L}_{\text{act}}(\phi)$ is

$$\mathcal{L}_{\text{act}}(\phi) = \arg\min_{\phi} \left(\mathbb{E}_{h \sim q_a(h)} \left[\left|\left| a - g_\phi(h) \right|\right|^2 \right] + \sum_{i,j \in \mathcal{A}} \left| \frac{a_i \cdot a_j}{||a_i|| \cdot ||a_j||} - \frac{h_i \cdot h_j}{||h_i|| \cdot ||h_j||} \right| \right) . \tag{4.5}$$

After training the VAE, if the dimensionality of $\mathcal{H}$ is equal to the dimensionality of $\mathcal{Z}$, we can use vectors h to preserve directions in the latent space $\mathcal{Z}$. As our objective is to preserve the original directions of the state space $\mathcal{S}$ in the latent space $\mathcal{Z}$, we train a second VAE with the loss function $\mathcal{L}(\theta)$ such that

$$\mathcal{L}(\theta) = \arg\min_{\theta} \left(\mathbb{E}_{z \sim q_x(z), h \sim q_a(h)} \left[\left|\left| x - f_\theta(z) \right|\right|^2 + \left|\left| z_t - (z_{t-1} + h_{t-1}) \right|\right|^2 \right] \right) , \tag{4.6}$$

where $h \sim g_\phi(a)$ is trained with $\mathcal{L}_{\text{act}}(\phi)$. The regularisation term in Equation 4.6 guaranties that the next latent state $z_t \sim f_\theta(s_t)$ obtained after applying an action $h_{t-1} \sim g_\phi(a_{t-1})$ is in the same direction as the original state s_t.

Later, in Section 4.5, we test the observation space with the action regularisation in the Kuka and Reacher environments, which we define in following Section 4.4.

4.4 Framework Formulation

We extend the Kuka environment in Section 4.4.1, and, additionally, we introduce a new Reacher environment in Section 4.4.2.

4.4.1 Kuka

First, we extend a Kuka manipulator framework, defined in Section 3.2, to accommodate visual observations instead of hand-crafted feature vectors, used in Chapter 3. We keep the action space $\mathcal{A}$ and the state space $\mathcal{S}$ the same as in Section 3.2. However, as a reward function R, we use the state-similarity reward, discussed in Section 4.2.

Additionally, we introduce a new task in which the agent has to achieve the goal state that is given as a set of images. All other aspects of the environment are kept the same as in Section 3.2.

4.4.1.1 Observation Space

Further, we define a new observation space Ω accommodating visual observations. In contrast to common practice, the Kuka environment generates two images each time step t, simulating two cameras. The virtual cameras are located perpendicular to each other facing at $x-z$ and $y-z$ planes. An example of a visual observation can be found in Figure 4.2. Each image has the height and width of 128 pixels and 3 colour channels. Before mapping them to a latent space $\mathcal{Z}$, we normalise each pixel to have intensity between 0 and 1 and combine two images together by their colour channels. That is, each visual observation o has 128 x 128 x 6 dimensions, i.e. $\mathbb{R}^{128\times128\times6}$.

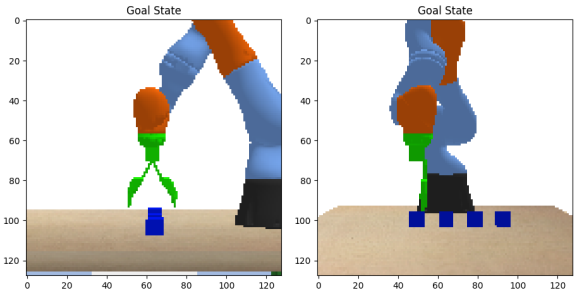

FIGURE 4.2: An example of a visual observation obtained from two cameras. Each image has the height and width of 128 pixels and 3 colour channels.

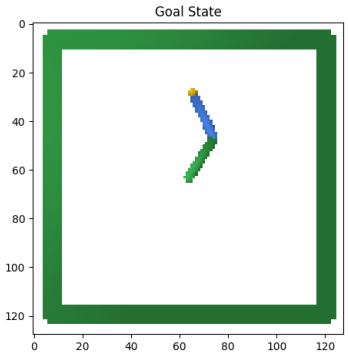

FIGURE 4.3: An example of a visual observation for the Reacher environment. Each image has the height and width of 128 pixels and 3 colour channels.

Finally, the observations are mapped into the latent space $\mathcal{Z} \subset \mathbb{R}^{256}$, from which feature vectors $z \in \mathcal{Z}$ are drawn. We sample them from a distribution $p(z|o)$ learned using a VAE with either the vanilla or action regularisation. In our experiment, we used the vanilla regularisation as a baseline.

To show that the same algorithm can be applied to a variety of problems, we define the second environment in following Section 4.4.2. Additionally, we would like to note that although we assume that all images have a constant size, the images of different sizes can be easily resized without loss of generality.

4.4.2 Reacher

We modified Reacher environment [9], an instance-invariant, continuous control problem. Originally, the agent's goal was to reach a ball, controlling the angles of two joints. However, we introduce a new task in which the agents goal is to reach a configuration that is given as an image. An example of a goal state can be found in Figure 4.3.

In this environment, the manipulator, comprised of two joints, has two degrees of freedom. The configuration space is instance-independent and contains the angles of both of the joints α, β. Thus, the configuration space is a Cartesian product of all possible configurations of

$$\chi = \alpha \times \beta \in \mathbb{R}^2 \,. \tag{4.7}$$

Additionally, there are no friction nor inertia in the environment.

We frame the environment on a POMDP with a tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O)$, where an initial state $s_{\text{init}} \in \mathcal{S}$ and a goal state $s_{\text{goal}} \in \mathcal{S}$ are assigned on each trial. The state space $\mathcal{S}$ is instance-invariant and equal to the manipulator's configuration space. Similar to the Kuka environment, we use a state-similarity reward as a reward function R. Also, time is discretised in the same manner as in the Kuka environment.

The action space $\mathcal{A}$ contains one action $a = \omega_1 \times \omega_2 \in \mathbb{R}^2$, where the agent can apply the angular velocities (shifts), $\omega_i \in \Omega_i \subset \mathbb{R}$, for both of the joints $i \in 1, 2$. Then, the next time step $t + 1$, the manipulator will move to the state with the configuration $\chi_{t+1} = (\alpha_t + \omega_1) \times (\beta_t + \omega_2)$. Thus, the action space $\mathcal{A}$ is a union of all possible actions such that $\mathcal{A} = \{\omega_1 \times \omega_2 \,|\, \forall\, \omega_1, \omega_2 \in \mathbb{R}\}$.

Concerning the observation space Ω, our goal is to show that the same algorithm can be applied to a variety of problem. Thus, we use the same technique as in the Kuka environment (Section 4.4.1.1) to obtain feature vectors from visual observations. The only notable difference is that the Reacher environment generates a single image for each time step. That is, each visual observation has the height and width of 128 pixels and 3 colour channels, i.e. $\mathbb{R}^{128\times128\times3}$. All other aspects of the environment are kept the same as in Section 2.1.

In following Section 4.5, we used these environments to examine the performance of the hand-crafted and the state-similarity reward functions in the Kuka and Reacher environments.

4.5 Experimental Results

Our goal was to learn policies that can reach a specified goal state when the state-similarity reward and visual observations were used. More broadly, our goal was to

show that the same problem setup was general enough to be applied to a variety of problems. Bearing that in mind, we tested our algorithm in the Kuka and Reacher environments.

We chose a new task in which the agent had to reach a goal state, given as a set of images, which was generated at the beginning of each trial. We used these images in the state-similarity reward function $R(o_t, o_g)$, defined in Section 4.2. However, because we utilised the same off-the-shelf PPO implementation as in Section 3.5.1, which cannot learn a goal-conditioned policy, only one goal state was used on each trial. For PPO, we kept the hyperparameters the same as in Section 3.5.1.

As a baseline, we used pairs of:

1. The latent space with the vanilla regularisation and the hand-crafted reward function $R(s_t, s_g)$. We labelled it as *vanilla, hand-crafted*.
2. The latent space with the vanilla regularisation and the state-similarity reward function $R(o_t, o_g)$. We labelled it as *vanilla, state-similarity*.

All the latent spaces had 256 dimensions, i.e. $\mathcal{Z} \subset \mathbb{R}^{256}$. The hand-crafted reward $R(s_t, s_g)$ was defined as a distance to the goal in the state space $\mathcal{S}$ (Equation 4.1). For this purpose, we used the same hand-crafted state space (no blocks in observations) as it was in Section 3.3.2.1 because it is instance-invariant. Thus, mathematically, the state space was a Cartesian product of all possible configurations of

$$X_{\text{goal}} \times \chi_{\text{gr}} \in \mathbb{R}^{3+7}\,.$$

We used two parameters to evaluate each policy over 200 trials:

- **Initial distance** – the mean distance from the initial states to the goal state, which was measured in the latent space. We defined it as a ball with a **radius** r, which we varied in the experiments.
- **Final distance** – the mean final distance to the goal state, which was measured in the latent space as well. Because we organised the latent space in a way the neighbouring states in the state space are close in the latent space as well, the smaller the distance, the better policy convergence to the goal.

All the policies were trained on a `c5.9xlarge`[1] cloud instance with 36 vCPUs from AWS Cloud Computing Services. To train the VAEs with both the vanilla and action regularisation, we used a `p3.2xlarge`[2] cloud instance with one Tesla V100 GPU and 8 vCPUs. All the policies were tested on MacBook Pro's Intel Core i5 that had 4 cores and 8 threads.

4.5.1 Performance Evaluation

In this section, we present empirical results comparing the performance of the state-similarity reward function with action-regularised latent space (*action, state-similarity*) to the baseline in the Kuka and Reacher environments. As the baseline, we used the latent space with the vanilla regularisation with either the hand-crafted reward (*vanilla, hand-crafted*) or the state-similarity reward (*vanilla, state-similarity*).

First, we visually studied the VAE's reconstruction errors, shown in Figure 4.4. That is, we encoded visual observations into the latent space and decoded them back. For the Kuka environment, the VAEs were trained by varying blocks' coordinates and the number of blocks. In both of the environments, we observed that the VAE with the action regularisation reconstructed cleaner images, which can be seen in Figure 4.4. We attribute this to the fact that the action regularised latent space had a better organisation, preserving more information.

Next, we examined the structure of our latent spaces. For this purpose, their dimensionalities were reduced from 256 to 3, which was achieved by using an off-the-shelf t-SNE algorithm [52]. To better understand the spatial structure, we chose the same initial state, from which all possible unit actions were taken five times in succession. For example, in the Reacher environment, the action space had two dimensions, where the actions represented the velocities of the joints. Thus, we consequently applied one of the actions $(1, 0), (-1, 0), (0, 1), (0, -1)$ five times. The same steps were taken in the Kuka environment, but there were four possible unit actions instead. The results can be found in Figure 4.5, where we additionally colour-coded the initial state and each distinct unit action.

[1] `https://aws.amazon.com/ec2/instance-types/c5/`
[2] `https://aws.amazon.com/ec2/instance-types/p3/`

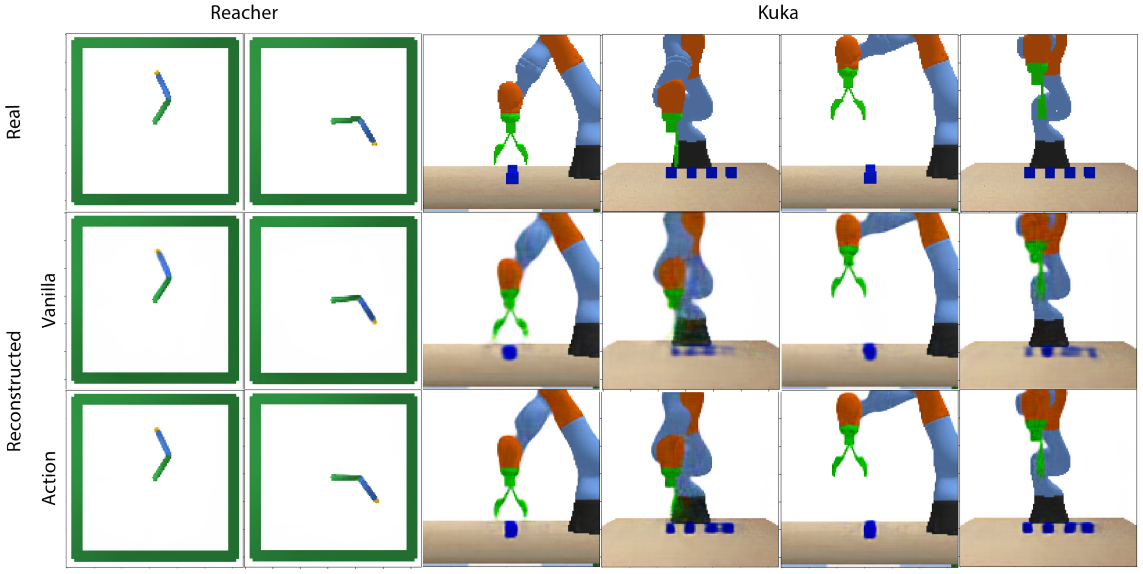

FIGURE 4.4: A visual representation of the VAE's reconstruction errors. The first row demonstrates the original observations. The second row shows the observations decoded from the latent space of the vanilla VAE. The third row demonstrates the observations decoded from the latent space of the action-regularised VAE. For Reacher's latent space, the manipulator's position may vary. For Kuka's latent space, the number of blue blocks and their coordinated, as well as the gripper's position, may vary.

We observed that, for the action regularisation, the consecutive actions were located closer to each other and formed straight lines, which is desirable behaviour, required for the state-similarity reward function. On the other hand, for the vanilla regularisation, the actions were spanned across the entire latent space with several states between them.

Next, we learned policies using both of the latent states. We chose a task in which the agent had to reach a goal state, given as a set of images. The cumulative rewards can be found in Figure 4.6, along with their training times in Table 4.1. However, because the rewards were auto-generated from different latent spaces, they are not on the same scale; thus, they cannot be properly compared. Also, we could not learn a working policy for *vanilla, state-similarity* because the vanilla latent space could not be used in the state-similarity reward as it was discussed in Section 4.2. Therefore, the policy *vanilla, state-similarity* was excluded from all further tests.

Finally, we analysed the policy performance by measuring the policy convergence to the goal state, measured in the latent space $\mathcal{Z}$. Primary, we compared the policy behaviours varying the initial distance to the goal state in the state space $\mathcal{S}$. In the Kuka environment, it was achieved by moving the gripper a distance of r-blocks along an arbitrary

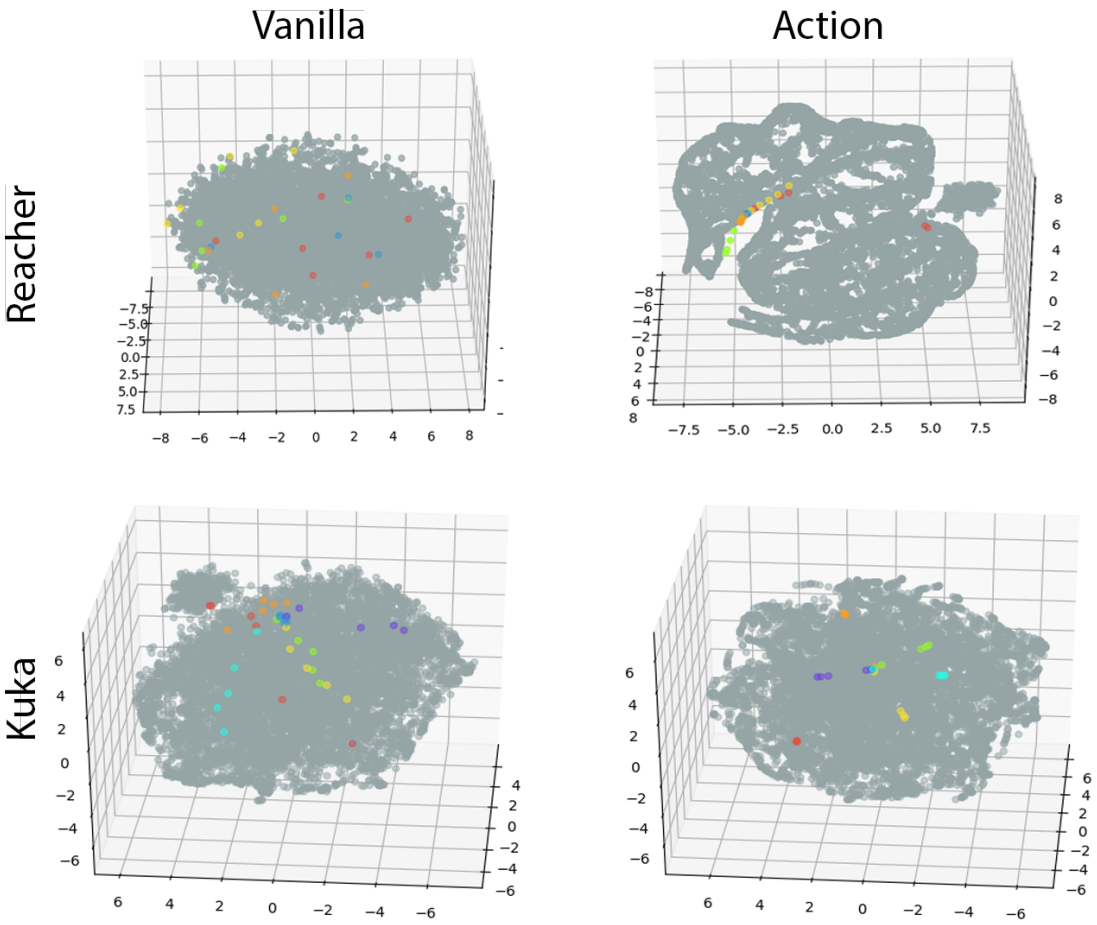

FIGURE 4.5: A visual representation of the vanilla and action regularised latent spaces. The spaces are reduced from 256 dimensions by t-SNE, where the colours represent distinct unit actions. The blue dots are the same initial state.

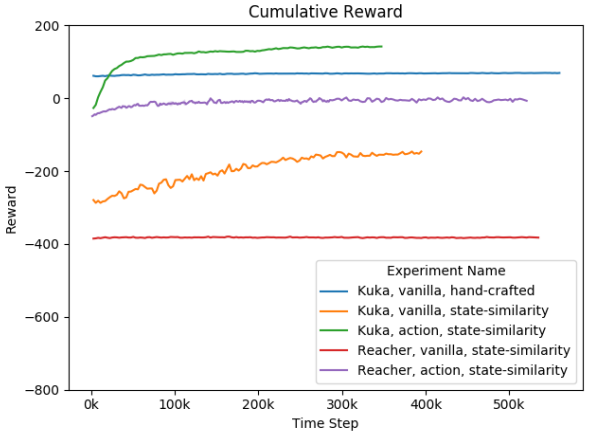

FIGURE 4.6: The cumulative reward for the Kuka and Reacher environments

Exp Name	Training Time
Kuka	
vanilla, hand-crafted	1h 25m 44s
vanilla, state-similarity	1h 24m 52s
action, state-similarity	1h 10m 19s
Reacher	
vanilla, state-similarity	1h 23m 06s
action, state-similarity	1h 25m 57s

TABLE 4.1: Training times.

Initial Distance	Initial Distance	Final Distance
	Kuka	
Vanilla,	**hand-crafted**	
1 bl	5.4433 ± 0.0087	13.9479 ± 0.194
2 bl	9.3228 ± 0.0231	13.8247 ± 0.1894
4 bl	11.9968 ± 0.0482	13.9264 ± 0.2001
5 bl	10.8223 ± 0.0755	13.7759 ± 0.186
Action,	**state-similarity**	
1 bl	2.2731 ± 0.0076	1.0240 ± 0.0521
2 bl	3.8930 ± 0.0106	1.0638 ± 0.0536
4 bl	5.4525 ± 0.0169	1.1039 ± 0.0495
5 bl	5.5911 ± 0.0593	1.1809 ± 0.0602
	Reacher	
Action,	**state-similarity**	
30°	3.1514 ± 0.0424	0.1612 ± 0.0091
80°	3.4921 ± 0.0560	0.4325 ± 0.1119
100°	3.4881 ± 0.0321	1.3341 ± 0.1736
120°	3.2542 ± 0.0616	1.7583 ± 0.1657
160°	3.4453 ± 0.0605	2.3935 ± 0.1044

TABLE 4.2: The policy performance. In the first column, the mean initial distance is measured in blocks for Kuka and in angles for Reacher. In the last two columns, the distances are measured in the latent space.

direction. To have some intuition, we chose to measure the distance in blocks because there is no unit of length in the Kuka environment.

We varied the initial distance r from one to five blocks, where the radius of five blocks represented the physical boundaries of the environment. The results are presented in Table 4.2. Increasing the distance r in the state space, we observed a consistent growth of the initial distances, measured in the latent space. However, in contrast to the *vanilla, hand-crafted*, the *action, sate-similarity* policy converged to the goal state. This difference in final distance can be seen in Figure 4.7, where we presented two examples of policy execution for the ball radius of four blocks.

In the Reacher environment, we varied the angle of the manipulator's bottom joint. More precisely, we varied the difference in the angles between the goal state and the initial state. In this case, the environment had a physical boundary of 180 degrees. In this task, our goal was to show that the same technique could be applied to a variety of problems. Thus, we learned policies only for the *vanilla, state-similarity* and the *action, state-similarity*, but the former policy did not converge; thus, it was excluded. The results are presented at the bottom of Table 4.2.

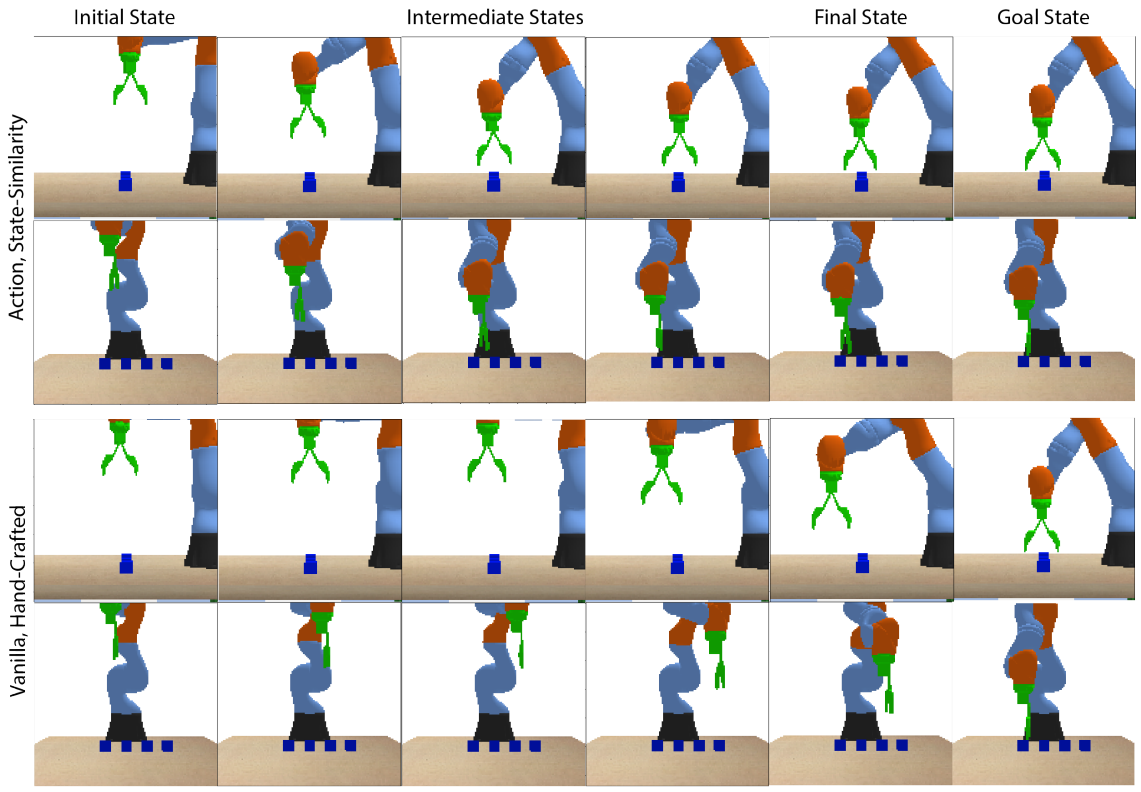

FIGURE 4.7: An example of policy execution for the latent space with the action regularisation and the state-similarity reward function (at the top), and the latent space with the vanilla regularisation and the hand-crafted reward function (at the bottom). The final state is the last state of the trial. The policy were executed for the ball radius of four blocks.

In this test, we observed that the *action, state-similarity* policy reached the goal state only when the difference in the angle of the first joint between the initial state and the goal state was less than 90°. This could be explained with the fact that each move to the goal decreased the latent distance; thus, the reward was positive. However, for the ball radius greater than 90°, initial moves decreased the reward, which led the agent to the local maximum. It was the case because the closest path was through the centre of the board, which is an illegal action and violates our assumption that the neighbouring states must be reachable with a single action. It is also reflected in the initial distance that stopped growing after the ball radius of 90°. We further illustrate this idea in Figure 4.8.

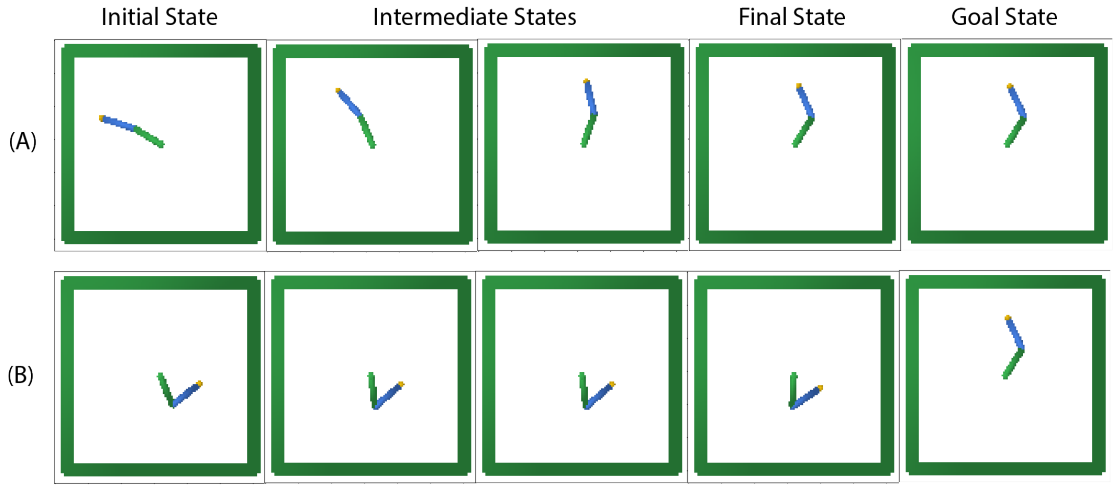

FIGURE 4.8: An example of policy execution for the latent space with the action regularisation and the state-similarity reward function. (A) - the angle of the bottom joint differs less that 90° from the goal state. (B) - the angle of the bottom joint differs more that 90° from the goal state.

4.5.2 Summary

It can be expedient to auto-generate rewards for a variety of problems measuring the distance between visual observations. However, in this case, the observations should be compressed into their latent representations reflecting the structure of the state space. For this purpose, a VAE can be used if its latent space is properly regularised.

We showed that the VAE's vanilla regularisation does not hold the triangle inequality; thus, it cannot be used in the state-similarity reward. Therefore, we could not learn a working policy using this latent space. However, it was possible to learn a policy if the action regularisation was used. Additionally, we showed that the same technique could be applied to the Reacher environment.

However, our method has a limitation. The environment must not violate the assumption that the neighbouring states in the latent space are reachable with a single action. Otherwise, the policy can converge to a local maximum.

Chapter 5

Conclusion

In this chapter, we summarise the contribution we made in this thesis. Additionally, we outline possible future work.

5.1 Summary of the Thesis

Picking and placing arbitrary objects in open, unknown environments or under uncontrolled conditions is still an open challenge in robotics. Although there is significant progress in learning end-to-end policies in continuous control tasks using Reinforcement Learning, the research community usually focuses on highly constrained, isolated tasks in a controlled environment. Thus, their policies cannot be readily transferred to another, although similar, problem or to another environment. However, it is also important to focus on the robot's ability to handle a variety of tasks or operate in a range of situations, i.e. to focus on generalisation.

In this thesis, we focus on two research questions, which we addressed in two chapters.

In Chapter 3, we aimed to explicitly construct an observation space that can be used for learning policies when the number of elements in the state space can vary. Particularly, our goal was to learn end-to-end policies for picking and placing objects when the number of objects can vary. We were interested in policies that can work with the number of objects that are not used in training. More broadly, we were interested in examining the connection between the policy's generalisation ability and its observation space.

We showed that it is useful to frame the environment on a POMDP instead of an MDP when the state space of a real-world environment can be unknown or instance-dependent. In this case, it allowed us to devise a more general instance-independent set of observations that were used in a broad range of tasks.

We demonstrated that the policy trained with the sensing sphere as observation space had better results in the *Pick* and *Place* tasks. At the same time, the policy trained with the space constructed by excluding all the surrounding blocks showed marginal results in generalising to the towers of different heights.

In Chapter 4, we aimed to generate the latent space from visual observations using a VAE. We were interested in the latent space that reflected the spatial structure of the hidden state space, from which visual observations were taken. In contrast to current practice, we used visual observations taken from two cameras instead of one. The cameras were placed perpendicular to each other.

Additionally, we used the latent space to construct a goal-conditioned reward function. Particularly, we focused on learning polices using visual observations and our goal-conditioned reward. More broadly, our goal was to show that the same problem setup is general enough to be applied to a variety of problems.

We showed that it is possible to auto-generate rewards for a variety of problems measuring the distance between visual observations in the latent space. However, in this case, the latent space should have the structure reflecting the spatial structure of the state space to be used in the state-similarity reward.

We showed that the VAE's vanilla regularisation does not hold the triangle inequality; thus, it cannot be used in the state-similarity reward. However, it was possible to learn a policy if the action regularisation was used. Additionally, we showed that the same technique could be applied to the Reacher environment.

5.2 Future Work

In our study, we addressed several problems focusing on constructing hand-crafted features and latent spaces from visual observations. However, there are still many problems to be solved.

In Chapter 3, our approach had several limitations. Due to the nature of the observation space, the sensing sphere cannot be easily transferred to another problem. Also, it does not fully capture the geometry to the problem. Therefore, a future line of work can be to design a better observation space.

Another line of work can be the learning algorithm. The vanilla PPO does not have a goal-conditioned policy. Thus, we had to include the goal state into the observation. It imposed a layer of complexity because PPO had to disentangle the goal state from the observations. It would be beneficial to study the performance of the proposed observation spaces while using goal-conditioned policies as in [12], [20].

In Chapter 4, our method also had a limitation. Constructing the action regularised latent space, we assumed that the neighbouring states in the latent space were reachable with a single action. This assumption was violated in the Reacher environment, which led the policy to converge to a local maximum on some trials. Therefore, a possible future work can be to search for a better regularisation, which can work in the Reacher environment.

Additionally, the properties of the regularised latent space could be studied further. For example, an estimate of the degree by which the triangle inequality is preserved is still an open question.

Finally, as a possible line of work, the proposed learning framework can be tested on a real robot using real cameras.

As a final word, we like to stress that this thesis focused on problems of the robot's ability to generalise. We hope our work can provide a starting point for researchers interested in this area and motivate them to search further.

Bibliography

[1] Lydia Kavraki, Petr Svestka, Jean Latombe, and Mark Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Tech. rep. Stanford, CA, USA, 1994.

[2] Roland Geraerts and Mark H. Overmars. "A comparative study of probabilistic roadmap planners". In: *Algorithmic Foundations of Robotics V*. Ed. by Jean-Daniel Boissonnat, Joel Burdick, Ken Goldberg, and Seth Hutchinson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 43–57. DOI: 10.1007/978-3-540-45058-0_4.

[3] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761.

[4] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Chicago, IL, USA, 2014, pp. 2997–3004. DOI: 10.1109/IROS.2014.6942976.

[5] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. "Learning to walk via deep reinforcement learning". In: *arXiv preprint* (2018). eprint: arXiv:1812.11103.

[6] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng,

and Wojciech Zaremba. "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20. DOI: 10.1177/0278364919887447.

[7] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. "Learning latent plans from play". In: ed. by Leslie Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. Cambridge, MA, USA: PMLR, 2020, pp. 1113–1132. URL: http://proceedings.mlr.press/v100/lynch20a.html.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "Openai gym". In: *arXiv preprint* (2016). eprint: arXiv:1606.01540.

[10] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. Vol. 48. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1928–1937. DOI: 10.5555/3045390.3045594.

[11] Matthew Hausknecht and Peter Stone. "Deep recurrent Q-learning for partially observable MDPs". In: *arXiv preprint* (2015). eprint: arXiv:1507.06527.

[12] Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. "Planning with goal-conditioned policies". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vancouver, Canada: Curran Associates, Inc., 2019, pp. 14843–14854. URL: http://papers.nips.cc/paper/9623-planning-with-goal-conditioned-policies.pdf.

[13] Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. "Action schema networks: Generalised policies with deep learning". In: *AAAI Conference on Artificial Intelligence (AAAI)*. New Orleans, LA, USA, 2018.

[14] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. "Deep Imitation learning for complex manipulation tasks from virtual reality teleoperation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 5628–5635. DOI: 10.1109/ICRA.2018.8461249.

[15] Dylan P. Losey, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, and Dorsa Sadigh. "Controlling assistive robots with learned latent actions". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 378–384. DOI: 10.1109/ICRA40945.2020.9197197.

[16] Debidatta Dwibedi, Jonathan Tompson, Corey Lynch, and Pierrev Sermanet. "Learning actionable representations from visual observations". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain, 2018, pp. 1577–1584. DOI: 10.1109/IROS.2018.8593951.

[17] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. "Learning spatiotemporal features with 3D convolutional networks". In: *IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4489–4497. DOI: 10.1109/ICCV.2015.510.

[18] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Universal planning networks". In: *arXiv preprint* (2018). eprint: arXiv:1804.00645.

[19] Yibing Zhan, Jun Yu, Zhou Yu, Rong Zhang, Dacheng Tao, and Qi Tian. "Comprehensive Distance-Preserving Autoencoders for Cross-Modal Retrieval". In: *Proceedings of the 26th ACM International Conference on Multimedia*. MM '18. Seoul, Republic of Korea: Association for Computing Machinery, 2018, pp. 1137–1145. ISBN: 9781450356657. DOI: 10.1145/3240508.3240607.

[20] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. "Learning actionable representations with goal-conditioned policies". In: *arXiv preprint* (2018). eprint: arXiv:1811.07819.

[21] Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. "Skew-fit: State-covering self-supervised reinforcement learning". In: *arXiv preprint* (2019). eprint: arXiv:1903.03698.

[22] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. "Visual reinforcement learning with imagined goals". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 9209–9220. DOI: 10.5555/3327546.3327593.

[23] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning latent dynamics for planning from pixels". In: ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, CA, USA: PMLR, 2019, pp. 2555–2565. URL: http://proceedings.mlr.press/v97/hafner19a.html.

[24] Arthur George Richards. "Robust constrained model predictive control". PhD thesis. Cambridge, MA, USA: Massachusetts Institute of Technology, 2005.

[25] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1 (1998), pp. 99–134. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(98)00023-X.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. Second. Adaptive Computation and Machine Learning. Cambridge, MA: A Bradford Book, 2018, p. 54.

[27] Christopher Watkins. "Learning from delayed rewards". PhD thesis. Cambridge, UK: University of Cambridge, 1989.

[28] Hado van Hasselt. "Double Q-learning". In: *Proceedings of the 23rd International Conference on Neural Information Processing Systems*. Vol. 2. NIPS'10. Vancouver, Canada: Curran Associates Inc., 2010, pp. 2613–2621. DOI: 10.5555/2997046.2997187.

[29] Hado van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double Q-learning". In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, AZ, USA: AAAI Press, 2016, pp. 2094–2100.

[30] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando de Freitas. "Dueling network architectures for deep reinforcement learning". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. Vol. 48. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1995–2003. DOI: 10.5555/3045390.3045601.

[31] Ronald Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8 (1992), pp. 229–256. DOI: 10.1007/bf00992696.

[32] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS'99. Denver, CO: MIT Press, 1999, pp. 1057–1063. DOI: 10.5555/3009657.3009806.

[33] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic policy gradient algorithms". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning*. Vol. 32. ICML'14. Beijing, China: JMLR.org, 2014, pp. 387–395. DOI: 10.5555/3044805.3044850.

[34] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. "Distributed distributional deterministic policy gradients". In: *arXiv preprint* (2018). eprint: arXiv:1804.08617.

[35] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. "Sample efficient actor-critic with experience replay". In: *arXiv preprint* (2016). eprint: arXiv:1611.01224.

[36] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". In: NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 5285–5294. ISBN: 9781510860964. DOI: 10.5555/3295222.3295280.

[37] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *arXiv preprint* (2018). eprint: arXiv:1801.01290.

[38] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholm, Sweden: PMLR, 2018, pp. 1582–1591. URL: http://proceedings.mlr.press/v80/fujimoto18a.html.

[39] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint* (2015). eprint: arXiv:1509.02971.

[40] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. "Trust region policy optimization". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. Vol. 37. ICML'15. Lille, France: JMLR.org, 2015, pp. 1889–1897. DOI: 10.5555/3045118.3045319.

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint* (2017). eprint: arXiv:1707.06347.

[42] Dana H. Ballard. "Modular learning in neural networks". In: *Proceedings of the Sixth National Conference on Artificial Intelligence*. Vol. 1. AAAI'87. Seattle, WA, USA: AAAI Press, 1987, pp. 279–284. ISBN: 0934613427.

[43] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[44] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems*. Vol. 2. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680. DOI: 10.5555/2969033.2969125.

[45] Mark A. Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE Journal* 37.2 (1991), pp. 233–243. ISSN: 0001-1541. DOI: 10.1002/aic.690370209.

[46] Alireza Makhzani and Brendan Frey. "K-sparse autoencoders". In: *arXiv preprint* (2013). eprint: arXiv:1312.5663.

[47] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1096–1103. ISBN: 9781605582054. DOI: 10.1145/1390156.1390294.

[48] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. "Contractive auto-encoders: Explicit invariance during feature extraction". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 833–840. DOI: 10.5555/3104482.3104587.

[49] Diederik P. Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint* (2013). eprint: arXiv:1312.6114.

[50] KUKA Roboter GmbH. *KUKA LBR iiwa, User's manual*. KUKA Roboter GmbH. 2017.

[51] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. "RLlib: Abstractions for distributed reinforcement learning". In: *International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholm, Sweden: PMLR, 2018, pp. 3053–3062. URL: http://proceedings.mlr.press/v80/liang18b.html.

[52] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: https://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.