School of Mechanical and Civil Engineering

A Deep-Reinforcement-Learning Approach to the Peg-in-Hole Task with Goal Uncertainties

Thibault Rouillard

0000 – 0002 – 2465 – 7678

This thesis is presented for the Degree of

Doctor of Philosophy

of

Curtin University

December 2020

Abstract

Fine robotic manipulation skills have countless applications both in industrial settings and uncontrolled human environments. Traditional methods to enable these skills are labour intensive, non-transferable and based on well-known models of both the manipulators and their environments. Emergent approaches to fine manipulation such as deep-reinforcement learning (DRL) shifts away from the constraints of traditional methods by offering model-free solutions.

DRL of fine manipulation tasks faces the following three limitations that the literature has yet to overcome: First, the formalism lacks consistency for its applications to fine manipulation. Second, state-of-the-art uses of the formalism all rely on perfect knowledge of a task's goal, which is unrealistic for real-world uses and leads to poor performance. Third, solutions to DRL problems do not stem from one common root but instead are fragmented and isolated in the literature and have no benchmark. DRL needs a common framework to apply to fine manipulation tasks with goal uncertainties.

This thesis aimed to create a framework and to apply it to fine manipulation tasks. The work proposed three aspects of a framework for formulating and training an agent to solve DRL fine manipulation tasks with goal uncertainties. The first aspect proposes to divide the state-space features of agents into three categories: goal features, constraint features and control features. A methodology

was developed to extract those features in fine manipulation tasks. Second, a method was developed to make a DRL agent more robust to goal uncertainties. It consisted of artificially inducing goal uncertainties during agent training to encourage exploration. Finally, the framework included a methodology to train DRL agents. It consisted of progressively scaling environment difficulty based on agent's success rate. It offered parameters to frame the methodology.

The proposed framework was then applied step by step to two different Peg-in-Hole manipulation tasks. Agents progressively learned to complete the task via increasingly small peg/hole clearances. The agent trained with three types of goal uncertainties (uniform,bi-modal, Gaussian) for a round PiH task (task 1) and one type of goal uncertainty (Gaussian) for a square PiH task (task 2) that affected the perceived hole position. In simulation, the agents trained with uncertain goals performed consistently better than those trained with definite goals. The behaviours were then imported to a physical robotic manipulator.

In the simulation of task 1, the agent trained with definite goals had insertion rates at least 18%, 30% and 60% smaller for environments with uniform, Gaussian and bi-modal uncertainties respectively. This translated to insertion rates 14%, 29% and 14% smaller on the physical manipulator hence confirming the validity of the proposed approach for task 1.

Furthermore, in the simulation of task 2, the agent trained with definite goals had insertion rates at least 12% smaller for environments with Gaussian uncertainties

which lead to an insertion rate 29% smaller on the physical manipulator. This result confirms the validity of the proposed approach for more than one manipulation task.

In summary, the work in this thesis proposed a framework to train deep reinforcement learning agents for fine manipulation tasks with goal uncertainties. The framework was used train two agents, one for a round PiH and one for a square PiH task. The agent behaviours were validated in a simulation and on a physical manipulator for both tasks. They showed that the agents trained using the framework performed better in every single case compared to agents trained without goal uncertainties. The framework can hence facilitate the application of DRL to similar fine manipulation task.

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgement has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature:

Date: September 9, 2021

Acknowledgement

I want to express my gratitude to my thesis supervisor, Dr. Lei Cui, for guiding my work and providing continuous feedback through my PhD study. I am immensely grateful to Prof. Ian Howard, my co-supervisor, for his constant support and encouragements during difficult times and for the constructive advice he has provided along the way.

I am first and foremost grateful to my partner, Marine, who supported me through all the ups and downs of this journey. She gave me the strength to keep going at every step of the way. To my parents, who encouraged me to undertake this work and are responsible for all of my successes. To my sister, brother and sister-in-law for their continued interest and support of my research.

I want to thank my colleagues, Gino Parisella, Ngoc Tam Lam, Zefang Shen and Pratik Pandya with whom I shared this experience and who provided helpful advice and a sense of belonging.

I am grateful for my friends who shared my joys and sorrows and who carried me through the finish line.

Finally, I would like to thank the Department of Mechanical Engineering for the CIPRS scholarship that funded my PhD work.

Contents

List of Figures

List of Tables

List of Symbols

S	Set of all states of an agent
A	Set of all actions of an agent
R	Reward function of an MDP or RL process
T	State transitions probability matrix in an
s_t	State at step t
a_t	Action at step t
r_t	Scalar reward at time
F_x	Force on the manipulators wrist in the x axis
F_y	Force on the manipulators wrist in the y axis
F_y	Force on the manipulators wrist in the z axis
$\mathcal{T}_x$	Torque on the manipulators wrist around the x axis
$\mathcal{T}_y$	Torque on the manipulators wrist around the y axis
$\mathcal{T}_z$	Torque on the manipulators wrist around the z axis
$p(X)$	Probability of X
H	RL episode return
π	RL policy
$\mathbb{E}$	Probabilistic expectation
Q	RL value function
∇J	Gradient of the objective function

y_i	Critic network targets for the i^{th} transition
L	Network Loss
θ^Q	Weights of the Critic Network
θ^μ	Weights of the Actor Network
$\mathcal{N}$	Normal probability density function
G	Set of all goals of an agent
G_{train}	Set of all goals of an agent in a training environment
G_{train}	Set of all goals of an agent in a training environment with goal uncertainties
G_{env}	Set of all goals of an agent in a validation environment with
g	Goal of an agent
$\mathcal{G}_{train}$	Distribution the goals in a training environment
$\mathcal{G}_{train}$	Distribution the goals in a training environment with goal uncertainties
$\mathcal{G}_{env}$	Distribution the goals in a validation environment
τ	Network update parameter
$\emptyset$	Empty Set
$\cap$	Intersection of two sets
x	Peg's position in the x axis
y	Peg's position in the y axis
z	Peg's position in the z axis
θ	Peg's orientation in yaw

Chapter 1: Introduction

1.1 Motivation

Manipulation is a significant aspect of robotics that holds incredible potential for the future. Robots with manipulation skills can alter their environments, making them ideal tools for a range of applications. Nonetheless, human labour must still take part in dirty, dangerous and demeaning tasks because only they possess the fine manipulation skills that robots still lack.

Fine manipulation is the skilful handling of a tool to accomplish a task that requires precise motions with small error margins. For robotic manipulators, these physical interactions are complex to model and vary from one task to another and from one manipulator to another. The literature to solve these tasks is abandoning model-based solutions for the benefit of self-learning and model-free methods. Deep reinforcement learning, is a formalism that offers an alternative to robotic manipulators to learn a behaviour on their own through repeated interactions. It is an emergent approach in the literature that still lacks rigour, benchmarks and usability.

There already exist conventional methods to solve fine manipulation tasks, such as impedance and force control. These methods are manipulator and environment

specific, and require calibrated apparatus. DRL, on the other hand, provides the potential to solve these same tasks, for any manipulator and environment based on the agent's ability to adapt and learn. Such premises is the reason why this work explores further the challenges of applying DRL to fine manipulation tasks.

Although model-based solutions offer concrete predictions on the effects of uncertainties on performance, this is not yet the case for DRL approaches to fine manipulations tasks. Uncertainties affect all manipulation problems whether they stem from the environment, sensing, actuation, modelling or computation. Fine manipulation tasks are most affected by uncertainties first due to their high precision and second because DRL agents can only learn them in simulation while having to perform in the real world. Allowing DRL agents to learn on real hardware is impractical and unsafe. There are no solutions to mitigate the effects of uncertainties on fine manipulation tasks for end-to-end DRL.

Fine manipulation tasks have a specific aim that can translate into a DRL algorithm goal features. They are an essential part of a DRL algorithm as they are the only way to assess and reward the behaviour of the agent. A goal is composed of features from the manipulator, and its environment which are affected by uncertainties. The definite values of the goal features cannot be used to learn manipulation tasks as sensors can never estimate them with the same accuracy in the real world. A solution that eliminates the adverse effects of goal uncertainties eliminates the need to estimate goals in the real world accurately and makes DRL more usable for fine manipulation.

This work develops a solution to tackle the issue of goal uncertainty for fine manipulation tasks via DRL. The work presents a framework that: strengthens the benchmarking of fine manipulation tasks; create agents more robust to goal uncertainties; proposes a progressive training approach to fine manipulation tasks.

1.2 **Aims and Objectives**

This work focuses on developing a framework to immunise DRL agents against goal uncertainties. The framework is applied to two fine manipulation task: round and square Peg-in-Hole. It is validated both in simulation and on a physical robotic manipulator.

The objectives of the work are to:

a) propose a design methodology for fine manipulation state-space models.
b) develop an approach that relies on these models and generates DRL agents that are more robust to goal uncertainty.
c) implement a training methodology that allows these agents to learn fine manipulation tasks in simulation and show that it improves performance on real hardware for a Peg-in-hole task.

1.3 **Thesis Outline**

The thesis is organised in 7 chapters. The structure is as follows:

Chapter 1 is an introduction to the thesis. It states the motivation, background and objectives of the thesis. It additionally describes its content.

Chapter 2 is an introduction to the mathematical tools used in the thesis. It first describes the aspects of reinforcement learning (RL) as a formalism before diving deeper into the algorithms that can solve RL problems. It explains how to compute the state-value functions in traditional RL and why this work employs a temporal difference learning approach. Finally, it presents the main algorithm used in the thesis, notably: deep deterministic policy gradient and hindsight experience relay.

Chapter 3 shows how the work in the thesis fits within the literature by providing a detailed review of related works and highlighting shortcomings of current approaches. First, it shows what approaches mitigate uncertainties in manipulation tasks, and focuses on the works that are then demonstrated on the Peg-in-Hole task. Then it explains how DRL/RL has been used for robotic manipulation while again showing when it was applied to the PiH task. Finally, the chapter offers a comparative perspective on how uncertainties have been addressed in DRL both implicitly and explicitly.

Chapter 4 introduces a framework for progressively training DRL agents for fine manipulation tasks with goal uncertainties. The three main aspects of the framework consist of: a methodology to extract state-space features from the robot and its environment. A method to artificially induce goal uncertainties during training to create agents more robust to goal uncertainties. Furthermore, a progressive training methodology consisting of scaling the difficulty of the task according to the agent success rate to ensure convergence to a solution. The chapter explains the mathematical proof for training with goal uncertainties and shows the full algorithm.

Chapter 5 adopts the framework from Chapter 4. First, it uses the framework to train four DRL agents, each with a different type of goal uncertainty for a round PiH task. It then uses the policies in simulation to validate that the agent trained with goal uncertainties has a higher peg insertion rate. Second, it applies the framework to a square PiH task. Again the chapter uses the policies generated by two agents to shows the efficacy of the framework. The chapter further inspects the elements that cause the agents trained with goal uncertainties to perform better.

Chapter 6 validates the polices generated in a simulation on a physical robotic manipulator. A physical manipulator performed both round and square PiH tasks using the transferred policies. The chapter explains the systems architecture used to conduct the experimental trials as well as the experimental setup. Finally, it reports the insertion rates of each of the agents as well as their trajectories during

insertion and compares them.

Chapter 7 presents the general conclusion of the thesis. It highlights the contributions of the work and the main achievements with an emphasis on novelty. The chapter finishes with future work that arose from the completion of the thesis.

Chapter 2: Preliminaries

This chapter summarises the concepts used throughout this work, notably Markov decision processes and reinforcement learning [1]. It explains and justifies the deep reinforcement learning (DRL) algorithms used in this work by underlining the limitations of traditional reinforcement learning algorithms. Finally, it describes the deep deterministic reinforcement learning algorithm that allows an agent to act on continuous action and state-spaces.

2.1 Markov Decision Processes

A Markov decision process (MDP) is a formalism for decision planning. It models an environment as states, controlled by actions, as shown in Figure 2.1. The goal of an MDP is to maximise performance criteria when acting in the environment.

2.1.1 Elements of a Markov Decision Process

MDPs are formally defined by the tuple $\{S, A, T, R, \gamma\}$ S, the state-space, encapsulates all states of the MDP. It may be discrete or continuous. In fine robotic manipulation, a state is a partial description of the manipulator and its surroundings. There are no formal definitions of what constitutes a state, but in this work, it respects the Markov assumption which stipulates that a state is independent of

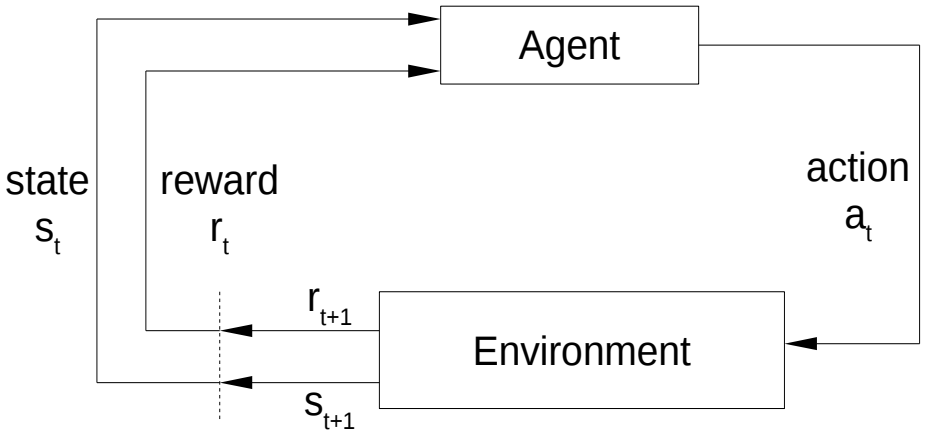

Figure 2.1: Agent-environment interaction in a Markov Decision Process [2]

the past and future, formally:

$$p(S_t|S_{t-1}) = p(S_t|S_{t-1}, S_{t-2}, ..., S_0) \tag{2.1}$$

The probability of a state being the next state remains the same whether or not any number of past states are known.

The action space, A, contains all actions that are susceptible to affect the state. The actions can also be discrete or continuous. An action in the MDP can represent a sequence of actions in the real world. They are only an abstract representation of what is available to an agent acting in an MDP.

The state transition matrix, T, gives the probability of switching from one state to another for all actions. Given a state S_t and action A_t, the next state S_{t+1} is visited with probability $p(S_{t+1}|St, At)$. Transitions in an MDP can be both stochastic or deterministic.

The reward function, R, is indicative of the performance of an agent. It is a scalar associated with taking a given action in a given state $R(s, a) \in \mathbb{R}$. It may take different forms such as $R(s)$ or $R(s, a, s')$. Finally, γ is the discount factor allowing an expected sum of reward to be bounded [3], it will be discussed later in this chapter.

2.1.2 **Solving a Markov Decision Process**

When a Markov Decision Process is fully known, the optimal mapping from state to action can be determined through dynamic programming [4]. However, in most robotic applications, T, the state transition probabilities and the rewards R are not known in advance. In this case, reinforcement learning allows an agent to iteratively build a policy that maximises the expected sum of rewards through interactions with an environment [1].

2.2 **Reinforcement Learning**

Reinforcement learning (RL) is built on the MDP framework. It is, however, a solution to solving a MDP when state transition probabilities and rewards are not known in advance. Reinforcement learning is one of the three categories of machine learning paradigms. Unlike supervised or unsupervised learning, reinforcement learning requires no prior data or model [5]. A reinforcement learning agent learns to map situations to action through interaction with an environment. An agent

takes actions in the current state and remembers the reward yielded. An RL agent builds knowledge of how to maximise rewards in the future using a value function over state and action.

Reinforcement learning problems consists of episodes which are a succession of states and actions until a terminal state, as shown in Figure 2.2.

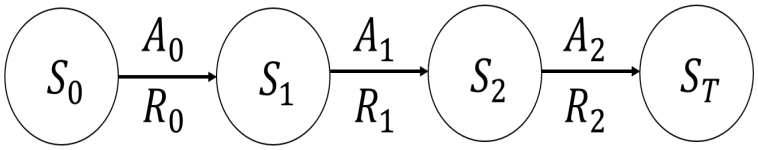

Figure 2.2: Reinforcement learning episode.

At each state S, the agent chooses an action A yielding a reward R until a terminal state S_T. The combination of states encountered during an episode is called a trajectory. The sum of discounted rewards gives the return H of an episode.

$$H_t = \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + ... \tag{2.2}$$

where γ is a discount factor allowing the sum of rewards to always be bounded for continuing tasks. Given a policy π (mapping from state to action), the value of a state-action pair for that policy is defined by

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[H_t|S_t{=}s, A_t{=}a] \\ &= \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t{=}s, A_t{=}a] \end{aligned} \tag{2.3}$$

where $\mathbb{E}$ is the expectation. A policy is the name given to the RL agent's strategy.

An optimal policy π^* maximises the expected sum of discounted rewards. It is obtained through the optimal value function computed using the Bellman optimality equation.

$$q_{\pi^*}(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_{\pi^*}(S_{t+1}, a') | S_t{=}s, A_t{=}a] \tag{2.4}$$

2.2.1 **Computing the state-action value function**

There are three classes of methods for computing the Bellman optimality equation: Dynamic Programming, Monte-Carlo learning and Temporal difference (TD) learning [2, 6]. Dynamic programming requires a model of the environment; that is, the state transition probabilities and reward distribution have to be known. Monte-Carlo learning, on the other hand, does not require a model of the environment and is, in that sense, a reinforcement learning method. Monte-Carlo learning, however, updates the value function and policies at the end of an episode only. TD learning combines ideas from both methods since it updates the state-action value estimates after each action in an episode, similarly to Dynamic Programming, but it also learns from raw experience similarly to Monte-Carlo learning. In this work, manipulators used a method analogous to temporal difference learning to complete a manipulation task .

2.2.1.1 **Q-Learning**

Q-learning is a temporal difference control method that allows a learning agent to estimate the state-action value function and to find the optimal policy [3]. It is an off-policy algorithm in the sense that it improves a policy from experience regardless of the policy used to generate that experience. Q-learning is analogous to the method used in this work to estimate an optimal policy for manipulation tasks.

Q-learning builds an estimate of the state-action value function using the Q update given as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.5}$$

where α is the step size. Q-learning updates the current state-action value estimate a step size toward the temporal difference error $[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$ using the TD-target $[R_{t+1} + \gamma \max_a Q(S_{t+1}, a)]$ as an estimate of the state-action value. The Q-learning algorithm for approximating $\pi \approx \pi^*$ is given in Algorithm 1.

Although guaranteed to converge to an optimal policy, Q-learning has limited applications for complex robotic tasks. The algorithm requires both discrete state-space and action-space. This constraint makes it obsolete for modern robotic manipulation where an agent experiences continuous values such as velocities both

Algorithm 1: Q-Learning

```
Initialise Q(s,a);
repeat
    S ← getCurrentState()
    repeat
        A ← π(S)
        R, S' ← takeAction(A)
        Q(S,A) ← Q(S, A) + α[R + γ max_a Q(S', a) − Q(S, A)]
        S ← S'
    until until terminal state;
until;
```

in the state and action-space. Nonetheless, the ideas of Q-learning are fundamental to much more powerful algorithms.

2.2.2 **Deep Reinforcement Learning**

Deep reinforcement learning (DRL) refers to the category of reinforcement learning approaches that use artificial neural networks to approximate the value function and/or policies [7]. DRL is more suited to complex robotic manipulation tasks since it enables the learning agent to reflect on large state-spaces more representative of robotic environments. The work here only focuses on one DRL method best suited to robotic manipulation task, Deep Deterministic Policy Gradient (DDPG).

DDPG was selected in this work for the following reasons: First, to align with the current literature as DDPG appears to be a recurrent solution to solve Robotic manipulation task through DRL [8–10]. Second, to ensure that Hindsight

Experience Replay, which is also used in this work, improves the performance of a deep learning agent since it was first developed as a wrapper for DDPG [11]. Finally, to follow the results from preliminary experiments that suggested that DDPG hyper-parameters were easier to tune and yielded better performance faster. The latter are not documented in this work.

2.2.2.1 **DDPG**

In this work, we propose a reinforcement learning approach to unsolved robotic manipulation problems. These tasks are formulated with states s and actions a that are multi-dimensional and continuous. A DDPG [12] algorithm can allow an agent to develop a desirable behaviour by iteratively improving it. DDPG is part of the larger class of reinforcement learning algorithms using an actor/critic architecture [6] and an even larger class of policy gradient methods.

Policy gradient methods aim to assign a higher probability of being selected to actions with a higher expected sum of rewards for a given state [13]. The objective remains to maximise the expected sum of rewards:

$$\mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}] \tag{2.6}$$

where T denotes the end of an episode and $r_{t+1} = R(s_t, a_t)$. An objective function

can be defined as:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}] \tag{2.7}$$

with the aim to maximise the policy parameter θ, this can be re-written as:

$$\begin{aligned} J(\theta) &= \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta] \\ &= \sum_{t=0}^{T-1} P(s_t, a_t|\tau) r_{t+1} \end{aligned} \tag{2.8}$$

where $P(s_t, a_t|\tau)$ is the probability of encountering the state action pair (s_t, a_t) in the trajectory τ. θ is maximised using the gradient of the objective function given by:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{t=0}^{T-1} \nabla_\theta P(s_t, a_t|\tau) r_{t+1} \\ &= \mathbb{E}[\sum_{t=0}^{T-1} \nabla_\theta log P(s_t, a_t|\tau) r_{t+1}] \\ &\approx \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta H_t \end{aligned} \tag{2.9}$$

where H_t is given in 2.2

Vanilla policy gradient algorithms have high variances in rewards and noisy gradients, leading to poor performance [2]. This is due to the gradient being subjected to Monte Carlo updates, that is using the return from an entire episode. Actor/Critic methods solve this by introducing a baseline which is a more conservative estimate

of the return and which contributes to reducing the variance:

$$\nabla_\theta J(\theta) = \mathbf{E}[\sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta (H_t - b(s_t))] \tag{2.10}$$

In practice, Q values have been proven a good alternative to the full episode return:

$$\nabla_\theta J(\theta) = \mathbf{E}[\sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta (s_t, a_t) Q_w(s_t, a_t)] \tag{2.11}$$

where w indicates that a neural network represents the Q function. It is updated using the temporal difference error as in Q-learning. There are, by consequence, two networks for learning. The critic network estimates the value functions while the actor network updates the policy as suggested by the critic.

DDPG is both a policy gradient and actor/critic method. It uses an actor network θ^μ, and a critic network θ^Q and computes the gradient of the policy. However, DDPG differentiates itself from Advantage Actor-Critic, another commonly used Actor/Critic algorithm, for several reasons. In addition to the policy and value function networks, DDPG uses two additional networks $\theta^{\mu'}$ and $\theta^{Q'}$ termed target policy network and target Q network respectively. These two networks are time delayed copies of their original counterparts and are used to perform the update. This technique dissociates network update from the value calculated by itself hence improves convergence and stability. Besides, in DDPG, the actor network directly maps states to actions rather than updating the probability distribution

over the actions space. The consequence of this is that DDPG applies to scenarios with continuous actions spaces contrary to classical actor/critic algorithms. The pseudo-code in Algorithm 2 shows the DDPG algorithm.

The Q values are updated similarly as in Q-learning using the Bellman equation and the target networks:

$$y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{2.12}$$

where i is the index of the transition in a sampled batch. The weights of the Q network are then updated using the loss:

$$L = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{2.13}$$

between the updated Q-values and the original ones. The policy network is then updated towards the gradients of the objective function:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \tag{2.14}$$

A soft update of the target network weights toward the original network weights is performed as shown in line 20 and 21 of Algorithm 2.

Similarly to all RL agents, a DDPG agent has to choose between exploiting the current policy or exploring the state-space [14]. It has a choice between using its

current knowledge to execute the best action or to explore the action space by choosing any other action but the one maximizing the state-action value function. Adding artificial noise either during action selection, where actions are drawn from a Gaussian distribution or a OrnsteinâUhlenbeck process [12], or in the neural nets where noise is added to the network weights [15], creates more exploratory agents. These change the action selected to stimulate exploration.

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t \tag{2.15}$$

2.2.2.2 **Hindsight Experience Replay**

The performance of deep reinforcement learning algorithms such as DDPG is dependant on the quality of the reward signal R [16]. Poorly formulated or sparse rewards lead to sub-optimal performance of the learning agent and by consequence, undesirable behaviour from a robot. Shaped reward functions, discussed in the next chapter, give additional indicators of performance to a learning agent besides a binary reward indicative of the completion of the task [16]. This solution, although viable in theory and practice, requires engineering of complex reward functions tailored to one particular task. It is one of the most challenging steps in the development of self-learning robots [17].

Hindsight Experience Replay (HER) [11] is a method to utilise better the experience collected during each episode of a reinforcement learning task. It applies to any off-policy reinforcement learning algorithm such as DDPG and allows the agent to

Algorithm 2: Deep Deterministic Policy Gradient

Initialise $Q(s, a|\theta^Q)$ and $\mu(s|\theta^\mu)$ with weights θ^Q andθ^μ;
Initialise Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$;
Initialise Replay buffer $\mathcal{B}$
$N \leftarrow batch\_size$
repeat
    Initialise action noise $\mathcal{N}$
    $s \leftarrow getCurrentState()$
    repeat
        $a_t \leftarrow \mu(s|\theta^\mu) + \mathcal{N}$
        $r_t, s_{t+1} \leftarrow takeAction(a_t)$
        $storeTransition(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{B}$
        if $size(\mathcal{B}) \geq N$ **then**
            $batch \leftarrow sample(\mathcal{B}, batch\_size)$
            $y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
            Update critic with loss:
            $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
            Update actor with gradient:
            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
            Update target network:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q^{'}}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu^{'}}$
        $s \leftarrow s_{t+1}$
    until *until terminal state*;
until;

learn from sparse rewards instead of shaped ones. Where classical DDPG agents learn little from unsuccessful episodes, DDPG combined with HER learns as much from unsuccessful episodes as from successful ones using either binary or shaped rewards.

Two mains ideas constitute HER. The first one comes from the method of universal value function approximators, where both state $s \in S$ and goal $g \in G$ are used as inputs to the value functions. This method leads to a better generalisation over previously unseen state-action pairs. The second and main idea in hindsight experience replay is to store the terminal state of each episode as an alternative goal in the replay buffer and use it as well as the actual goal to learn. Given a trajectory $s_0, s_1, ..., s_T$, each transition $s_t \rightarrow s_{t+1}$ is stored with the original goal as well as with the alternative goal $m(s_T)$.

Combining HER with an off-policy algorithm leads the learning agent to pursue more and more challenging goals towards the original goal. In practice, HER has been proven to work better with sparse, binary rewards which punish any state other than the goal state. HER proposes a partial solution to overly complicated reward functions.

2.3 Conclusion

This chapter introduced the reader to the concept of reinforcement learning. It showed that there exist different algorithms to solve a reinforcement learning problem and that novel, state-of-the-art ones are capable of solving problems with large action-space and state-space in theory. It explained in detail the Deep Deterministic Policy Gradient algorithm used in this work to solve manipulation tasks. Finally, it explained how the performance of any off-policy algorithm can be enhanced using hindsight experience replay.

This precursor helps the reader to understand how the paradigm can be applied to robotic tasks, more precisely, manipulation tasks. The next chapter provides a review of the literature on the subject.

Chapter 3: Literature Review

3.1 Introduction

The field of robotic manipulation has transitioned from controlled environments with low bandwidth sensory inputs to noisy, uncontrolled environments with high dimensional inputs. The tasks themselves, have evolved to be more involved, often requiring both motion and task planning [18], sequential actions, and fine/specific behaviours. Throughout their whole evolution, all manipulation tasks have been subjected to and remain affected by uncertainties.

Technological advances have allowed reducing uncertainties from sensing, actuation and perception significantly, nonetheless today's complex and fine manipulations task remain sensitive to small uncertainty from these modules. Besides, contemporary tasks rely on manipulator control, task planning and knowledge of the environment where the uncertainties are still significant.

First, this chapter summarises the methods that have been used in the literature to mitigate the effects of uncertainties on manipulation tasks. Second, it shows that the complexity of manipulation tasks and the types of uncertainties they are subjected to has lead researchers to use reinforcement learning approaches. Third, it shows that DRL manipulation tasks are still prone to uncertainties and that

they are often dealt with implicitly. Finally, this chapter highlights the persistent gaps in the literature; notably that, uncertain DRL goals are not addressed; there are no benchmark state-space models for manipulation tasks and that DRL agent design lacks a consistent methodology.

3.2 **Mitigating Uncertainties in Robotic Manipulation**

There are multiple types of uncertainties in robotic environments. Thrun *et.al.* [19] stated that there are five main sources of uncertainty to any robotic task: environments, sensors, actuators, models and computation:

- Uncontrolled robotic environments are inherently unpredictable; many algorithms such as path planner require a known environment configuration [20]. Uncertainty in the configuration can lead to obstacle collision.
- The physical world limits sensors for what they can perceive, and they are subject to noise which in turn affects the features such as position of objects or manipulator's end-effectors [21].
- Robotic actuators are uncertain due to control noise, motor wear and the use of and low-cost actuators. If the environment is known, fine manipulation task are still prone to fail to due to control uncertainties [22].
- Models are inaccurate abstractions of the real world. Solutions to manipulation tasks may rely on an accurate model of a manipulators differential kinematics which is itself uncertain.

- Finally, most algorithms relied on by manipulators often trade off speed for accuracy. State estimation algorithms such as particle filters [23] and others [24, 25] trade speed of execution for accurate state estimation hence have embedded uncertainties. All these sources of uncertainty have significant effects on fine robotic manipulation tasks.

3.2.1 **Partially Observable Markov Decision Process**

Partially Observable Markov Decision Process or POMDP [26] is a framework for decision making under uncertainty. It is pertinent to high-level sequential task planning under partial observability. Based on its current belief of the state, a robot takes action and subsequently update its belief. POMDP is a general formulation of Markov decision processes where states are not known but rather observed [27].

The framework takes into consideration non-deterministic state transitions to plan for the best action. It is analogous to robotic manipulation tasks where sensory feedback, for example, are better represented as a Gaussian distribution. The framework is generic and applicable to any state belief; hence it has been the subject of numerous research for application to a wide variety of manipulation tasks.

Li *et.al*. [28] used the framework for planning actions of a robot searching for an object in a cluttered space. The robot first had to move occluding objects to

find an occluded one. Vien *et.al.* [29] employed POMDPs for both an end-effector reach task and a Peg-in-Hole task. Finally, Pratama *et.al.* [30] used POMDPs for a robot to learn how to pour water in a glass.

POMDPs rely on a state belief used for planning. The state of a manipulation task may include aspects of the robot and the environment, such as objects poses. Some approaches additionally rely on Bayesian filtering methods to obtain an accurate state estimation. In the paper from Pajarinen *et.al.* [31], the authors used a particle filter for state estimation of a manipulation task. A robot had to move several red objects from a cluttered space to a designated area. They used an RGB-D camera to segment the scene and perform the grasp. The belief state was a point cloud segment corresponding to an object hypothesis. In latter work [32], the authors relied again on particle filtering. The manipulation task aimed to move "dirty" dishes into a selected area. The authors used a segmented point-cloud to obtain a state belief which corresponded to objects attribution, historical data of observation and action success for each object. Similarly, Kaelbling *et.al.* [33] used POMDPs to perform a mobile manipulation task. They proceeded to use an unscented Kalman filter to estimate the pose of the robot.

POMDPs allows a robot to consider uncertainties before acting based on a non-deterministic environment model. The frameworks presented drawbacks for fine robotic manipulation tasks. It did not scale with the dimensionality of the state-space hence was only successful in problems with discrete state-spaces [34] and small continuous problems [35] . It required additional processing of the sensory

data through a state estimator that prevented robots from acting in real-time [23]. Finally, it was useless for a task with environment contact due to lack of an accurate model [36]. POMDPs take into account uncertainties before action execution. The literature also presents ways to mitigate the effects of uncertainties after execution.

3.2.2 **Compliant Motion**

Compliant motion is an alternative to mitigate uncertainty after the act. Given a belief state, a robot takes action in the environment. Online sensor measurements and compliant motions can allow a robot to slightly modify its behaviour based on newly acquired knowledge [32].

Compliant motion can apply to simple manipulation tasks. Lozano-Perez *et.al.* [37] presented a compliance framework to allow a robot to complete tasks analogous to 2D Peg-in-Hole. They mentioned that in the presence of uncertainty, their approach based on backwards-chaining of pre-image identified locations in space where reaching the goal is not guaranteed. Using compliant motions, they showed that there existed a sliding behaviour for goal completion. Pai *et.al.* [38] considered a similar framework. The authors considered both environment and end-effector uncertainty to develop a solution based on joint compliance of a planar manipulator for a Peg-in-Hole task.

Compliant motion can also apply to more complex systems. Ajoudani *et.al.*

[39] proposed a compliant framework for mobile manipulators. They enabled compliance of a complete humanoid during a valve turning task. Valve turning is sensitive to uncertainties in both valve position and robot position. Small inaccuracies quickly lead to large forces applied to a manipulator that follows a wrong turning trajectory. Their approach based on motion primitives and a compliant stiffness matrix showed successful results experimentally.

Finally, although compliant motions often require complex modelling of a manipulator, there exists a solution to simplify this process. Balatti *et.al.* [40] proposed a self-tuning impedance controller for a robot to adapt to uncertain task conditions. The controller parameters are modified based on the robot's interaction with the environment captured via fused force and vision data. The authors used a finite state machine to perform an object manipulation task where the machine iteratively returned to a parameter tuning state.

Compliance after acting allows a robot to compensate for uncertainties from the environment. It shows success experimentally; however, these methods require significant effort for parameter tuning, which differ from robot to robot or automatic parameter tuning methods that can become intractable for complex robot models. Specific types of fine manipulation tasks can benefit from compliant motion, but the framework alone cannot compensate for all types of uncertainties.

3.2.3 **Peg-in-Hole**

Peg-in-Hole is an example of a fine manipulation task that remains challenging to solve with uncertainties. It refers to the generic task of insertion of an object (the peg) in a hole. It often requires adhering to constraints such as force and torque applied to the object through means of a robotic manipulator. Peg-in-Hole does not impose conditions on the shape of the object or hole but instead encloses any generic pair. In that sense, the problem is analogous to many real-world scenarios such an assembly tasks, key in a lock insertion, screw driving and more. The difficulty of the task is adjustable by varying clearance and peg/hole geometry. This task is ideal to benchmark fine manipulation approaches.

Peg-in-Hole for real-world applications is a challenging problem for numerous reasons. First, it inherently has two indistinct phases, search and insert. While many approaches focus on the latter, the former is crucial to the completion of the task. Both phases require different behaviour from the robot; besides, it can be challenging to detect the transition between one or the other and to this extent, significant effort is required to program these behaviours manually. Second, uncertainties affect both phases; hole location and orientation in the first phase and contact state in the second phase. Finally, the haptic nature of the task makes it dangerous to attempt on physical hardware.

The problem has attracted much interest in the robotic community, which led to different methods to solve it. The most widespread method relies on a hybrid

force/position controller known as impedance control. This technique allows the stiffness in the manipulator to reduces artificially hence mitigating the effects of uncertainties during the insertion phase.

This form of compliant motion can be applied to solve the Peg-in-Hole problem. Song *et.al.* [41] proposed such a solution for assembly of complex geometry. Their method relied on feature extraction from CAD models, online pose estimation based on vision, choice of assembly direction and selection of force thresholds for insertion. A human-inspired impedance control technique allowed the robot to complete the Peg-in-Hole task, which were separated into 3 phases: Detection of contact force, alignment and avoidance jamming. The authors ensured limited uncertainties by relying on more that one mitigation technique.

Wang [42] considered a similar technique for a simpler 2D version of the Peg-in-Hole problem. Impedance control was used in joint space to control compliance of the peg hence decreasing the effects of uncertain peg position obtained through vision. The authors additionally made use of tendons as naturally compliant actuators.

Other approaches from Jokesch *et.al.* [43] and Wei *et.al.* [44] also combined impedance control to other techniques to diminish the effects of uncertainties in Peg-in-Hole like scenarios. Although these approaches presented positive results, they also had drawbacks such as reliance on known environment models, oversimplification of the problem or accurate parameter tuning.

Another approach consists of performing the task using vision feedback combined with visual-servoing [45–47], human demonstration [48] or adaptive control [49]. Other approaches such as Evolutionary algorithms [50], Particle filters [51] and sensor-less techniques [52] participated to mitigate the effects of uncertainties in a PiH task.

3.2.4 **Summary**

The literature shows that manipulation tasks considered both tasks and motion planning. These tasks have numerous sources of uncertainties that can be mitigated both from a task planning and motion planning point of view. POMDPs enabled agents to plan under uncertainty but required environment models and did not scale to large tasks. Compliant motion techniques such as impedance control always paired with other techniques for robotic manipulation. It allowed to mitigate the effects of uncertainties after the act and presented successful results. This technique showed promising results for a Peg-in-Hole task but required accurate parameter tuning and further modelling of the environment.

It is evident that there currently does not exist a state-of-art technique to solve manipulation tasks under the effects of uncertainties. CAD modelling and geometry-based technique do not account for uncertainties whether aleatoric or epistemic and are therefore inadequate for complex fine manipulation tasks. To this extent, model-free reinforcement learning techniques were a better alternative to solve complex manipulation problems. The paradigm relies on repetitive interactions; hence, the

resulting behaviour is the product of the environment with embedded uncertainties. The next section considers reinforcement learning approaches to manipulation tasks and surveys on how to deal with uncertainty in these applications.

3.3 Robotic Manipulation, Reinforcement Learning and Uncertainty

3.3.1 Explicitly Addressing Uncertainties in RL

A reinforcement learning agent minimises uncertainty about its perceived environment by interacting with it. Analogous to POMPDs, a reinforcement learning agent builds a policy that accounts for its uncertain knowledge of the environment but with no prior model. Seldom approaches explicitly addressed uncertainty in a reinforcement learning context. However, the current trend of using RL for more and more fine and complex tasks forced recent approaches to address them explicitly.

Disregarding uncertainties can be costly in RL manipulation tasks. Khan *et.al.* [53] expressed the necessity of having cost functions dependant on uncertainty in order to allow reinforcement learning agents used on real hardware to learn safely. They proposed an uncertainty-aware reinforcement learning approach to a collision avoidance task. They first trained a neural network to predict collision probability. Then, the authors obtained an uncertainty measure of their model using a bootstrapping algorithm. Finally, they used the uncertainty-aware collision

model to form a cost function used by a model-based reinforcement learning algorithm.

Uncertainty awareness can also be used further downstream in the RL process. Clements *et.al* [54] proposed a method for estimating risk and uncertainty for deep reinforcement learning. Uncertainty for reinforcement learning algorithms falls under two categories: epistemic and aleatoric. The latter refers to uncertainty stemming from the randomness of the environment. The former is uncertainty from imperfect knowledge of the environment [55]. Their Bayesian inference-based approach to estimating uncertainty was used in a reinforcement learning context to direct the action selection process.

Epistemic uncertainty decreases by interacting with the environment. In reinforcement learning, this constitutes the exploration vs exploitation problem. By consequence, exploration technique such as noise insertion and random exploration mitigate the effects of uncertainty and in turn, allow for better generalisation of learnt policies. This was the premise from Martin *et.al*. [56] who proposed a framework for more efficient exploration improving on classic techniques such ϵ-greedy in high dimensional environments.

3.3.2 **Reinforcement Learning for Robotic Manipulation**

Manipulation is the handling or controlling of a tool or object skillfully. By definition, robots need to acquire skills to be able to manipulate objects in their

environments. Reinforcement learning (RL) was identified since its early stages as a promising paradigm due to its applicability for stochastic environments. Seed approaches include the ones from Asada *et.al.* [57], Barto *et.al.* [58] and Gullapalli *et.al.* [59] with attempts to solve the Peg-in-Hole problem in 2D. RL allowed to represent or approximate the non-linear models governing most manipulation tasks. Leaps in computer performances and breakthroughs in the field of machine learning have allowed researchers to use the paradigm for more and more complex applications.

Although viable from a theoretical point of view, reinforcement learning presents numerous challenges in applications stemming from large, uncertain environments. These have led to a variety of ways RL is used to learn manipulation skills.

3.3.2.1 **Learning from Demonstration**

Learning from demonstration is a popular technique to accelerate learning. Expert knowledge incorporated in the learning can reduce uncertainty significantly by allowing an agent to improve an already well-formed policy. Learning from demonstration takes multiple forms in the literature.

Expert demonstrations allow a robot to monitor how an expert, generally a human, would proceed for a given scenario and to use this information as a precursor to improving a policy. Zhu *et.al.* [60] used a model-free reinforcement learning algorithm to grasp objects for raw RGB camera data. They leveraged a few rounds

of expert robot jogging scenarios to form an initial policy. Rajeswaran *et.al.* [61] again showed accelerated learning performances for various manipulation tasks, namely in-hand manipulation. They provided expert demonstrations for the latter by capturing data from a specially designed glove.

Kinaesthetic teaching is another form of incorporating expert knowledge to the reinforcement learning process. It consists of physically guiding the robot while performing the desired manipulation task. It is a widely spread technique as explained by Popov *etal.* [62]. Kalakrishnan *et.al.* [63] used kinaesthetic teaching for robots to learn fine manipulation with compliant motions. They initially guided the robot to perform the task in order to obtain an initial policy.

Other ways to incorporate expert knowledge includes dynamic motion primitives when instead of actions mapping directly to robot actuator control, each action consists instead of a sequence of actions known as motion primitives defined in advance. They were the premise from the work in [64] and [65] where the authors used dynamic motion primitives for reinforcement learning.

On large domains, learning from demonstration may require many expert runs to obtain an adequate base policy. Furthermore, some applications have shown promising results with no prior knowledge at all. These implementations relied on accurate simulation in order to transfer learnt policies directly to physical robots and high computational power to train learning agents on tens of millions of observations. These approaches all used model-free reinforcement learning

algorithms.

3.3.2.2 **Model-free RL**

Model-free reinforcement learning is a recurrent method considered for robotic manipulation tasks. Although only used for tasks with simple representation until the start of the decade [66–68], the publication of novel algorithms such as DDPG [13], Normalised Advantage Function [69] and more recently Hindsight Experience Replay [11] has led to an explosion in the number of applications of reinforcement learning to robotic manipulation tasks.

RL can apply to various types of robotic manipulation tasks. Clavera *et.al.* [70] proposed a model-free reinforcement learning approach to pushing an object to the desired location using a robotic manipulator. The authors in this paper exposed some crucial aspects of deep reinforcement learning that are in most similar work left un-addressed. Firstly, the authors stated that different reinforcement learning tasks might require additional state inputs in what they termed the minimal observation consisting of the robot joint angles and velocity. Second, they differentiated between training in simulation and a real robot. They proposed a framework for modularity that decomposed the policy into separate modules to facilitate transfer to a real robot. The authors, however, did not address uncertainty explicitly in their work. De La Bourdonnaye *et.al.* [9] used a model-free RL algorithm for similar end-effector reach task. In their paper, the authors studied the effects of different reward functions on learning performance but also

left uncertainties not addressed.

RL/DRL also showed uses in fine manipulation tasks such as PiH. Xu *et.al.* [10] used a deep reinforcement learning algorithm for a double Peg-in-Hole manipulation task. The authors assumed perfect knowledge of the goal, that is hole position and orientation. They only considered the insertion part of Peg-in-Hole. They mitigated the challenge of reward formulation partly by introducing a fuzzy reward system. Haarnoja *et.al.* [71] also formulated an assembly task as a deep reinforcement learning problem which they solved using a model-free method. A soft q-learning agent explicitly balanced exploration and exploitation to obtain a more robust, certain policy.

Uncertainties were also implicitly mitigated in uses of model-free DRL approaches. Gu *et.al.* [8] used a model-free reinforcement learning approach to several manipulation tasks, including door opening and pick-and-place. Although again not addressing uncertainties of the task explicitly, the authors proposed a framework for asynchronous training which leverages experience from several learning agents rather than one − this reduced epistemic uncertainty by exposing the agent to a large portion of its environment. The authors remained vague about state-space formulation for the diverse tasks.

Pre-trained agents may reduce learning time and mitigate the effects of uncertainties during further learning. Devin *et.al.* [72] proposed to trained model-free agents end-to-end, that is, controlled the actuation directly from inputs, for general

purpose uses. Their framework allowed the use of general purpose agents over given degrees of variation (DoV); degrees of freedom of a manipulator, for example. The approach required additional training for a given DoV, which contributes to reducing uncertainty for that scenario.

Finally, the wide variety of manipulation tasks approached using RL shows the viability of the framework. Applications such as grasping and dexterous manipulation [73–75], co-manipulation [76] and collision avoidance for manipulators [77] all used model-free reinforcement learning approaches to solve large and continuous manipulation problems. While some works addressed uncertainties explicitly, most relied on the intrinsic abilities of reinforcement learning agents to act in stochastic domains.

3.3.2.3 **State-space Representation**

State-space representation participates in mitigating the effects of uncertainty in reinforcement learning tasks as hinted by the work from Clavera *et.al.* [70] earlier. The literature disregards this aspect, although crucial to the success of any manipulation scenario. All reinforcement learning problems have an underlying Markov decision process characterising the task in terms of states, actions and rewards. These are used by the reinforcement learning agent to make decisions. The RL-agent state-space is the same or a subset of the MDP's which in turn is a subset of the world's state space [2]. Depending on the problem at hand, defining the RL-agent's state-space can be challenging and may drastically affect

the learning by reducing agents epistemic uncertainties.

A common way of representing a robots environment is to use raw camera pixels as an input vector to a deep neural network. It was the premise in the work from Anso *et.al.* [78] who used different types of raw images to solve a pellet eating Attari game or again in the work from [79] for a similar application. Raw pixels alone cannot suffice for a task requiring contacts with the environment, and additionally, there is a lack of information about the level of accuracy attainable in manipulation tasks using only raw pixels. Gu *et.al.* [8] used raw pixel augmented with other information and showed promising results for a manipulation task. The technique they used, however, represents well the current harmful trend towards which reinforcement learning applications are leaning. DRL algorithms can process large amounts of information; hence authors include as much information as available as part of the state-space. Some of the information included may well be redundant and create a less robust policy. However, there are no benchmarks to formulate fine manipulation agents' state-space models.

A more recurrent way to represent state-space models for robotic manipulation is to have as base observation, the manipulator joint angle, and their derivative. Sangiovanni *et.al.* [77] used these elements in their state-space representation of a normalised advantage function agent. The agent trained for real-time collision avoidance in a simulation. Levine *et.al.* [80] employed a similar formulation to train a control system destined for visual servoing. Joint states alone do not represent the state of a robot accurately for a manipulation task but can only be

used to learn manipulator control.

As an alternative to the robots joint states, manipulation tasks may use the robots end-effector state. It can be obtained through forward kinematics as shown in the work in [77, 81, 82], or through other means such as visual markers [83], although more significant uncertainty may arise from this type of state estimation.

Finally, as part of state-space models, authors considered including indications on the current manipulation task, making them application-specific [11]. It unclear in this type of application, just how much information the state-space model should include on the environment, and again how much the environment features help in the mitigation of uncertainties while retaining learning efficiency. This reinforces the need for a benchmark. Additionally, most reinforcement learning applications focused on diminishing uncertainty of the robots while failing to address precisely how the learnt polices behave given uncertain states or if it was even possible to build an adequate policy.

3.3.2.4 **Reward Formulation**

A RL agent relies on a reward signal to build a policy. In Markov decision processes, the agent obtains a reward in a goal state. Such sparse rewards are, however, not viable for complex reinforcement learning tasks such as fine manipulation. To better guide the agent and reduce its uncertainty, several approaches propose to give additional information through a scalar reward.

Potential-based rewards are an addition to the MDP reward. Mataric *et.al.* [84] proposed the idea originally which Ng *et.al.* [16] refined later. Contrary to the standard MDP reward, potential-based rewards were incurred at every step of the decision process and reflect closeness to the goal. Ng *et.al.* showed that potential based rewards significantly accelerated the learning while maintaining the convergence property for discrete reinforcement learning algorithms. The paradigm was recurrently addressed in the contemporary reinforcement-learning literature [85–89] with extensions of the idea to automatic reward shaping, [90–93] and to applications to robotic manipulation.

A potential reward signal can comprise multiple parts. Popov *et.al.* [62] proposed a composite reward formulation for dexterous manipulation. A robot had for a task to stack Lego bricks. The reward signal included a sparse component and a potential-based one. The former was a pre-set positive scalar value for reaching the brick and grasping the brick while the latter consisted of the distance between the hands and the brick before the grasp and the brick and the stack of bricks after grasping.

Although potential based reward appears as the most common way to reduce epistemic uncertainty from sparse rewards, some aspects of the paradigm make it challenging to apply to complex, fine manipulation tasks. The design and engineering of an adequate reward signal is a time-consuming endeavour requiring trial and error, making it perilous on physical hardware [17]. Additionally, it is unclear how other uncertainties such sensor noise affects potential based reward

signals as most of them rely on a perfectly known goal distance. Finally, some applications, such as Peg-in-Hole, do not have a goal known in advance. In that sense potential based rewards might not be applicable.

The literature shows alternative methods that were not goal dependant or require a minimal goal specification for formulating rewards. Xu *et.al.* proposed a fuzzy reward system for a Peg-in-Hole insertion task. They used a two layer fuzzy logic system to output a negative reward for the RL agent. Their use of fuzzy logic allowed to the elimination of some of the uncertainty from an inaccurate measurement of the peg depth; however, it is unknown how the authors defined the inference rules.

Furthermore, there exist other techniques where goal specification is less crucial such as inverse reinforcement learning [94, 95] where instead of learning a policy from interaction with the environment, an agent learned the reward signal from collected data. Concept networks [74] were also used where conceptual behaviours could be trained and reused for different tasks with reduced training times. Hindsight experience, which leverages all of the agent experience to learn from sparse rewards, is also a helpful technique.

Similarly to the state-space formulation, it is unclear just how a reward formulation participates in mitigating uncertainties during the RL process. Authors rely on the paradigm for a solution while disregarding if specific reward design can provide a more precise signal for the RL agent to learn, resulting in an overall more efficient

learning process.

3.3.3 **Summary and Gaps**

The literature has addressed uncertainties in several ways for manipulation tasks. POMDPs allowed a robot to take into account the effects of uncertain knowledge before executing actions but did not scale to the dimensions required for today's fine manipulation tasks. Compliant motions, on the other hand, served to mitigate the consequence of uncertainties after execution. Instances of the framework such as impedance control showed promising results; however, it alone cannot be used as the only method to diminish the effects of uncertainties in manipulation tasks as it required careful tuning of parameters and may still cause hardware damage in some instances.

Deep reinforcement learning eliminates all the shortcomings of traditional methods and offers a one-stop solution to solving manipulation tasks. The paradigm is prone to uncertainties that are mitigated through various methods in the literature including uncertainly awareness techniques and ones that implicitly deal with uncertainties. Model-free DRL implicitly mitigates uncertainties via extensive environment interaction, adequate state-space formulation and reward signal design.

Deep reinforcement learning has the potential to be a standard tool for robotic manipulation. DRL builds a policy around inherent epistemic and aleatory uncer-

tainties. Nonetheless, the literature that show DRL applications to manipulation tasks still lacks rigour, benchmarks and usability. The following comments expresses those gaps:

- First, state-space models, one of the main contributors to mitigating uncertainties, do not have a formulation methodology to allow them to be consistent across the literature. It becomes apparent when the same manipulation problem, Peg-in-Hole, was expressed in vastly different ways in the works from [10, 96, 97].
- Second, it is unclear how uncertainties can be mitigated explicitly for the more modern, goal-oriented DRL [11, 98], where goal uncertainties have not yet been addressed.
- Third, the literature does not offer any framework that includes hyperparameter tuning, "sim-to-real" policy transfer or policy convergence techniques to train DRL agents for fine manipulation tasks. These aspects are omitted from the current body of work.

These aspects will be the main focus of the work in the thesis.

Chapter 4: A Framework for Fine Manipulation under Goal Uncertainty using Deep Reinforcement Learning

This chapter proposes a framework to allow a deep reinforcement learning (DRL) agent to perform manipulation tasks with uncertain goals. It consists of progressively training a DRL agent across more and more challenging versions of a manipulation task with a degree of uncertainty in the goal. This chapter shows that state-of-the-art DRL algorithms rely on an exact formulation of a goal state to learn from large and high-dimensional environments. A review of related works highlights the shortcomings in the literature. Finally, the chapter details all aspects of the framework and justifies its viability for training DRL agents.

4.1 Introduction

Fine robotic manipulation tasks are hard to solve with conventional methods since these rely on controlled environments and precisely known quantities. DRL is a versatile paradigm that applies to a range of robotic tasks, including fine manipulation. It does not call for precise models of robotic manipulators or manually engineered solutions to solve manipulation tasks. Algorithms such as deep deterministic policy gradients (DDPG) allow agents to establish non-linear

relationships between state and action, to create policies for solving particular tasks.

Model-free DRL algorithms use a reward signal as well as input features that are functions of the goal. A goal is the desired state of the agent. In a real-world environment, elements forming the goal, such as velocities, positions and forces, are prone to uncertainties. Relying on a perfect version of the features for training DRL agents while disregarding their real-world counterpart leads to poor results.

Similarly, DRL can often fail to converge to a solution due to sparse reward signals from fine manipulation tasks. The overall lack of standards in the literature to formulate these task, tune DRL algorithms hyper-parameters and train agents makes this approach inapplicable to real-world problems.

Finally, DRL algorithms require the formulation of an observation space from which to learn. An adequate formulation of this space contributes to significantly reduce the uncertainties of the task. Inadequate formulation of this space leads to uncertain DRL agents equating to inefficient or unsuccessful policies.

The framework in the chapter addresses state-space formulation, policy convergence, training methodology and goal uncertainties. Chapter 3 showed that these aspects formed the gap between applying DRL agents to real-world fine manipulation tasks and the current reality. DRL agents trained based on the framework in this chapter perform better across a range of environments other than the one

trained in hence are filling the gap further.

4.2 Related Work

4.2.1 Deep Reinforcement Learning for manipulation

DRL has been used to solve robotic manipulation tasks as early as the 1990s. The works from Mataric *et.al.* [84] and Asada *et.al.* [57] presented fundamental approaches to solving simplified manipulation problems using reinforcement learning and neural nets. More recently, deep reinforcement learning algorithms such as deep deterministic policy gradient [13] and hindsight experience replay [11] were capable of solving more complex problems.

Fine manipulation tasks can be solved using DRL. Inoue *et.al.* [96], for example, considered the round peg into a round hole problem. They observed forces, moments and position of the peg. The actions of the agent consisted of the desired force on the peg and desired orientation. They considered both hole localisation and peg insertion separately. The agent received a positive reward upon the successful insertion of the peg for both stages. Upon a failure in the search stage, the agent received a negative reward, proportional to its distance from the goal. This form of reward is potential-based [16] as it increases or decreases with distance to the goal.

DRL has also solved more challenging versions of the same task. Xu *et.al.* [10] considered a more challenging version of the problem where two pegs rigidly attached to each-other are inserted simultaneously. They observed the position and orientation of the peg as well as forces and moments applied to the peg. The actions were translations and rotations of the peg. The reward included two parts: The first part is similar to the one in the work from [96], dependent on the number of steps taken. The second part depended on fuzzy logic, rewarding depth while considering maximum force and torque thresholds. The final reward was the summation of the two parts.

State-space models and reward signals lacked consensus in these applications of the same manipulation task. Besides, both applications used a potential-based reward, relying on the real goal location. These measures are prone to uncertainties in a non-controlled environment; hence the policies created cannot generalise to change.

4.2.2 **Uncertainty Awareness in Reinforcement Learning**

A DRL agent learns to maximise the sum of rewards in the environment while implicitly dealing with uncertainties. Nonetheless, these uncertainties, whether epistemic or aleatoric [55], may contribute to unsafe and/or unsuccessful learning. Reciprocally, explicitly considering uncertainties may improve performance of deep learning agents.

In manipulation tasks, relying on DRL while disregarding uncertainties can be costly. Khan *et.al*. [53] expressed the necessity of having cost functions dependent on uncertainty for safe, real-world reinforcement learning. They proposed an uncertainty-aware reinforcement learning approach to collision avoidance for a manipulator. They first trained a neural network to predict collision probability. The authors then obtained an uncertainty measure of their model using a bootstrapping algorithm. Finally, they used the uncertainty-aware collision model to form a cost function used by a model-based reinforcement learning algorithm. Similarly, Clements *et.al* [54] proposed a method for estimating risk and uncertainty for deep reinforcement learning. Their Bayesian inference-based approach to estimating uncertainty is used in DRL to direct the action selection process. They showed their approach in Atari games.

4.2.3 **Feature Selection**

Agent observations are a partial representation of the underlying state of the world. Although deep neural nets can allow agents to create policies over large observation spaces both for virtual, Mnih *et.al*. [7] and real-world applications, Levine *et.al*. [80], the choice of features is crucial to the agents' learning performance [99]. Feature selection for supervised learning [100, 101] and model-based RL [102–104] have well established methods. It is not the case for model-free DRL. Deep reinforcement learning approaches to manipulation problems such as the one from [10, 96] did not consider the feature selection problem.

Engineers often disregard feature selection in DRL for manipulation tasks because of the ability of large networks to cope with irrelevant features. Selecting only relevant features, however, will improve learning performances. When transferring a general policy across environments, appropriately selecting features minimises the chances of encountering new states after transfer and eases the generalisation process.

4.2.4 **Training with Uncertainties**

Artificially adding noise is a typical process in reinforcement learning to stimulate learning and encourage an agent to explore. This process intervenes at different levels of the Reinforcement learning paradigm.

During action selection, a reinforcement learning agent chooses the action that maximises the expected sum of rewards. An ϵ-greedy policy [105] forces the agent to choose a random action with a small probability to encourage exploration.

In continuous action spaces, a recurrent method found in the literature involves drawing actions from a Gaussian distribution with a mean at the action output from the algorithm [106]. Similarly, the action can come from an Ornstein-Uhlenbeck process [12].

DRL algorithms can inject noise before action selection through the neural network weights. This process known as parameter noise [107] consists of adding noise to

parameters of the neural nets hence affecting the actions. The changed parameters cause the agent to experience a larger part of the state-space and enables richer behaviours.

Although purposefully adding noise and uncertainties is recurrent in the literature, it is yet unknown if the process would yield similar behaviours when applied to goal features. Measurements of the goal features are inherently uncertain in fine real-world manipulation. Exposing agents to goal uncertainties during training both increases exploration and allows better generalisation. The following section presents a framework that answers some of the shortcomings of the current DRL approaches to a manipulation task.

4.3 **Preliminaries**

This section details the mathematical framework used to train a deep reinforcement learning agent with uncertain goals. It first elaborates on the principles of deep reinforcement learning using the deep deterministic policy gradient algorithm. Then it shows how hindsight experience replay is employed to train an agent for goal-orientated tasks. Finally, it proposes to sample goals from a probability density function and explains how it allows agents to generalise better.

4.3.1 **Deep Deterministic Policy Gradient (DDPG)**

This chapter proposes a DRL approach to fine manipulation tasks. These tasks are formulated as a MDP with observations $s \in \mathbb{R}^n$ and actions $a \in \mathbb{R}^n$ with $n \neq 0$. Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning algorithm that can find a suitable policy allowing a manipulator to complete a task with continuous action and state-spaces.

While trying to complete a task, the robotic manipulator can recursively build a better estimate of the state-action value function using the Bellman equation.

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{4.1}$$

where $Q^{\pi}(s_t, a_t)$ denotes the state-action value for observation s_t and action a_t using policy π, and $r(s_t, a_t)$ is a scalar reward function of s_t and a_t.

A function approximator replaces tabular, discrete expressions of $Q(s, a)$. A non-linear function approximator in the form of a neural network forms the critic $Q(s, a|\theta^Q)$, where θ^Q is the vector of parameters associated with the network. It is updated on a batch of experience using the temporal difference target

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^{\mu})|\theta^Q) \tag{4.2}$$

where $\mu(s_{t+1}|\theta^{\mu})$ denotes the action chosen by the actor network, which is the second part of the algorithm. The actor $a_t = \mu(s_t|\theta^{\mu})$ with parameter vector θ^{μ}

maps the continuous state-space to a continuous action space. The actor policy is updated a step size towards the policy gradient from the experience batch

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i} \tag{4.3}$$

where $\nabla_{\theta^\mu} J$ is the gradient of the actor network or policy and N is the number of transitions sampled in a batch. Both networks are updated using the Poliak averages:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \end{aligned} \tag{4.4}$$

where $0 \leq \tau \leq 1$.

DDPG, just as any reinforcement learning algorithm, faces the dilemma of exploration vs exploitation. It refers to the choice the agent has to make between using its current knowledge of the policy to execute the best action or to explore the action space by choosing any other action but the one maximising the state-action value function. In this framework, Gaussian noise is added after the action selection to ensure the continued exploration of the action space.

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t \tag{4.5}$$

where $\mathcal{N}_t$ is drawn from a Gaussian distribution with mean 0 and variance σ_t^2.

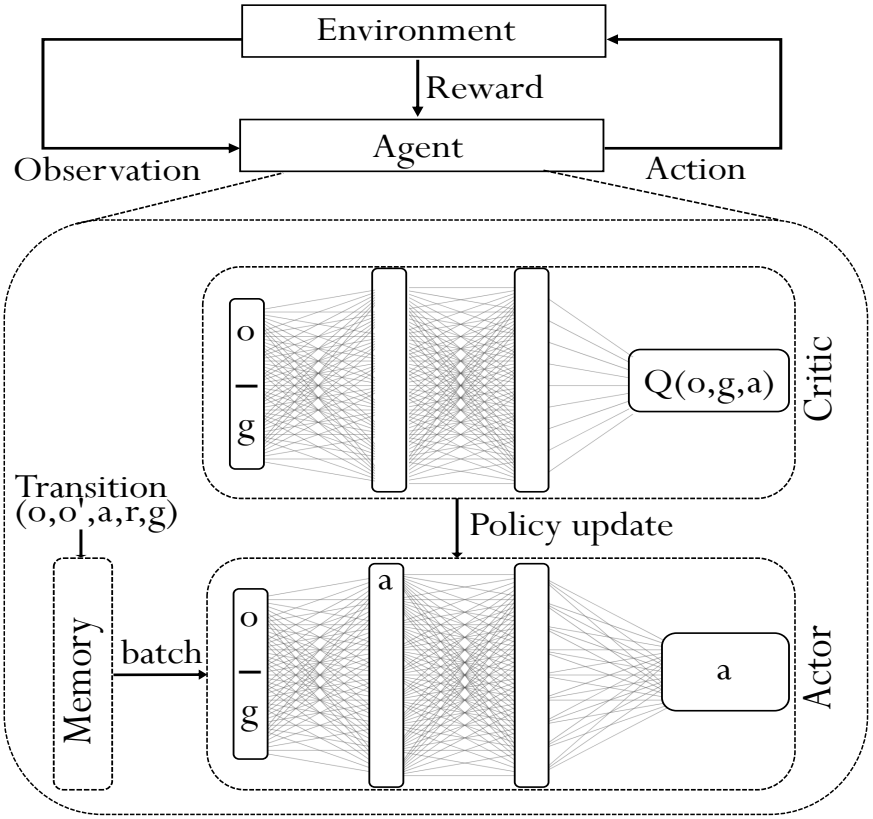

Figure 4.1: Schematic of a DDPG agent

The variance periodically reduces for better stability.

Fig. 4.1 shows the architecture of a DDPG agent within the DRL process. The figure shows the actor and critic network with their respective output. It shows a learning batch sampled from a memory buffer containing all the transitions experienced. It also shows that the action is part of the agent policy since its features add to the first hidden layer of the actor network.

4.3.2 **Hindsight Experience Replay (HER)**

Experience replay [108] is common practice in reinforcement learning to maximise knowledge from experienced agent trajectories. Instead of updating the agent's

parameters based on the transition (s_t, r_t, s'_t, a_t), at each step t, the agent learns from a sampled batch of experience from a memory buffer $\mathcal{B}$.

The framework in this chapter uses Hindsight experience replay [11] for a fine manipulation task. Its fundamental idea is to allow a model-free reinforcement learning agent to leverage experience from unsuccessful episodes by refactoring end of trajectories as goals.

In addition to the experience replay, HER stores transitions that are augmented with the current goal of the agent (s_t, r_t, s'_t, a_t, g) as well as a sampled goal g' for the current trajectory T where $g' \in S \ \forall s$ in S(current episode).

4.3.3 **Uncertain Goals**

The paradigm presented here allows a deep reinforcement learning agent to perform fine manipulation tasks. In such scenarios, accuracy required to accomplish the task may be greater than the accuracy of sensors. However, the literature shows that application of DRL for such tasks do not consider this issue and instead rely on perfect information.

A solution is to train the DRL agent on a goal g from $G \subseteq S$ drawn from a probability distribution reflecting uncertainty from sensory information.

$$g \sim \mathcal{G} \tag{4.6}$$

Both actor and critic rely on the goal as shown in the diagram in Fig. 4.1

The reward is a function of goal elements, carrying information about closeness to the goal, which by consequence relies on knowledge of the goal itself. Both binary reward and potential-based reward formulation use some elements of the goal.

$$r : S \times A \times G \to \mathbb{R} \tag{4.7}$$

By allowing the DRL agent to create a policy over an uncertain goal, the policy maintains performance across environments.

Remark 1: *Let G_{train} and $G^{\sim}_{train}$ be the set of goals encountered during training of a deep learning agent without and with goal uncertainty respectively. Additionally, assuming that $\forall g \in G_{env}, g \sim \mathcal{G}_{sensor}$, and that G is finite. Then $p(G_{train} \cap G_{env} = \emptyset) \geq p(G^{\sim}_{train} \cap G_{env} = \emptyset)$.*

G_{env} is the set of goals drawn from the environment. This goal are drawn from the environment according to to sensor PDF $\mathcal{G}_{sensor}$.

Theorem 4.3.3 explains why training with goal uncertainty may increases agent

performance. Uncertain goals sampled from a probability density function are more likely to resemble the ones drawn from the environment. By consequence, an agent experiencing goals from $G^{\sim}_{train}$ is likely to perform better. This assumption relies on the concepts that neural nets generalise better to features close to the ones they have already experienced.

Fig. 4.2 shows an example of the overlap between $\mathcal{G}_{env}$ and $\mathcal{G}^{\sim}_{train}$. The set of goals drawn from $\mathcal{G}_{env}$ and $\mathcal{G}^{\sim}_{train}$ are likely to be similar whereas certain goals $g \sim \mathcal{G}_{train}$ can be out of the set of goals drawn from the environments completely.

4.3.4 **Algorithm**

Algorithm 3, used to train DRL agents, shows the distinction between s and o where s is a concatenation of the agent's observation and current goal g. The goal in line 9 is initially attributed at random. At this stage, g corresponds to the exact goal for the current episode. In line 10, the sampled goal, g, distances itself from the true goal. An uncertain goal, sampled of a probability density function $\mathcal{G}$, replaces the definite goal which the agent uses for training. Hindsight Experience Replay (HER) stores transitions augmented with the current goal of the agent $(o_t, r_t, o_{t+1}, a_t, g)$ as well as a sampled goal g' for the current trajectory T where $g' \in S \ \forall s$ in $S_{(current-episode)}$.

Algorithm 3: HER with uncertain goals

Initialise $Q(s, a|\theta^Q)$ and $\mu(s|\theta^\mu)$;
Initialise Q' and μ';
Initialise Replay buffer $\mathcal{B}$
Place holder $s = o||g$ # || denotes concatenation
$N \leftarrow batch\_size$
repeat
    Initialise action noise $\mathcal{N}$
    $o \leftarrow getCurrentObservation()$
    $g \leftarrow sampleGoal(G)$ # change goal for each episode
    $g \sim \mathcal{G}$ # sample a goal from pdf $\mathcal{G}$
    repeat
        $a_t \leftarrow \mu(s|\theta^\mu) + \mathcal{N}$
        $r_t, o_{t+1} \leftarrow takeAction(a_t)$
        $storeTransition(o_t, a_t, r_t, o_{t+1}, g)$ in $\mathcal{B}$
        g' $\leftarrow sampleGoal(G := S)$
        $storeTransition(o_t, a_t, r_t, o_{t+1}, g')$ in $\mathcal{B}$
        if $size(\mathcal{B}) \geq N$ **then**
            $batch \leftarrow sample(\mathcal{B}, batch\_size)$
            $y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
            Update critic with loss:
            $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
            Update actor with gradient:
            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
            Update target network:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q^{'}}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu^{'}}$
        $o \leftarrow o_{t+1}$
    until *until terminal state*;
until;

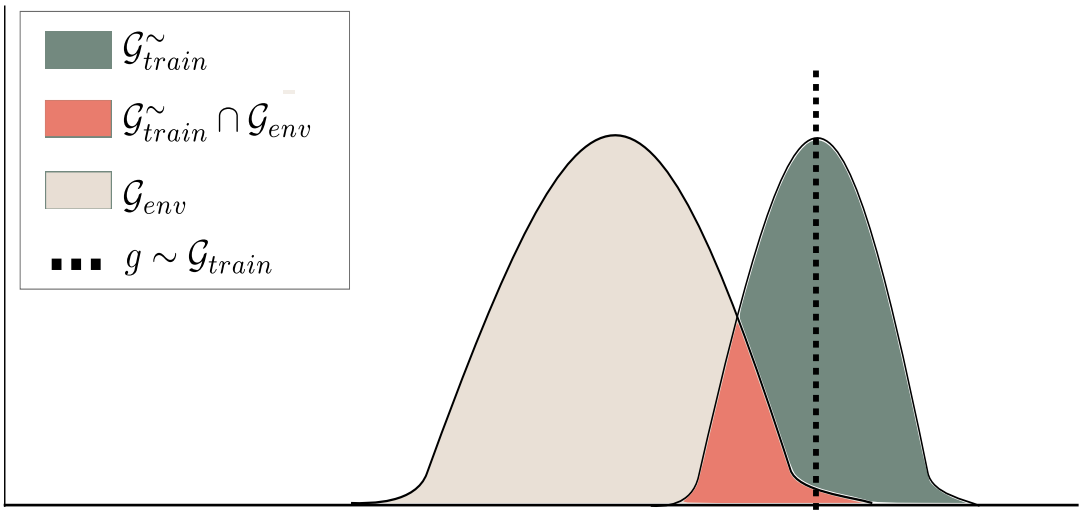

Figure 4.2: Schematic of the training goal and environment goal intersection

4.4 **Training a deep reinforcement learning agent for a fine manipulation task**

This section details a methodology to train a deep learning agent for a fine manipulation task. First, it proposes a set of guidelines for state-space formulation as it is the basis of any DRL agent. Then, it presents a framework for progressive training that encourages agents to converge faster to an adequate policy. In this section, the phrases "feature selection" and "state-space formulation" are interchangeable.

4.4.1 **Feature selection**

Feature selection affects learning performance. Algorithm 3 relies on a methodical selection of features to perform optimally. By training with uncertainty, the goals observed during training are more likely to resemble the ones observed in other

environments (Theorem 4.3.3), however, the larger the number of features in the state-space, the less likely two states are to resemble each other. By consequence, it is crucial to choose the minimum amount of features to represent the goal.

The literature provides no information about what model-free learning agents need to know (features) to complete a task. There are no categories to group features, no distinction between environment and manipulator features and no indication of how they may affect task completion. In this work, state-space features belong to one of three categories: goals, constraints and controlled quantities. By including features belonging to these categories and pertinent to the task in the state-space model, the agent has sufficient information to perform the task entirely.

An agent's goal is a subset of its observations; hence the whole state-space must also be selected carefully and kept to a minimum. Although there are several approaches to determining relevant features in neural nets, there exists little literature about selecting features in the context of model-free deep reinforcement learning. To this extent, the following set of guidelines explains how to best formulate the state-space (or features) for a DRL agent performing a fine manipulation task before training with uncertainty:

1. Goal definition

   The first step to solve a DRL manipulation task is to define its goal. It pertains to both the robotic manipulator and its workspace. The success of specific manipulation tasks can be unrelated to the manipulator's state,

such as pick-and-place tasks. In contrast, others can correspond precisely to the manipulator's state, such as collision avoidance tasks. This distinction must be made earlier-on in the state-space formulation process. In this first step, it is enough to complete the sentence: the task is successful if... .

2. Constraints definition

   The constraints of a manipulation task are all aspects that can cause the task to fail. Contrary to the first step, completing the following sentence helps to define the constraints: the task has failed if... . They can be physical constraints, such as a collision, force or boundary. They can also be conceptual, such as time, number of iterations or uncertainties. Constraints inform the DRL agent what not to do.

3. Goal formulation

   In this step, features are extracted from the manipulator and its workspace that are indicative of task completion. Each of these features needs an associated scalar value corresponding to its desired state. These pairs of features are the minimum required for the DRL agent to perform the task and will be part of the state-space. This step is not only essential for obtaining the state space but will also serve in defining the rewards of the task. The goal may include information about an object in the environment or about the robotic manipulator. Defining the manipulation task as dynamic or static, or haptic or contact-less provides further insights into the extraction of goal features. The outcome of this step is a $2n$-dimensional vector where n is the

number of goal features.

4. Constraints formulation

   In this step, features are extracted from the manipulator and its workspace that are indicative of task failure. Goal and constraint features can overlap. Constraint features do not need an associated scalar value or threshold; however, they must be quantifiable. These quantities may participate in computing the reward during the manipulation task. The robotic manipulator or its peripherals must have the mechanism to extract a scalar value for each of the constraint features.

5. State-space formulation

   Most elements required to formulate the state-space model are by now identified; two of three categories are complete. The last addition to the state-space model is the elements controlled by the DRL agent. In continuous control problems, these correspond to positions or velocities of the joints or end-effector, for example. These allow the agent to establish a stronger relationship between features and actions. These features may again overlap with both goals and constraints.

   Finally, concatenating goals, constraints, and controlled elements is the last step to express the state-space. Such a formulation forms a strong basis for a DRL agent to learn a manipulation task while maintaining the number of features to a minimum. It facilitates policy transfer to different environments, notably when training agents with goal uncertainties.

Manual pruning can further ensure that the state-space model contains only relevant features. Some of the features carry meaningful values for only a small portion of an episode making automated feature selection a challenging process in DRL. For example, during a Peg-in-Hole task, force and torque values might only be non-zero while the peg is in contact with the surface. This shorter active period means that they might carry little weight in the neural nets. By considering such factors, features can be pruned manually from the input vector after learning, based on the portion of the total weight they represent in the first fully-connected layer of an actor network. It can be calculated as follows:

$$weight\% = \frac{\sum_{i=0}^{i=N} \theta_i^{\mu}}{\sum_{j=0}^{j=M} \sum_{i=0}^{i=N} \theta_{i\,j}^{\mu}} \times 100 \tag{4.8}$$

where N is the number of neurons in the first hidden layer, M is the number of input features and θ_{ij}^{μ} is the weights of the i_{th} neuron of the j_{th} input feature. Features carrying less weight for their active period can be pruned from the input feature vector as they may not affect how the agent acts. Training can then be re-started and performances compared.

Selecting the minimum amount of features required to perform a task ensures better learning performance and that states are drawn from a similar distribution no matter what environment the agent performs in. An agent that learned a policy in a simulation from the hundred-thousands of features from an image has fewer chances of performing well in the real world than an agent that learned from a minimalist representation of the same problem. Additionally, a consistent way

of defining state-space models for fine manipulation task provides a means to benchmark generic applications such as PiH.

4.4.2 **Training methodology**

Hindsight experience replay allows an agent to learn from a sparse reward. Traditionally, the agent received a binary reward (-1 or 0) based on a user threshold meaning that the agent had reached a goal. For fine manipulation tasks, where goals are challenging to reach, using binary rewards does not allow the agent to create an adequate policy. Additionally, the real goal may be out of reach of the robot and the learning agent when training with goal uncertainties. The following two methods address this problem:

A potential-based reward reflecting distance to the sampled goal ensures that the agent can maximise the reward. When using potential rewards, one must ensure that its maximisation corresponds to the optimisation of the robot's behaviour. Often the agent optimises the reward in an unpredictable way and so it does not create the desired behaviour.

Fine manipulation problems have rare spikes in the rewards from which the agent learns. One solution to this problem is to allow the agent to complete more accessible versions of the task in order to create base policies that can then be improved on for harder versions of the task. Such a process increases the chances of the agent to develop a successful policy. This technique can be used both

in simulation and on real hardware. Simulation offers other advantages such as automatically scaling the task difficulty based on the agents' success rate. The larger the success rate, the more difficult the task becomes.

Progressive training is depicted in Fig. 4.3. The process has three user-defined parameters. The first one, the success threshold, determines when the agent needs to switch to a more complex version of the task. It indicates that the DRL agent is capable of completing the current version of the problem often enough. The second parameter is the step/success ratio. It indicates, on the other hand, when the task is too challenging for the agent and must be made simpler. The parameter represents the number of algorithm steps the agent can take without reaching the success threshold. If the agent exceeds the step/success ratio too many times, the DRL agent hyper-parameters must be revised. Finally, the last parameter required from a user is the difficulty target. It indicates the last level of difficulty the DRL agent must be able to complete. Once reached, the agent has been trained fully for the task.

The literature offers little solutions to hyperparameter tuning for a deep model-free reinforcement learning algorithm. A common way to select these parameters is to copy ones from a similar solved problem. Trial-and-error is another way to tune hyper-parameters but is a lengthy process for large tasks. Progressive training encourages to set metrics to assess agent performance early during training. It is often unknown whether an agent will start succeeding after 1000 training steps or $10e^6$ training steps. Progressive training provides an early estimate of the chances

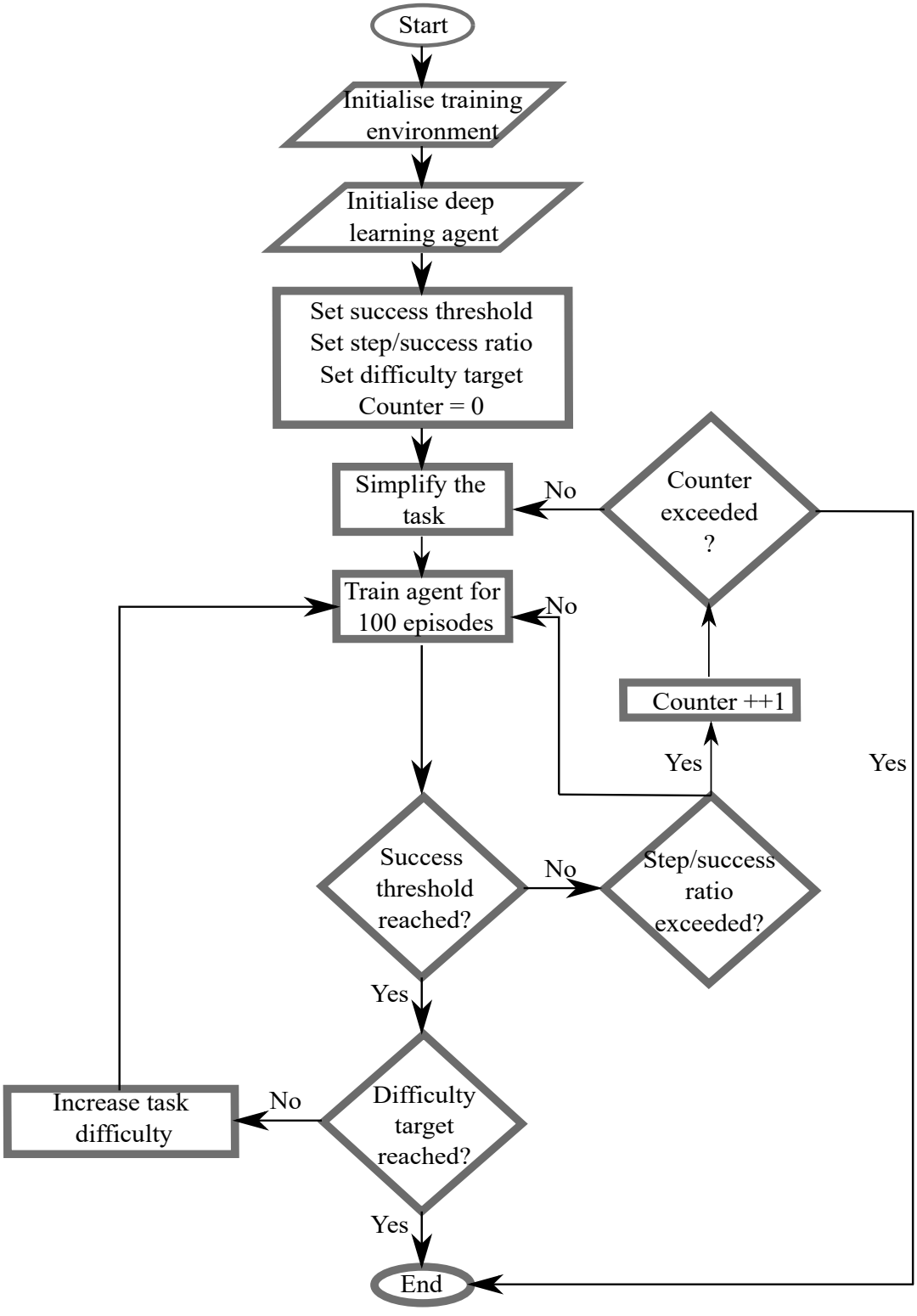

Figure 4.3: Flowchart of progressive training

of agent success and whether hyper-parameter tuning is necessary.

4.4.3 **Filtering**

Selected features are crucial for learning and completing the task. However, both actor's and critic's neural nets must be able to make sense of the data coming through these features during learning. Noisy features can be rejected by the neural nets and lead to inadequate policies and low success rates. Haptic or dynamic tasks may present significant noise in force/torque and acceleration readings, respectively. Low-pass filtering in these cases is often a solution to improve the quality of the data. It, however, can cause lagging data which is problematic for highly dynamic tasks.

4.4.4 **Goal sampling**

A DRL agent learns in a simulation with goals sampled from a probability distribution. This framework aims to facilitate the transfer of policy learned in one environment to another since the agent would learn how to behave with uncertain goals. Goals are samples from different probability density functions such as uniform, Gaussian or bi-modal each reflecting uncertainties from different types of sensors or sensory data processes. According to the framework, any of these distributions have a larger intersection with the true distribution from the environment and will lead to higher success rates of the learning agent. The probability density functions must reflect uncertainties at each difficulty level of the task. Larger deviation from the true goal should apply with lower difficulty versions of the task.

4.4.5 **Discussion**

The framework presented in this chapter synthesises methods and techniques often implicitly used to train a DRL agent for manipulation tasks. Every architect of a DRL problem must use the guidelines proposed to formulate the state-space features before the training begins. This process is often undocumented throughout the literature; however, it requires more intricacies and insights into the problem than one might consider.

Progressive training appears as a natural way to learn and is not unlike how children learn to assimilate specific behaviours. First, a tool is used for a simple problem to build the basis of behaviour to solve this problem; then the same device is used for increasingly challenging versions of the problem. This idea is already present in the DRL context. HER teaches a DRL agent to reach more and more challenging targets by first teaching it to reach targets that it has already achieved. It does not require modifying the agent's environment. Progressive training, on the other hand, proposes to build knowledge by changing the environment itself.

Training with goal uncertainties allows the agent to build a policy that will transfer to another environment where the goals come from an unknown probability density function. It is not necessary to train an agent with uncertainties if used in the same environment because the policies developed are based on the environment themselves. Training with uncertainties is especially relevant to manipulation where training must often be done in a different environment, i.e. in a simulation.

Training is impractical and unsafe on real hardware due to time constraints and potential hardware damage.

Advanced simulation software offers the possibility to accurately replicate a fine manipulation scenario, including all sensors necessary in the real-world task and their respective uncertainties. In such cases, all sources of uncertainties will be similar in the simulation and the real task. Nonetheless, time and effort to create such accurate simulation is significant; more importantly, in such cases, the application remains sensor dependant. The framework in the chapter for training with goal uncertainties makes no such assumption and is by consequence, transferable.

4.5 **Conclusion**

This chapter proposes a framework to solve fine manipulation tasks through DRL under goal uncertainty. It shows how training an agent with uncertain goals allows the transfer of a policy to different environments by enlarging the intersection $G_{train}^{\sim} \cap G_{env}$. A set of guidelines explains how to select features for model-free DRL manipulation tasks. Besides, the framework proposes a methodology for progressive training consisting of increasing the difficulty of the task until acceptable behaviour from the agent. Finally, the framework suggests filtering as a technique to strengthen the relationship between noisy input features and actions. Drawing goals from a probability density function that includes the true goal will

lead to a higher success rate of the deep learning agent. In the next chapter, a DRL agent for a Peg-in-Hole task uses the framework presented here. It allows the deep learning agent trained with uncertain goals to achieve higher success rates when confronted with goals drawn from a different probability density function.

Chapter 5: Peg-in-Hole Using DRL and Uncertain Goal Training

Peg-in-Hole is a fine manipulation task where a peg is inserted into a hole using a robotic manipulator. Manually programmed search patterns and manipulator compliance are traditional solutions to this problem. DRL offers a solution to solve this problem with little or no-pre-programming. DRL however, relies on a definite goal which is unrealistic and can cause agents to perform poorly.

In this chapter, the framework in Chapter 4 was applied to a round Peg-in-Hole and square Peg-in-Hole fine manipulation task. In both cases, the framework allowed the training of agents that performed better across a range of environments. It shows that the framework could be applied to other manipulation tasks, eliminating the need to retrieve perfect goals.

5.1 Introduction

DRL showed applications to the Peg-in-Hole task. It is not possible to train a DRL agent entirely on real hardware since training periods are too long and robotic manipulators are subject to damage. Agents instead learn the task in a simulation and policies are then transferred to real hardware. In most training environments, however, agents rely on definite goals for the task which do not have the same

accuracy on real hardware.

The framework in Chapter 4 proposed to train a DRL agent with goal uncertainty and aimed to facilitate the transfer of policies to other environments. In this chapter, the framework was applied to a Peg-in-Hole manipulation task. The agent used several types of goal uncertainties during training. The policies were then cross-validated in a range of environments. The approach showed that the framework can be applied to different types of goal uncertainties and manipulation tasks.

The chapter is organised as follows: Section 5.2 describes both versions of the Peg-in-Hole task. Section 5.3 shows how the framework applies to both versions of the task. Section 5.4 and Section 5.5 present the results obtained from training and cross-validating the policies in a range of environments for the round and square PiH task respectively. Section 5.6 discusses the implications of the results. Finally, Section 5.7 concludes this chapter.

5.2 **Problem Statement**

This section describes the two versions of the PiH problem in this chapter. In both versions, a robotic manipulator with 6 degrees of freedom used a peg as the end-effector. In the first version, the peg and hole were circular; it is referred to throughout this chapter as the round Peg-in-Hole. The goal of the task consisted

of inserting the peg at a depth of $3cm$. There was no *a-priory* knowledge about the task except the orientation of the hole. During insertion, the peg could translate in x, y and z. The clearance between peg and hole was $0.5mm$.

In the second version of the task, referred to as the square Peg-in-Hole, the peg and hole were square-shaped. The goal was again to insert it at a depth of $3cm$. The orientation of the hole in pitch and roll were known, but the peg had to be translated in x, y, z and oriented in yaw to fit into the hole. The desired clearance was also $0.5mm$.

Square PiH is a different, more complex manipulation tasks than the round PiH. In both cases, the agent had 500 steps per episode to complete the task. The peg could move by a maximum of $0.5mm$ per action. By consequence, an episode started with the peg at a maximum of $3.5cm$ away from the goal.

5.3 **Method**

The framework was used to obtain policies for both versions of the Peg-in-Hole problem. The first part of the framework consisted of selecting the state-space features according to the set of guidelines proposed in Section 4.4.1. The second part was to prune the state-space further by analysing the feature/weight proportion in the first layer of the actor's network. Finally, the last part showed how three different forms of uncertainties were used to train the agents. The framework was

applied to both versions of the problem simultaneously throughout this section.

5.3.1 **State-space formulation**

The guidelines given in 4.4.1 were used to formulate the agents state-space.

1. Goal definition

   Round Peg-in-Hole & square Peg-in-Hole: In both versions of the Peg-in-Hole task the DRL agent aimed at inserting the peg in the hole at the desired depth. An episode was successful if the peg was inserted deep enough inside the hole.

2. Constraints definition

   Round Peg-in-Hole & Square Peg-in-Hole: Both versions of the Peg-in-Hole insertion had failed if the peg moved too far from the hole. In such cases, the peg may collide with the environment. The task was also unsuccessful if the peg contact with the hole surface exercised too much force on the robotic manipulator.

3. Goal formulation

   Round Peg-in-Hole: Success depended on the peg's depth inside the hole; hence its displacement in z and position in x and y were observed. The scalar goal value for x and y were the positions of the hole about the manipulator's base while z was a desired depth below the perceived hole surface.

Square Peg-in-Hole: Success depended on the peg's depth inside the hole; hence its displacement in z and position in x and y were observed − additionally, the agent required to know the orientation of the peg, θ, in yaw. The task was only successful if the peg was in the hole at the perceived hole orientation.

4. Constraints formulation

   Round Peg-in-Hole & Square Peg-in-Hole: Excessive force and torque could cause damage to the robotic manipulator; hence they had to be observed during the insertion task. These quantities were the forces and moments x, y and z at the manipulator's wrist.

5. State-space formulation

   Round Peg-in-Hole: In this scenario, the end-effector was directly controlled by the agent hence affecting its position and velocity. These were the controlled quantities. The full state-space was formulated as the concatenation of the peg's position x, y and z, the goal position, g_x, g_y and g_z, the peg velocities $\dot{x}, \dot{y}$ and $\dot{z}$, and the forces and torques applied to the manipulator $F_x, F_y, F_z, \mathcal{T}_x, \mathcal{T}_y, \mathcal{T}_z$.

   Square Peg-in-Hole: The state-space model was similar to the round Peg-in-Hole model, with the addition of the θ as a controlled quantity. It overlapped with the goal formulation for this scenario. The full state-space was the concatenation of the peg's position and orientation x, y, z and θ the

goal position and orientation, g_x, g_y, g_z and g_θ, the peg velocities $\dot{x}, \dot{y}$ and $\dot{z}$, and the forces and torques applied to the manipulator $F_x, F_y, F_z, \mathcal{T}_x, \mathcal{T}_y, \mathcal{T}_z$.

The guidelines offered an initial formulation of the state-space for both versions of the problem. An initial learning session was conducted based on this formulation and further pruning applied to the state-space based on the techniques in Section 4.4.1

5.3.2 **State-space pruning**

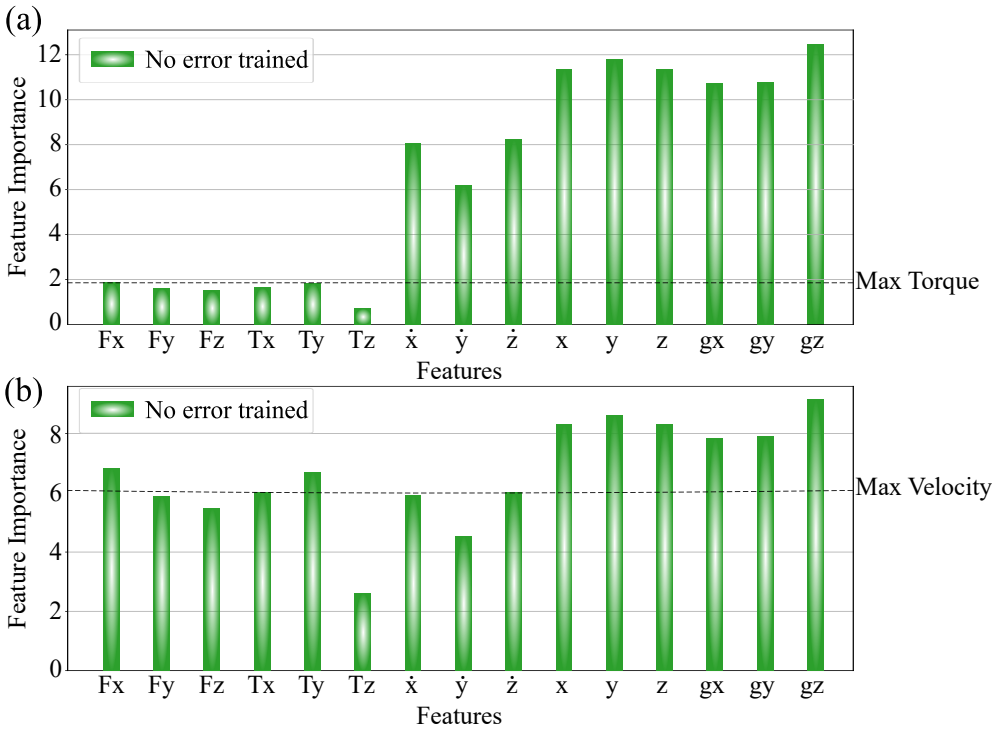

Figure 5.1: Input features importance for the actor network, (a) un-normalised; (b) normalised

The state-space model of an agent for a fine manipulation task can be reduced by analysing the weight distribution in the first layer of the actor's network. The magnitude of the weights associated with a specific feature relative to the total weight correlates to the importance of that feature.

An agent was trained on the round Peg-in-Hole task on a simple instance of the problem (peg/hole clearance = $1.5mm$). The weights distribution for each input feature are shown in Figure 5.1(a). On the other hand, 5.1(b) shows the weight distribution normalised to the active period of the features during the task. The figures show that velocity features were slightly less important than other features to the first layer of the actor's network. A velocity feature represented at most 6% of the total weight. Besides, although PiH was qualified as a dynamic task, the peg could only move at a maximum velocity of $0.22mm/s$ making it almost static. By consequence, velocity features were pruned from the state-space of both the round Peg-in-Hole and square Peg-in-Hole problem.

The state-space vectors were reduced to

$$S = [x, y, z, g_x, g_y, g_z, F_x, F_y, F_z, \mathcal{T}_x, \mathcal{T}_y, \mathcal{T}_z]$$

and

$$S = [x, y, z, \theta, g_x, g_y, g_z, g_\theta, F_x, F_y, F_z, \mathcal{T}_x, \mathcal{T}_y, \mathcal{T}_z]$$

for the round Peg-in-Hole and square Peg-in-Hole agents respectively, where (x, y, z, θ) were the position and orientation in yaw of the peg, $(g_x, g_y, g_z, g_\theta)$ were the goal position and orientation of the peg and $(F_x, F_y, F_z, \mathcal{T}_x, \mathcal{T}_y, \mathcal{T}_z)$ were the forces and torques applied to the peg.

5.3.3 **Progressive Training with Goal Uncertainties**

The state-space vectors of the agents for both round and square Peg-in-Hole were determined using the framework established in Chapter 4. To further follow the framework, the agents were trained progressively with an increasingly harder version of the problem while subject to goal uncertainties.

5.3.3.1 **Training Setup**

All agents learned to insert the peg in a simulated Mujoco® environment. The simulated manipulator was a 6 degree of freedom $UR5$ robotic arm with round or square peg rigidly attached to the last link. The simulation and DRL algorithms were executed on a Lenovo $P920$ ThinkStation with dual $P6000$ GPUs of $24GB$ each and a Intel® Xeon® $2.3GHz$ CPU with 24 cores.

Table 5.1 shows the hyper-parameters used by both the round and square Peg-in-Hole agents during training. All hyper-parameters were determined based on state-of-the-art best practices or through trial and error advised by progressive training. An episode is the number of steps an agent can take to maximise the reward of the task. In the case of Peg-in-Hole, it can be challenging to select the length of an episode. For both Peg-in-Hole versions, the peg was manually jogged and inserted repeatedly to establish a benchmark for the number of steps in an episode. This resulted in a 500-step episode for both versions of the task.

Table 5.1: Hyper-parameters used to train both square and round PiH DRL agents

Hyper-parameter	**Value**
Network	[64,64]
Steps of actor learning rate	$1e^{-3}/1e^{-3}/1e^{-4}/1e^{-4}$
Steps of critic learning rate	$1e^{-3}/1e^{-4}/1e^{-4}/1e^{-5}$
Gamma	0.95
Tau	0.1
Sample goals	4
Random exploration	0.1
Batch size	512
Buffer size	$1e^{6}$
Policy	MlpPolicy
Activation	tanh
Normalised observation	True
Noise	Parameters noise

5.3.3.2 **Progressive Training**

Progressive training allowed the agents to create policies in a realistic time-frame. The task can be made harder by physically modifying the environment the agent is subjected to. For the Peg-in-Hole tasks in this chapter, the peg/hole clearance was adjusted to increase or decrease task difficulty. A larger peg/hole clearance corresponded to an easier problem to solve. The difficulty was increased by reducing the peg/hole clearance by $1mm$ from $3.5mm$ to $0.5mm$ each time the agent reached a 100% insertion rate.

Reducing the learning rate of the agent at each difficulty step was an important aspect of progressive training. In both versions of the problem, actor and critic network learning rate were decreased at all difficulty steps. Table 5.1 shows the learning rates.

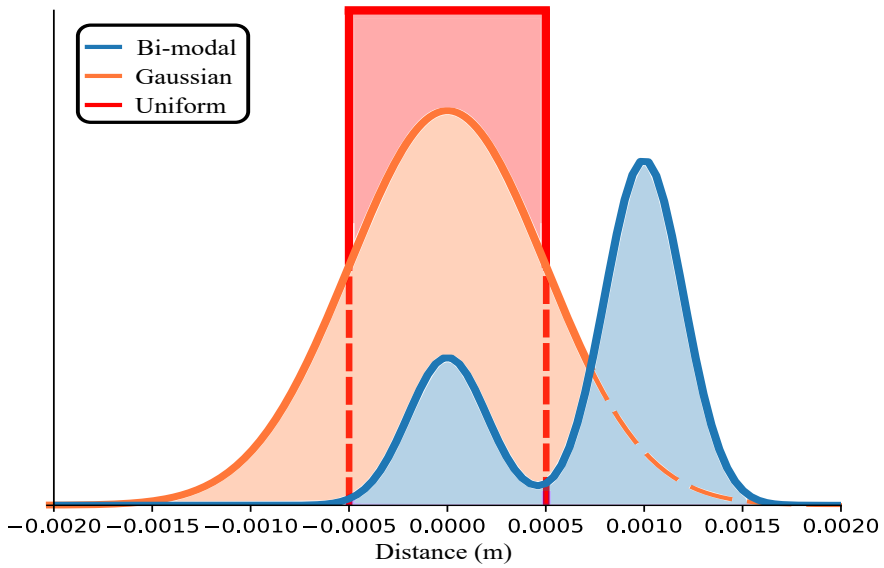

Figure 5.2: Probability density functions used to train the DRL agent with goal uncertainties.

5.3.3.3 **Training with Uncertain Goals**

Three types of uncertainty affected the goal. They may be the result of noisy sensor readings or of various processing algorithms used to recover the hole location and orientation. They were uniform, Gaussian and bi-modal uncertainties. An agent learned with goals drawn from probability density functions reflecting each of these uncertainties as well as from definite goals. A total of six agents were trained, four for round PiH and two for square PiH.

Fig. 5.2 shows the probability density functions used to draw goals in the last difficulty step (0.5mm peg/hole clearance). 0 is the definite goal for each of the uncertainty types. All position elements of the goal were subjected to the same type of uncertainty during training. The orientation goal for the square PiH was subjected to smaller uncertainties.

Rewards were generated based on the uncertain goal, which could physically be out of reach of the peg. The framework used potential rewards to avoid this issue. In the round PiH scenario, the reward was calculated based on the distance to the uncertain goal.

$$r = -\sqrt{\delta x^2 + \delta y^2 + \delta z^2} \tag{5.1}$$

In the square PiH problem, a similar reward signal was chosen with addition of a small part of the reward for orientation

$$r = -(\sqrt{\delta x^2 + \delta y^2 + \delta z^2} + 0.1 \times \delta\theta) \tag{5.2}$$

where $(\delta x, \delta y, \delta z)$ and $\delta\theta$ was the distance between the current peg position and orientation, and the uncertain goal position and orientation in (x, y, z, θ) respectively.

5.3.3.4 **Filtering**

Finally, features prone to noisy readings were filtered using a low-pass filter. These features consisted of the force and torque readings at the manipulator's wrist for both PiH versions. The low-pass filter is given by:

$$x_{fltd} = \frac{x_{t-2} + 2x_{t-1} + x - (p^2 - 1.414p + 1)x_{t-1} - (-2p^2 + 2)x}{(1 + p^2 + 1.414p)} \tag{5.3}$$

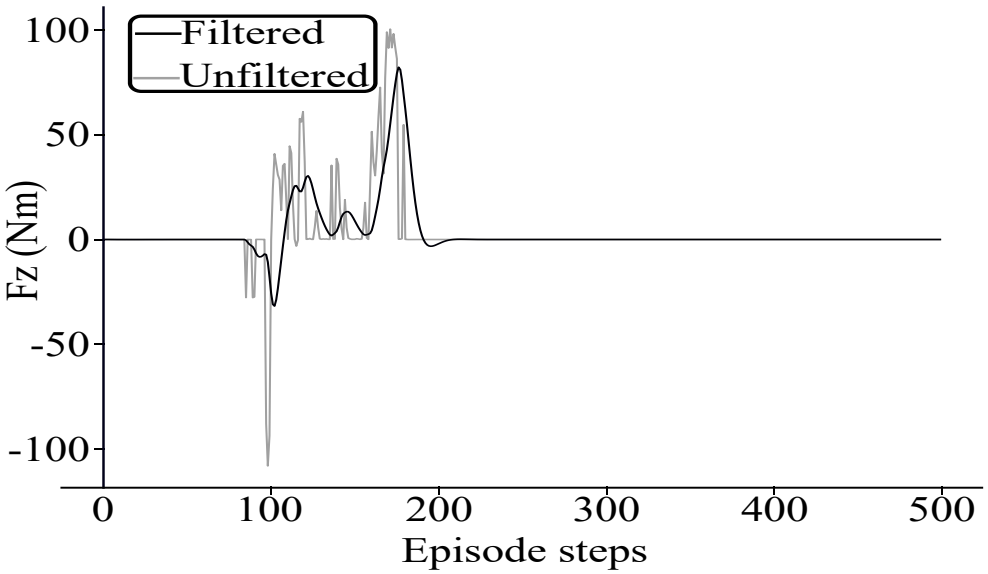

Figure 5.3: Graphs of the filtered and unfiltered force readings

where x_{fltd} was the filtered measurement, x, x_{t-1}, x_{t-2} were the values at time $t, t-1$ and $t-2$ respectively and p was a parameters that was chosen experimentally to be 4.0. This filter was applied to each of the force and torque readings. Fig. 5.3 shows the force reading in z with and without the low-pass filter. Most high frequency fluctuations were filtered out. This strengthened the relationship between these features and the action of the agents.

5.4 **Round PiH: Results and Validation**

This section shows the results from training the four different agents for the round PiH problem. Fig. 5.4 shows the simulation environment for policy validation. First, it shows that the selected state-space model, in combination with progressive training and filtering, allowed each agent to achieve 100% success rate at $0.5mm$ peg/hole clearance. Then it shows the cross-validation of the policies created

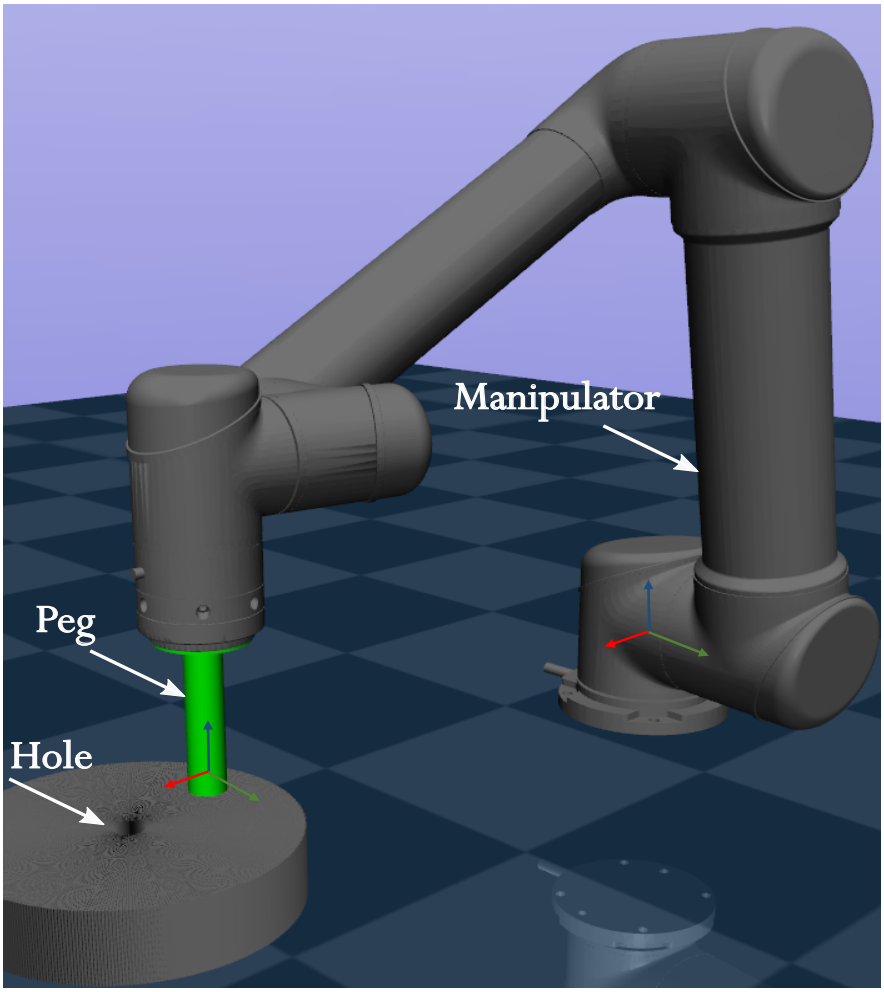

Figure 5.4: Simulation of the round Peg-in-Hole task

by each of the agents. In every case, the policy created with goal uncertainty, irrelevant of the type of uncertainty, was able to achieve a higher success rate. The action space was a translation of the end-effector between 0 and $0.5mm$ in x, y and z.

5.4.1 **Results**

All 4 of the DRL agents learned to complete the task 100% of the time in their respective environment. The first agent learned with definite goals, the second

with uniform uncertainties in the goal, the third with Gaussian uncertainties and the fourth with bi-modal uncertainties.

Figure 5.5 shows the rewards collected by each of the agents through the last of the progressive training steps. The training was interrupted as soon as an agent reached a 100% success rate. This technique allowed each agent to create a successful policy for the final desired peg/hole clearance of $0.5mm$.

At the end of each training session, each agent achieved a reward between -13.2 and -8 after less than $1.1e^6$ training steps. The agent trained with no error in the goal reached 100% success rate sooner than the other agents, on the other hand, they were able to collect larger rewards which correlated to richer behaviour and better convergence of the policies. The reward function did not penalise the number of steps to reach the goal; This meant that an agent should have developed a converging policy rather than one that led to 100% insertion rate faster .

The input features importance are shown in Fig. 5.6. The bar graph shows the percentage of the weights each input feature represents with respect to the first fully connected layer of the actor network. It was calculated as follows:

$$weight\% = \frac{\sum_{i=0}^{i=N} \theta_i^\mu}{\sum_{j=0}^{j=M} \sum_{i=0}^{i=N} \theta_{i\,j}^\mu} \times 100$$

where N was the number of neurons in the first hidden layer, M was the number

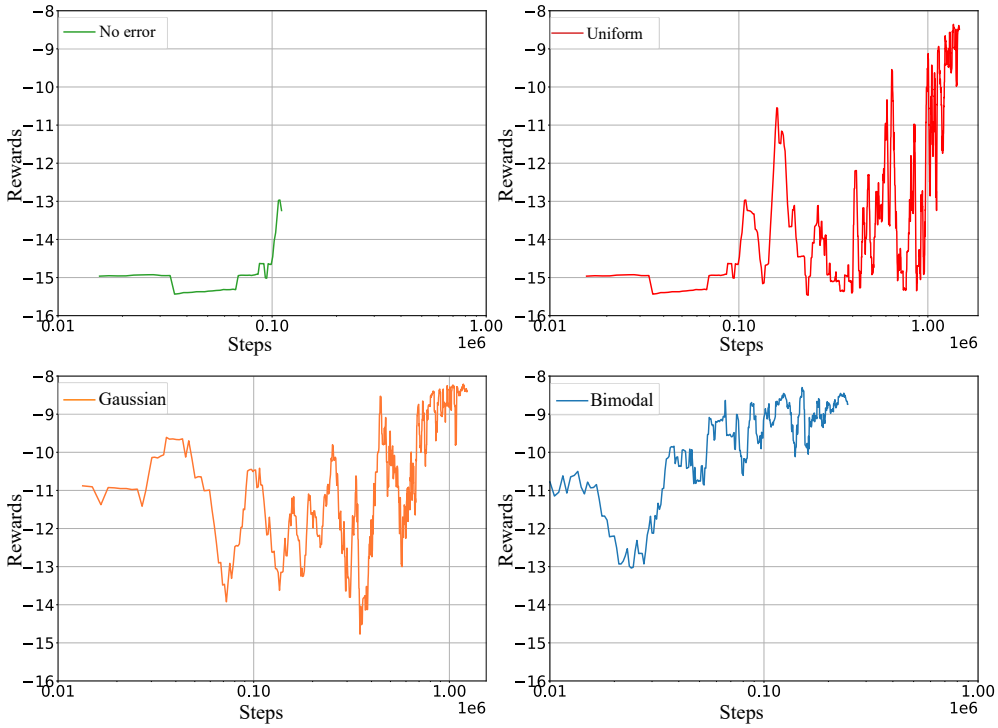

Figure 5.5: Rewards collected by each of the four agents trained for the round PiH with 0.5mm peg/hole clearance

of input features and θ_{ij}^{μ} was the weights of the i^{th} neuron of the j^{th} input feature. The graphs show that the feature representing the peg's location and goal location affect the action significantly more than the force-torque features. However, there were no drastic changes in the distribution of the weights between the different agents. Additionally, the figure shows that some weight was attributed to each of the features selected for learning, showing that the network accepted them. When scaling the weights to the features active period in Fig. 5.6(b), the force/torque features were almost as important to the actor's network for selecting the right action.

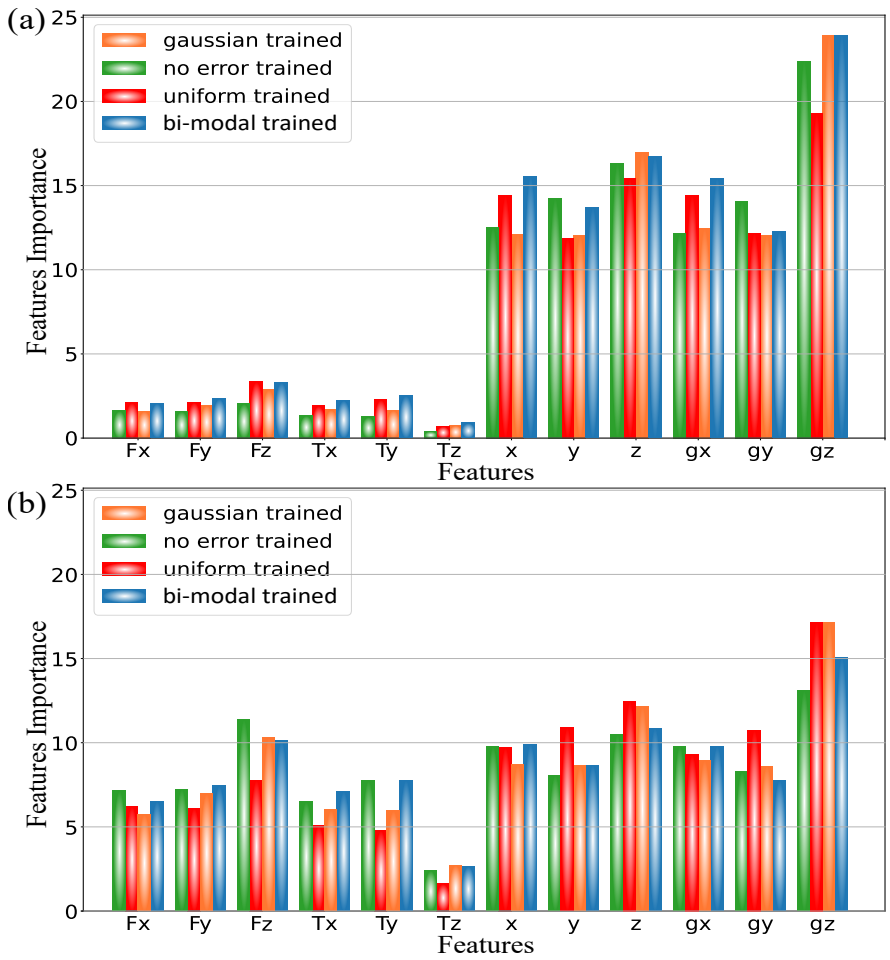

Figure 5.6: Bar graph showing the percentage of weights attributed to each element of the input features in the first layer of the actor network for each trained agent at 0.5mm clearance and 100% success rate. (a) un-normalised, (b) normalised

5.4.2 **Simulated Validation**

To confirm the validity of training with uncertainty, the agents trained with uncertain goals had to insert the peg more often for a range of environments; by consequence, they must have different policies and be more exploratory.

The policies developed were cross-validated in a range of environments with different types of uncertainties. In each environment, the policies were subject to larger and larger uncertainties respective to this environment. In the uniform

environment, each element of the goal were samples of a uniform distribution with a larger and larger interval. In the Gaussian environment, the standard deviation increased from 0 to $5mm$. Finally, in the bi-modal environment, the distance between the two modes became larger and larger.

Fig. 5.7 shows the success rate of each trained agent in each type of uncertainty considered. The graphs show that in all cases, the agent trained with exact goals did not perform as well as the agent trained with uncertainties. The $2mm$ uncertainty mark is a reference point of the figures. In Fig.5.7(a), (b) and (c), the agent trained with a definite goal has a lower insertion rate of at least 30%,18% and 60% respectively at the reference mark. In the environment with bi-modal uncertainties, the graphs show that the agent trained with bi-modal uncertainties performed up to 70% better than the agent trained with no uncertainty.

Figure 5.8 shows the trajectories of the peg for the four trained agents. The trajectories differed for each trained agents showing that they developed a different policy. Figure 5.8 also shows that the agent trained with no uncertainty was unable to insert the peg in an environment with $2mm$ standard deviation for the goal when all the other agents inserted the peg successfully. It is reflected by the trajectory stagnating at a given height and only moving in the (x, y) plane.

Finally, an agent would conclusively be more exploratory if it displayed such behaviours in the trajectory of the peg in the (x, y) plane. Figure 5.9 shows graphs of these trajectories for all 4 trained agents. It shows that for the agents trained

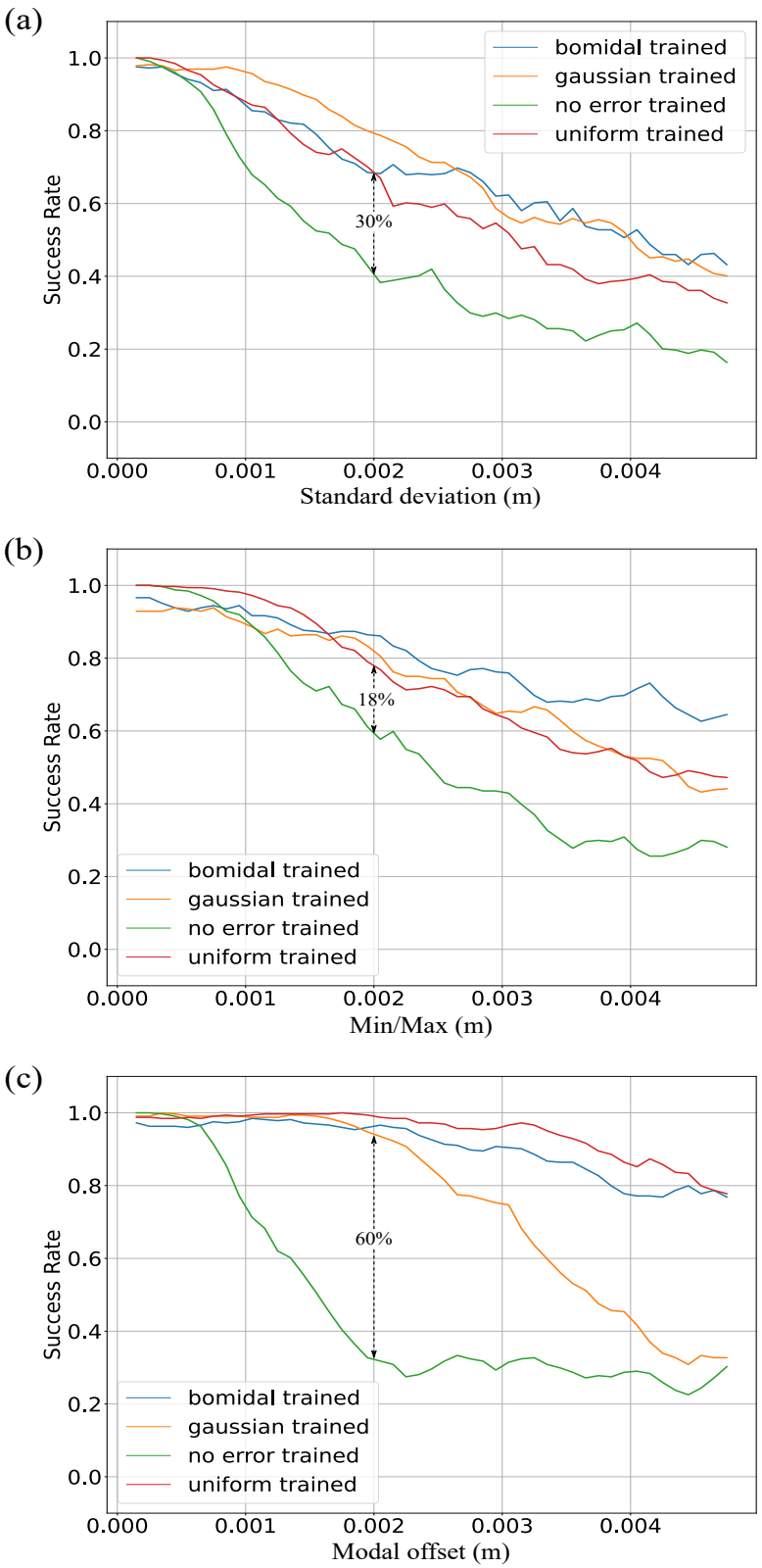

Figure 5.7: Insertion rate vs goal uncertainty for each agent trained for the round PiH task. Each value is an average over 80 episodes. Goals from a (a) Gaussian, (b) uniform and (c) bimodal probability density function.

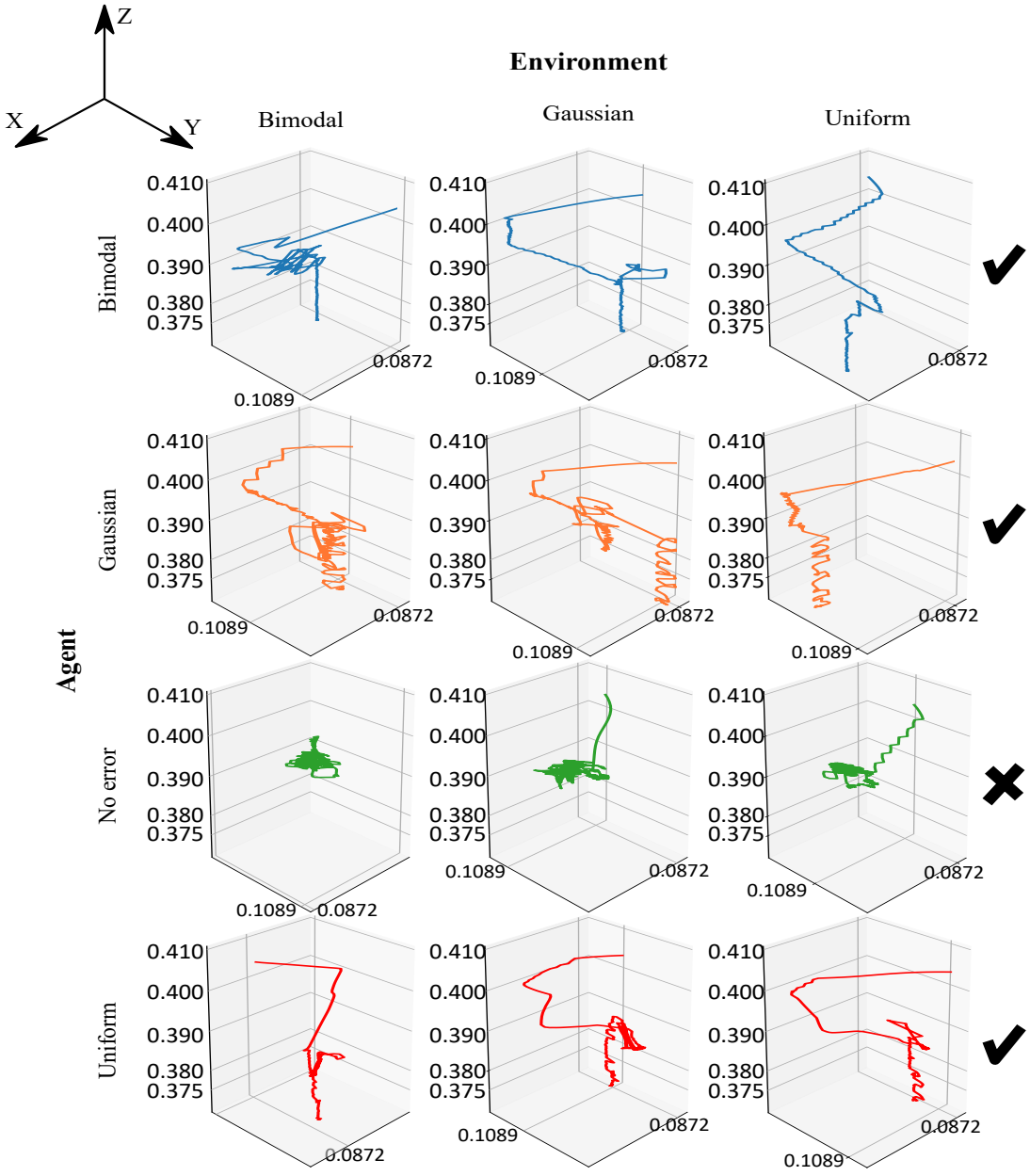

Figure 5.8: Graphs of the peg trajectory during insertion with 0.5mm peg/hole clearance. Goals are generated with 2mm respective of bimodal, Gaussian or uniform environment.

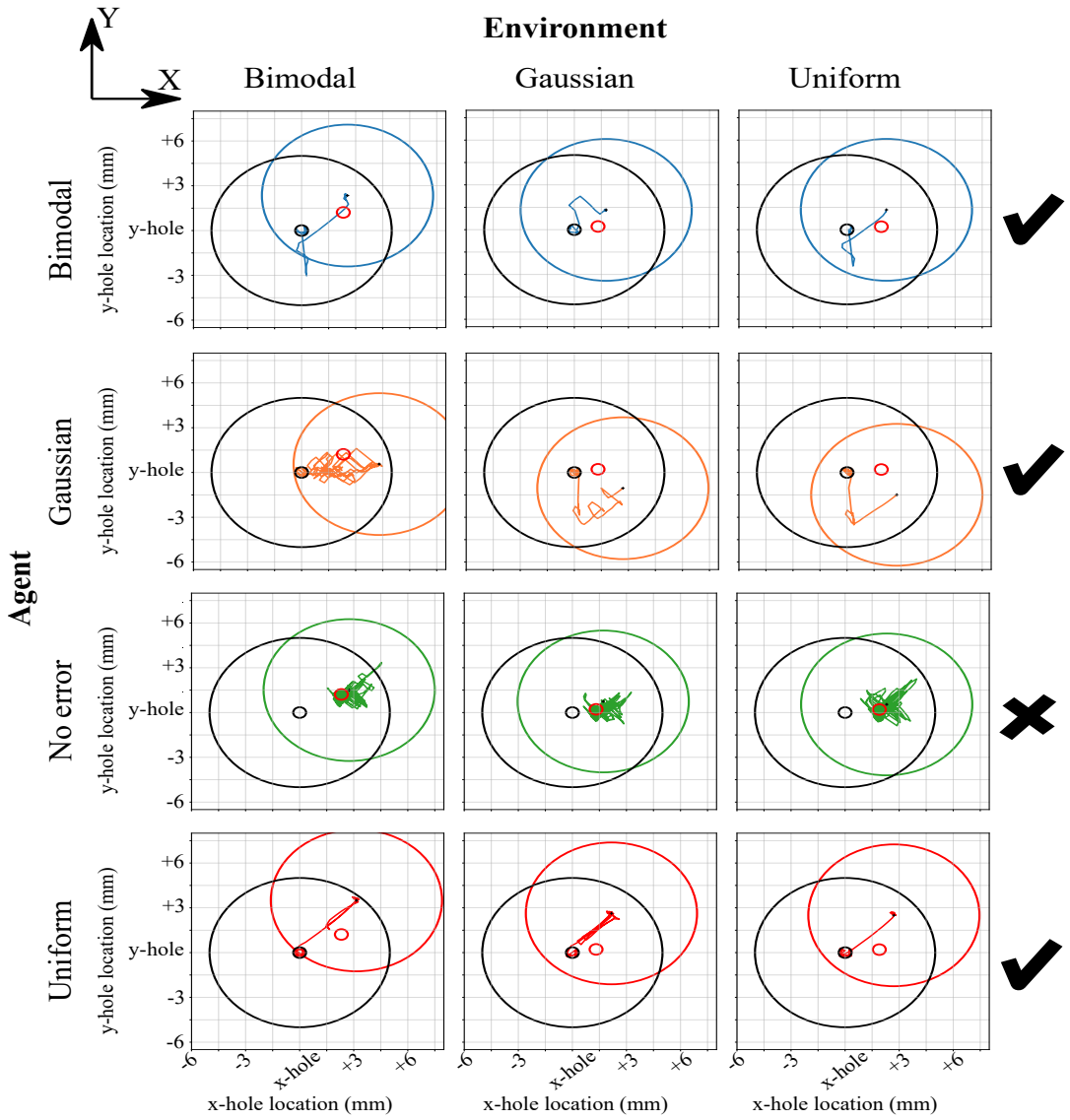

Figure 5.9: Graph showing the 2D trajectory of the peg in the (x, y) plan during insertion for each trained agent with a $0.5mm$ peg/hole clearance. Goals are generated with $2mm$ respective of bimodal, Gaussian or uniform environment.

with goal uncertainties, the search patterns towards the goal covers a larger portion of the space and appears to be more guided towards the true location of the goal. The agent trained without uncertain goals keeps the peg at a shorter distance from the perceived goal in a quasi-random pattern. Covering a larger portion of the state-space is considered as exploratory. Furthermore, the figure shows that the agent trained with uncertainty knows how to behave in the presence of uncertain

goals.

5.5 Square PiH: Results and Validation

This section shows the framework applied to a more complex fine manipulation task such as the square PiH as illustrated in Fig. 5.10. This task is more challenging since it required the agent to find the orientation of the goal as well as its position. An agent learned a policy with definite goals, and another agent learned with Gaussian goal uncertainties. The policies then allowed agents to complete the task in an environment with increasing goal uncertainties. The policy created with a Gaussian-trained agent was able to perform better for larger goal uncertainties.

5.5.1 Results

Both agents were progressively trained across more and more challenging environments with varying peg/hole clearance down to $0.5mm$. The agent trained with a definite goal was able to insert the peg with 100% rate after $2.5e^6$, whereas the agent trained with Gaussian goal uncertainties was only able to achieve a success rate of 95%.

5.5.2 Simulated Validation

The policies built by both agents attempted to complete the same task in an environment with increasingly uncertain goal locations. To validate the framework,

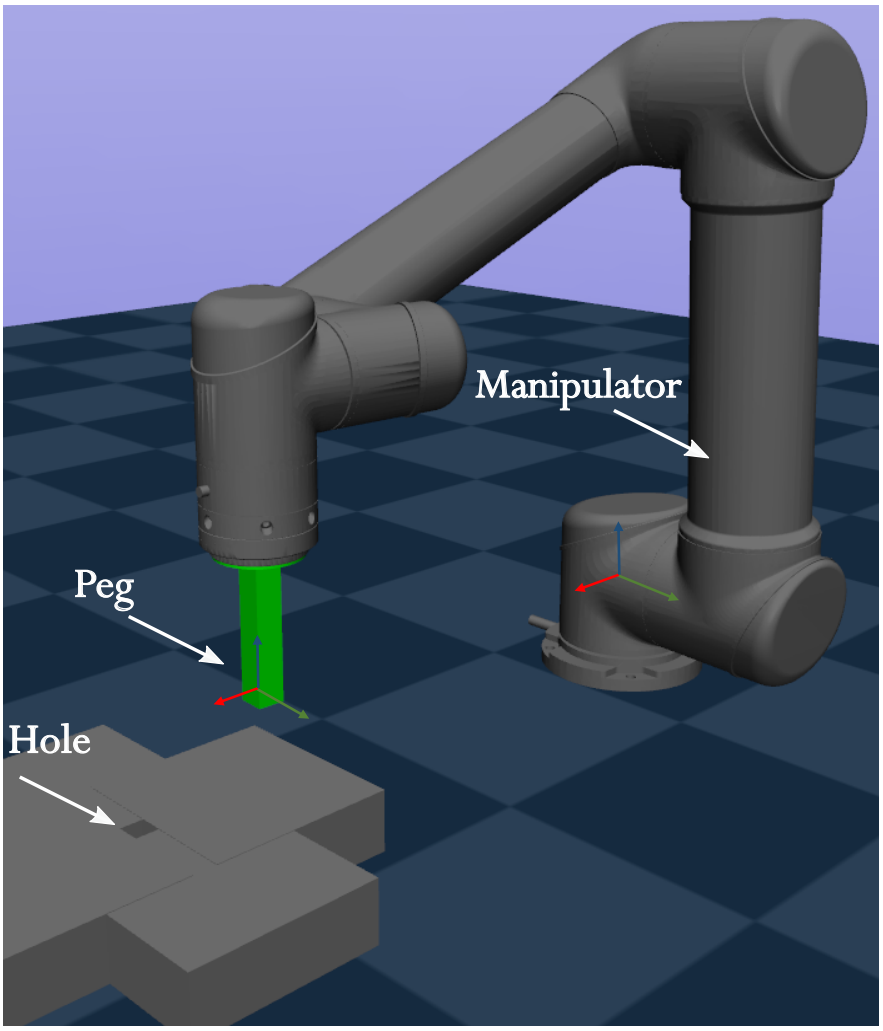

Figure 5.10: Simulation of the square Peg-in-Hole task

the agent trained with uncertainty must be able to insert the peg more often than the agent trained with definite goals as the environment uncertainties increased. Fig. 5.11 shows the success rate of both agents. The agent trained with definite goals was able to insert the peg more often when the Gaussian goal uncertainties have a standard deviation smaller than $0.001m$ and $0.01rad$ for orientation. However, for a standard deviation greater than $0.001m$, the agent trained with goal uncertainties performed up to 20% better with $0.0026m$ standard deviation. At $0.002m$ standard deviation, the uncertainty-trained agent was able to insert the peg with a 12% higher success rate.

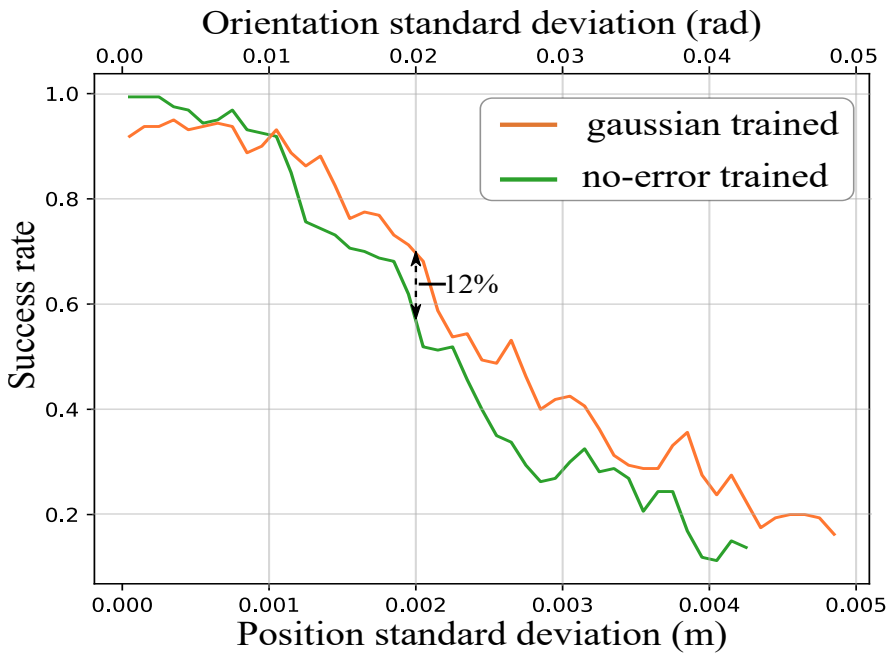

Figure 5.11: Insertion rate vs goal uncertainty for each agent trained for the square PiH task. Each value is an average over 80 episodes. Goals were generated from a Gaussian distribution from $0.0m$ to $0.005m$ standard deviation

Fig. 5.12 shows the (x, y) path and orientation of the peg during insertion while in contact with the surface. The goal was a sample of a Gaussian distribution with $0.002m$ standard deviation. The agent trained with definite goal position and orientation landed further away from the perceived goal and progressed rapidly towards it. The agent appeared lost around the perceived goal, which led to the inconsistency of the path's pattern. On the other hand, the agent trained with goal uncertainties landed closer to the perceived goal but proceeded in the consistent search pattern towards the true goal. Both agents rotated the peg to the perceived goal orientation before landing on the surface.

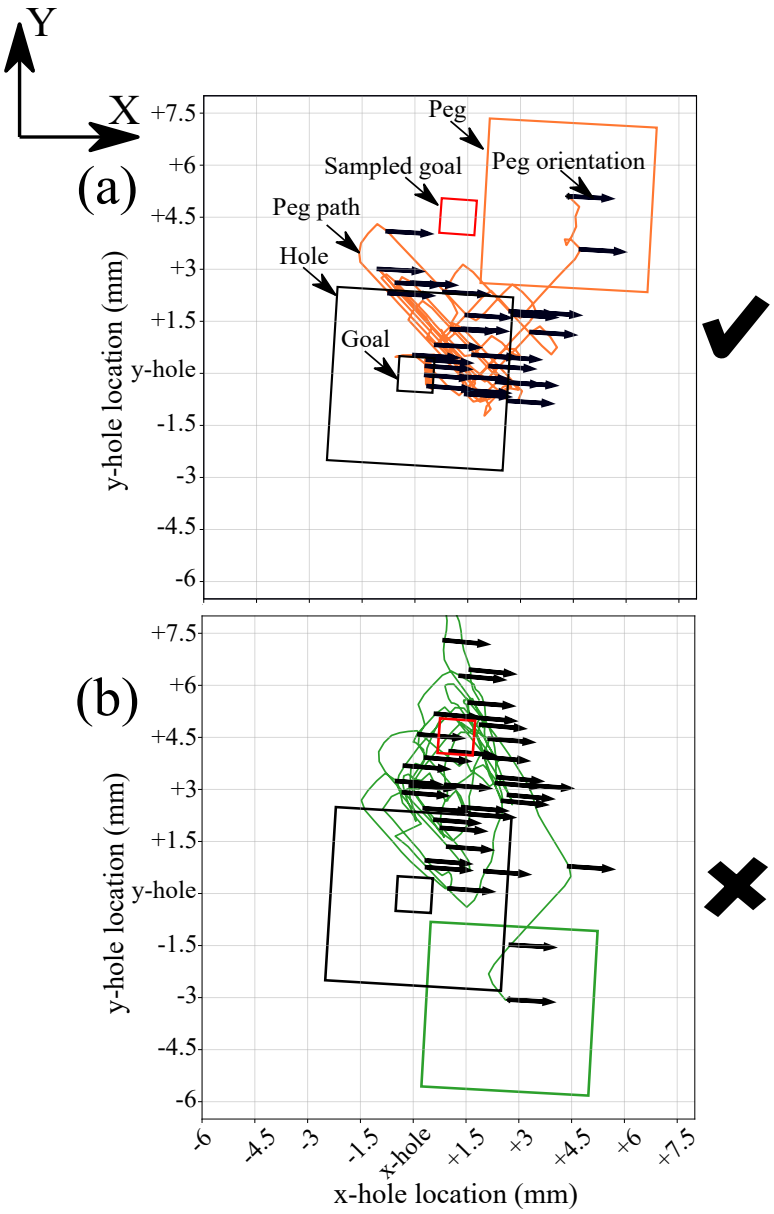

Figure 5.12: Graph showing the 2D trajectory of the peg in the (x, y) plan during insertion for each trained agent with a $0.5mm$ peg/hole clearance. Goals are generated with $2mm$ standard deviation. Agent goals Gaussian (a), No error (b).

5.6 Discussion

Training a Deep reinforcement learning agent with uncertain goals drawn from a probability density function increased the insertion rate for a PiH task. All agents trained with uncertain goals inserted the peg more often across a range

of uncertainties and environments. In all cases, the agent trained with definite goals always displayed more subpar performances in all environments against all agents trained with uncertainty. These findings are compelling as they confirm that training with goals drawn from probability distributions that include the real goal leads to better results when transferring a policy across environments.

The state-space selection guidelines, as well as the pruning technique, allowed the formulation of a concise state-space model for the problem with which all trained agents were able to complete the task entirely. Agents learned with uncertainties in a number of steps in the same order of magnitude as the agent trained with no uncertainties in the goal. Although the actor network showed similar weight distribution for all trained agents, they developed different behaviours. In the round PiH task, the agents trained with uncertainties explored a larger part of the space while the other tends to explore only a small space around the perceived goal. It suggests that training with goal uncertainty is analogous to action noise or parameter noise in DRL, which allows an agent to explore rather than exploit the current policy. In the square PiH task, the Gaussian-trained agent displayed more guided exploration through a consistent path pattern towards the goal.

The (x, y) peg pattern obtained during insertion for the goal-uncertainty-trained agents was different to traditional search patterns such as coordinate descent hole search, gradient descent hole search and Polak-Ribere hole search [109]. These techniques showed paths that were more directed towards the goal than the one shown in Fig. 5.9. However, these methods cannot apply in environments where

there are no physical indications of the goal location, such as a countersink. The proposed framework did not assume the hole's geometric features hence can be used for a wide range of Peg-in-Hole scenarios.

The scope of the framework was broad as it could be applied to numerous fine manipulation tasks. Peg-in-Hole is a generic representative of such tasks since it entails complex interaction with the environment, small clearances and fine motions. Besides, PiH tasks have a clear and visual indicator for success which was the peg inserted in the hole. This chapter showed that the framework has merit for more than one fine manipulation task: round PiH and square PiH.

The framework has limitations reflected during the square Peg-in-Hole task. It does not specify the scale of uncertainties with which to train the DRL agents. The type and amount of uncertainty are task-specific and can be challenging to tune during the training to obtain a high enough success rate. Finding the uncertainties with which to train the square PiH agent was time-consuming and only led to a success rate of 95%. Furthermore, it is unclear when the uncertainty-trained agent will start to perform better when increasing goal uncertainty. The ability to quantify the uncertainties in the goal dictates whether the agent should train with or without goal uncertainties.

The difference of insertion rates between the two agents in the square PiH task was only 12% compared to the 40% difference in the round PiH task for the same agents at $2mm$ standard deviation. This difference is reflective of another

limitation of the framework. In the square PiH task, the goal elements have two scales: radians and meters; hence it was more challenging to tune both scales for optimal training performance. Other fine manipulation agents might require forces or torques as goal elements whose training uncertainties would be challenging to tune. The framework could scale poorly with the number of scales in the goal features. Normalising the goal features and training uncertainties could be a way to improve scaling the framework.

The paradigm presented in this work showed that an agent trained with uncertain goals performed better across a range of uncertainties during validation. This framework eliminates the need for complex mechanisms to recover the exact goal for fine robotic manipulation tasks and ensures a more robust transfer of the policy from "sim-to-real".

5.7 Conclusion

This chapter showed that progressively training DRL agents with goal uncertainties led to higher success rates across multiple environments and tasks. Four agents trained for a round PiH task: three with goal uncertainties and one with perfect knowledge of the goal. The agents trained with uncertainties had higher success rates when applied to a peg insertion task across a range of environments. These agents explored a larger part of the state-space during insertion while maintaining a guided route towards the hole. Each agent created a different policy; however,

weight distribution for the actor network showed no particular shift towards force/torque elements of the input feature vector. Two agents then trained for a square PiH task. Again the agent trained with uncertainties performed better across a range of environments although trained only up to 95% insertion rate. This agent showed a more consistent search pattern towards the goal. The framework applies to multiple manipulation task and mitigates the effects of uncertainties in these tasks.

Chapter 6: Experimental Validation

Chapter 5 showed that the policies generated using the framework had a higher success rate than polices created with no goal uncertainties across a range of environments in simulation. In this chapter, a physical robotic manipulator utilised the policies generated earlier and successfully validated them. It shows that the ones with goal uncertainties have higher success at inserting the peg than the one generated with definite goals. It is consistent with the simulation results and validates the findings where the framework mitigates the effects of uncertainties from "sim-to-real".

6.1 Introduction

Fine manipulation tasks require complex interactions with the world, which is difficult for robotic manipulators. DRL requires extensive interactions but allowing agents to learn on physical hardware is not realistic. An alternative is to allow an agent to learn a policy in simulation and to transfer it to real hardware.

Transferring policies from "sim-to-real" is problematic as agents learn from exact measurements in a simulation when sensors estimate these measurements in the real world. The framework in this work mitigates the effects of emerging uncertainties from such transfer. It allowed an agent to perform better across

a range of simulated environments in the presence of goal uncertainties. This chapter shows that the simulation's findings were also true for a physical robotic manipulator.

This chapter is organised as follows: Section 6.2 presents the hardware used to validate the policies and describes the robot's task and environment. Section 6.3 describes the methodology used to validate the policies as well as the results for both versions of PiH. Section 6.4 discusses the implications of the results. Finally, 6.5 concludes this chapter.

6.2 **Robot Hardware, Software and Environment**

The robot controlled by the DRL agents was a Universal Robot® UR5 arm with a peg as an end-effector. The manipulator was constrained to 3 and 4 degrees of freedom for the round and square PiH tasks, respectively. They corresponded to the action-space of the DRL agents. A force/torque sensor was attached between the robot's last link and the peg to measure force and torque at the wrist. The hole was machined into a $10cm \times 10cm \times 5cm$ block of aluminium having the desired peg/hole clearance. Fig. 6.1 depicts the complete apparatus.

Robot Operating System (ROS) nodes performed control and sensor readings from the robot. ROS provides a framework for communication between fragmented processes, called nodes, and a collection of open-source software packages. The

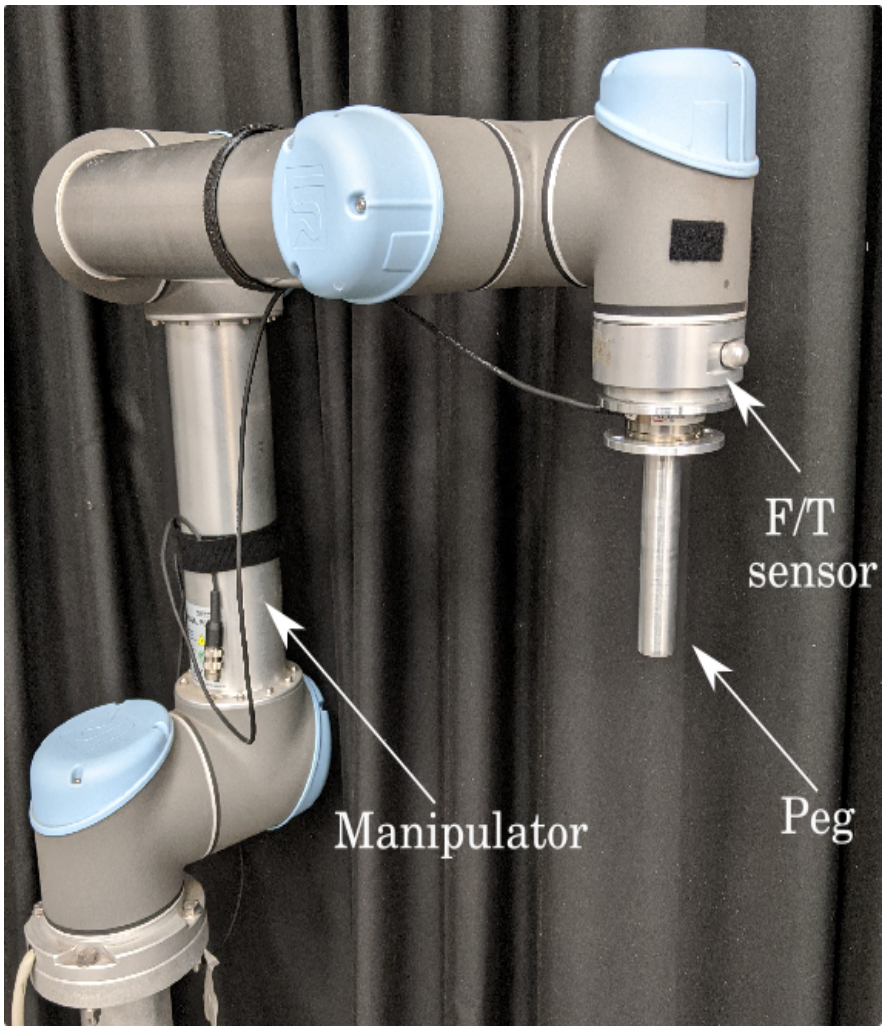

Figure 6.1: Robotic manipulator with attached force/torque sensor and peg.

motion planning library, MoveIt, was used to solve the differential kinematics allowing the DRL agents to control the robot's end-effector directly. The models trained in the simulation used the real position and force/torque readings, and the actions were applied to the robot. Fig. 6.2 shows the system architecture.

Fig.6.3 shows the peg and hole for both the round and square PiH tasks. The peg's position was the same at the start of each experimental trial. The hole location, on the other hand, was sampled randomly. The maximum distance between the peg and the hole was $4.9cm$ at the start of each episode. The agent had a zone of

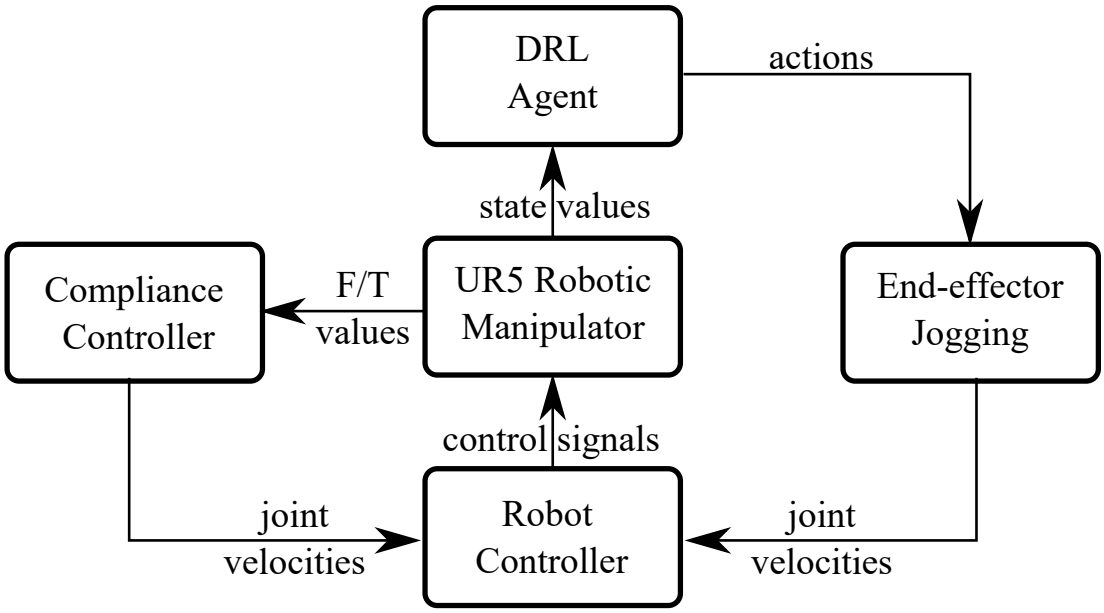

Figure 6.2: Experimental system's architecture for validation on a robotic manipulator

operation of $125cm^3$ outside of which the insertion failed. The hole was placed by jogging the peg inside the hole manually. A compliance controller prevented damage to the hardware by reducing the forces on the manipulator.

6.3 **Experimental procedure and results**

6.3.1 **Round PiH**

Seven goal locations were sampled randomly to assess the four DRL agents. Uncertainty was then added to the goal based on a probability density function. The agents had five tries to insert the peg in the hole. Re-positioning the hole after each trial ensured consistency of the results.

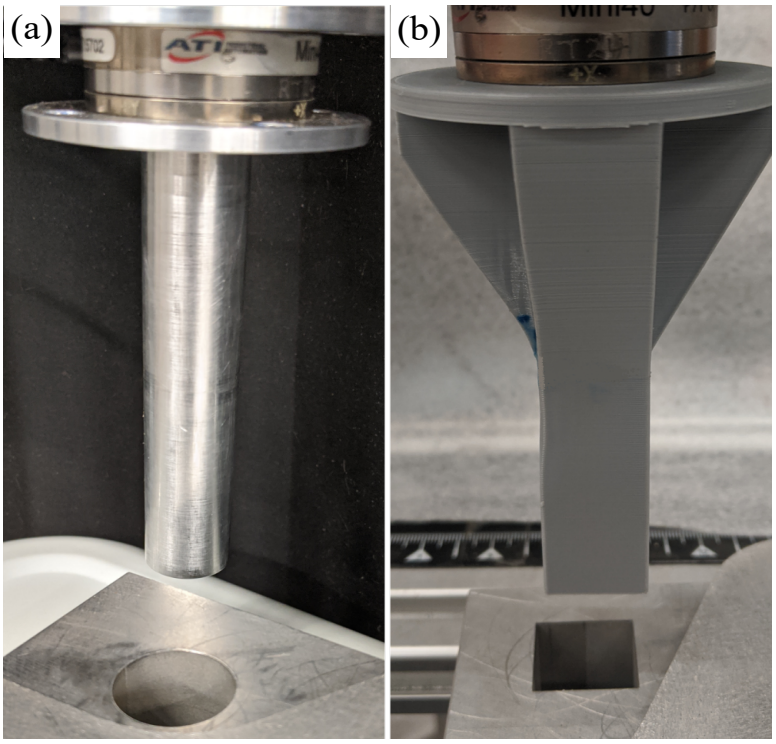

Figure 6.3: Robotic end-effector for round (a) and square (b) PiH and holes used for experimental validation.

Fig. 6.4 shows the peg trajectory in 3 dimensions (Fig. 6.4(a)) and 2 dimensions (Fig. 6.4(b)) for the agent trained with Gaussian goal uncertainties. In the figure, the 2D trajectories begin after the peg made contact with the surface. The peg landed near the perceived goal location and proceeded directly towards it. After passing the sampled goal, the agent moved in zigzag motions towards the true hole location where it inserted the peg. Fig. 6.5 shows the same information for the agent trained with no uncertainty. The peg landed in a similar location, then moved towards the perceived goal. When not able to insert the peg, the agent moved the peg in a loop and continued its path in the wrong direction hence failing to insert the peg.

Table 6.1 shows the successful and failed trials for each of the seven agents' goal

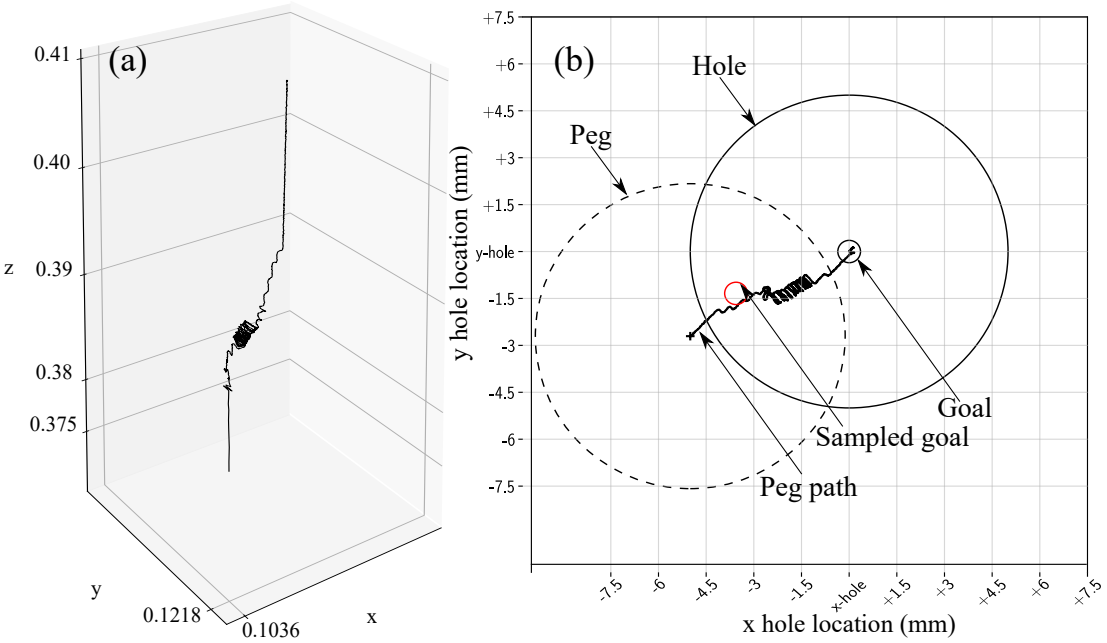

Figure 6.4: (a) 3D trajectory and (b) 2D trajectory of the peg during successful insertion for the round PiH task.

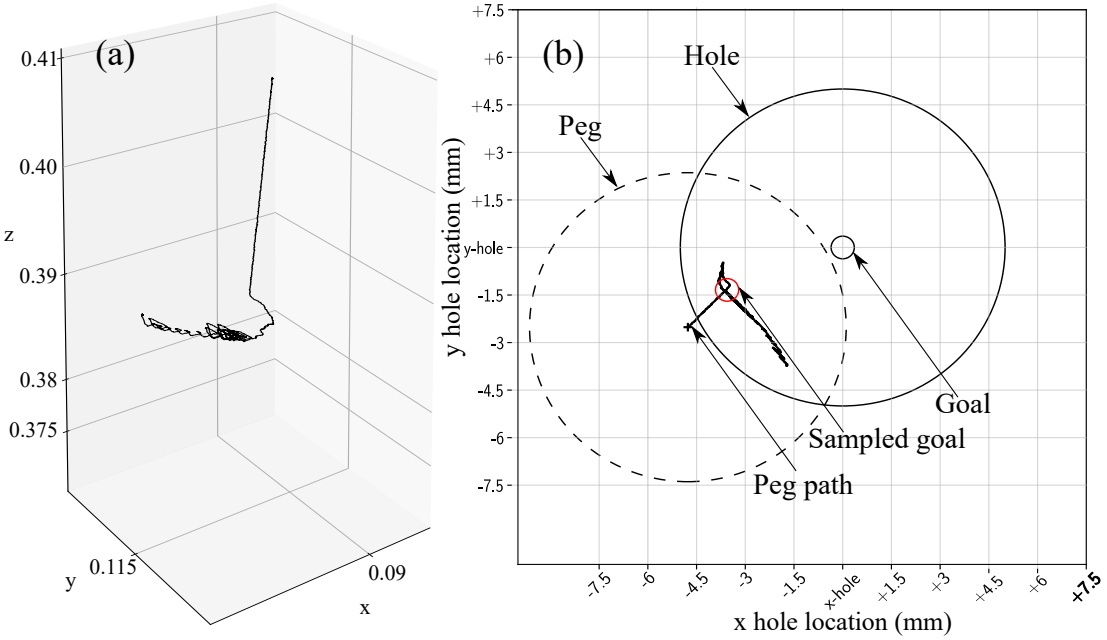

Figure 6.5: (a) 3D trajectory and (b) 2D trajectory of the peg during failed insertion for the round PiH task.

locations. The definite goal positions come from a uniform distribution with a Max/Min of 1cm from the peg in x and y. The noisy goal locations were sampled from a Gaussian distribution with 2mm standard deviation. The results show

Table 6.1: Experimental validation results for a round PiH task.

Goal	Goal coordinates		Sample goal coordinates		Gaussian agent success rate	**No-error agent success rate**	Bi-modal agent success rate	Uniform agent success rate
	X (mm)	Y (mm)	X (mm)	Y (mm)				
1	77.82	114.98	76.49	117.60	100%	**100%**	100%	0%
2	90.74	109.00	90.49	110.67	100%	**100%**	100%	0%
3	84.76	105.66	85.68	107.13	100%	**0%**	100%	100%
4	89.14	114.38	90.92	114.38	100%	**0%**	0%	100%
5	77.30	105.13	78.17	103.87	100%	**100%**	100%	100%
6	92.98	104.68	95.09	106.67	0%	**0%**	0%	100%
7	92.98	104.68	92.68	104.68	100%	**100%**	100%	100%

that an agent either failed or succeeded in all trials for a given goal. The agent trained with Gaussian goal uncertainties inserted the peg successfully for 30 of the 35 trials. The agent with lower insertion rates was the one trained with definite goals with only 20 successful trials out of 35. Goal 7 had no error in the hole position to ensure that each of the agents was still able to insert the peg. The table represents a total of 140 experimental trials. Each trial lasted about 2 minutes which included re-positioning the hole; re-setting the manipulator; biasing the force/torque sensor values and conducting the insertion. It added up to a total of 4.6 hours of experimental trials for the round PiH.

Fig. 6.6 shows the peg's motions during insertion. In Fig. 6.6(a-d), the agent trained with no goal uncertainties moved away from the hole before landing on the surface. It then moved towards the perceived goal and remained close to it, hence failing to insert the peg. The four images span over 35 seconds, where the maximum peg velocity is $0.001m/s$. In Fig. 6.6(e-h) the agent trained with Gaussian goal uncertainties landed at the same location. It then instead moved towards the hole before inserting the peg. The successful agent completed the

task in 17 seconds, half the time attributed to the other agent. The maximum peg velocity remained $0.001m/s$. A video of the trials are provided as part of this thesis.

6.3.2 **Square PiH**

The same experiment was repeated for a square PiH task. A square peg was attached to the last link of the robotic manipulator. Seven goal locations with uncertainties were generated using a Gaussian distribution with $2mm$ standard deviation in position and $0.02rad$ in orientation. For each of the uncertain goals, both agents had three trials to insert the peg. The hole was re-positioned after each trial.

Table 6.2 shows the results of the experimental trials. The definite goal positions were samples of uniform distribution with a Max/Min of $1cm$ from the peg for positions x and y and $0.1rad$ for orientation θ. The agent trained with definite goals inserted the peg three times out of the 21 trials, whereas the agent trained with Gaussian goal uncertainties inserted the peg nine times. The table shows the results of 42 trials lasting an average of 2.5 minutes each which included re-positioning of the hole and manipulator, biasing of the force/torque values and insertion of the peg. It added up to a total of 1.75 hours of experimental trials. The results showed that the policy from the agent trained with uncertainty transferred better from "sim-to-real".

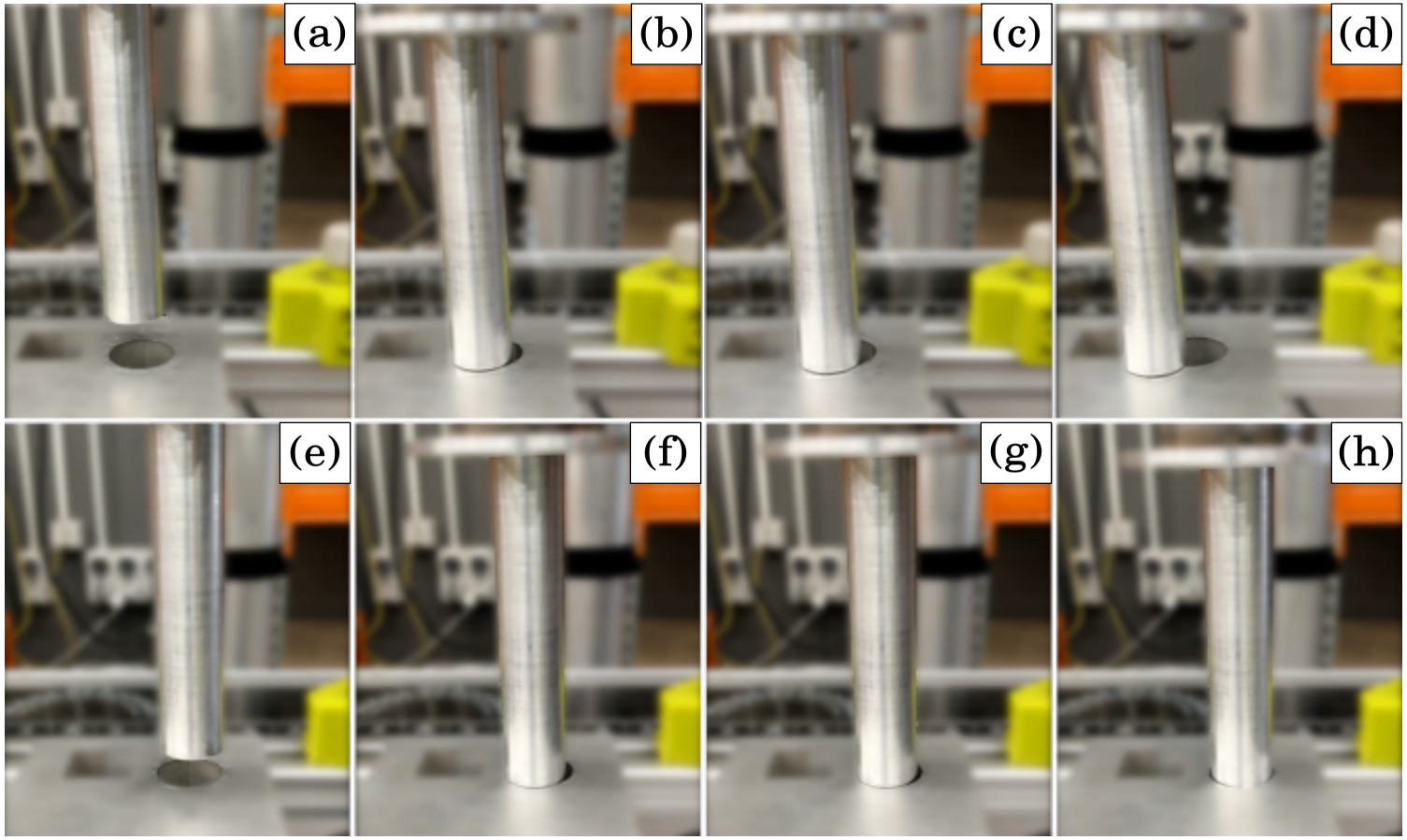

Figure 6.6: Peg insertion sequence for a round PiH task. (a-d) show the failed insertion from the agent trained with definite goals. (e-h) is a successful insertion from the agent trained with Gaussian goal uncertainties.

Table 6.2: Experimental validation results for a square PiH task

Goal	Goal			Sample goal			Gaussian	**No-error**
	X (mm)	Y (mm)	th (rad)	X (mm)	Y (mm)	th (rad)		
1	84.73	117.76	-0.0248	85.77	116.38	-0.0535	100%	**0%**
2	82.92	106.72	-0.037	79.13	107.06	-0.0226	100%	**100%**
3	94.11	103.16	0.0472	95.76	103.34	0.0186	100%	**0%**
4	90.04	107.44	-0.091	91.74	109.8	-0.1023	0%	**0%**
5	83.55	105.77	-0.0073	81.76	106.85	-0.0062	0%	**0%**
6	87.78	110.42	-0.0354	88.27	111.66	-0.0357	0%	**0%**
7	90.34	116.83	-0.0045	89.4	115.46	-0.0377	0%	**0%**

Fig. 6.7 shows the motion of the peg during insertion. In Fig. 6.7(a-d), the agent trained with definite goals failed to insert the peg by moving away from the true goal after landing. The four images span over 20 seconds after which the process was interrupted. In Fig. 6.7(e-h), the agent trained with uncertain goals inserted the peg by moving towards the true location of the hole after landing closer to it. The insertion lasted 22 seconds with the peg moving at a maximum $0.001m/s$. A video of both trials is provided as part of this thesis.

6.4 **Discussion**

The experimental trials proved that the framework allowed policies from agents trained with uncertain goals in simulation to transfer with higher success rates on a real robot. In the round PiH experiments, the agent with definite goals had a success rate of 57% whereas the agent trained with bi-modal, uniform and

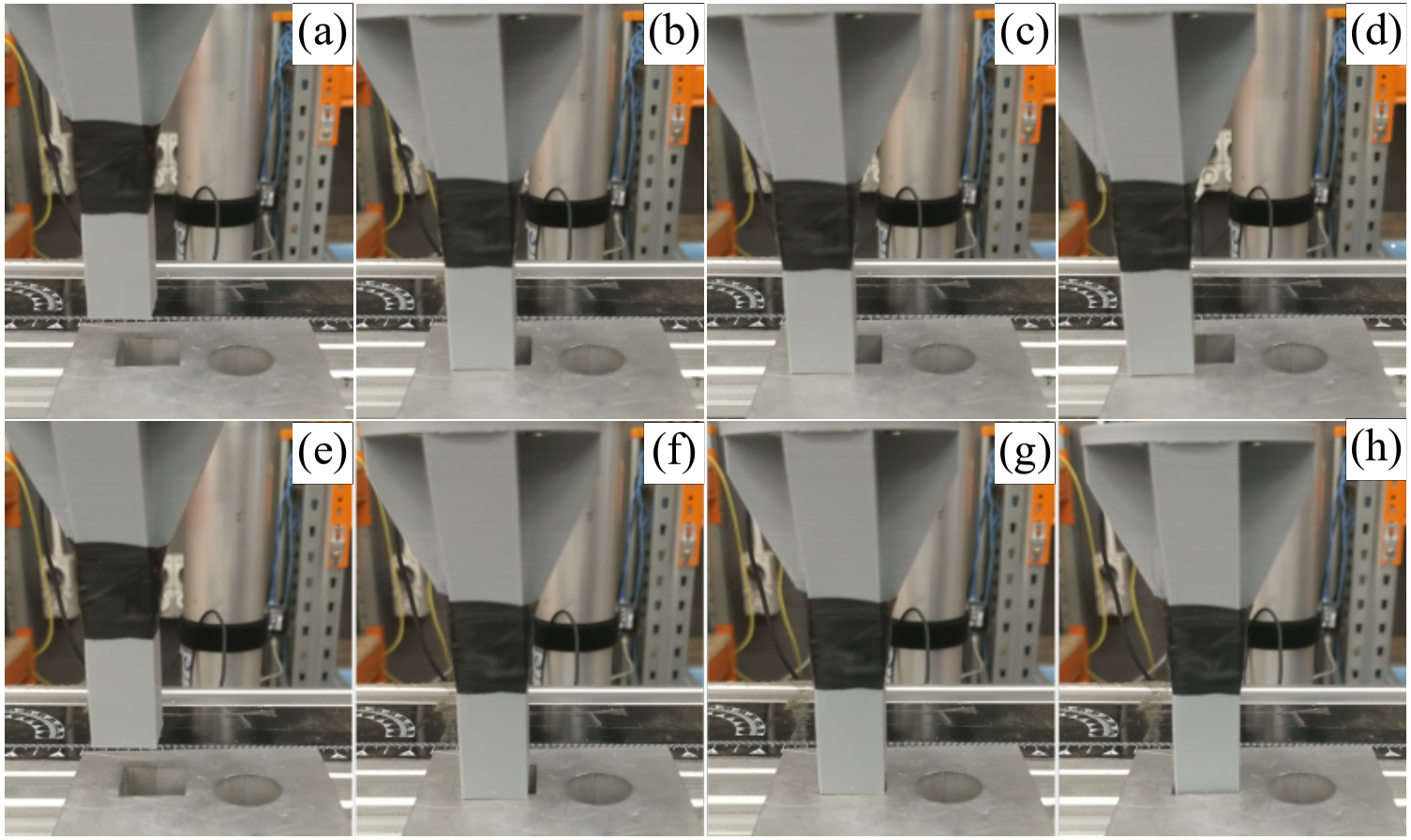

Figure 6.7: Peg insertion sequence for a square PiH task. (a-d) shows a failed insertion from the agent trained with definite goals. (e-h) show the successful insertion from the agent trained with Gaussian goal uncertainties.

Gaussian uncertainty had higher rates of 71%, 71% and 86% respectively. Similarly, in the square PiH experiments, the agent trained with uncertainty had a success rate of 43% versus 14% for the agent trained with definite goals. These results show consistency with simulation results as well as the mathematical framework presented in Chapter 4. An agent trained with uncertain goals drawn from a probability distribution that includes the true goal is bound to perform better in the presence of goal uncertainties.

The scope of the framework is to enable DRL agents to perform a task better in the presence of goal uncertainty. Transferring policies from "sim-to-real" without losing performance is beyond the scope of this framework. Although the state-space formulation technique presented aimed at mitigating the adverse effects of policy transfer, it did not ensure that performance was maintained. It ensured, on the other hand, that the states encountered after transfer were not so different as to nullify the policy obtained in simulation. Maintaining a 43% insertion rate experimentally versus 70% in simulation for the Gaussian-goal trained agent in a square PiH task demonstrates the intent of the framework.

Similarly, it was not in the scope of the framework to train the agents to insert the peg with a considerable distance between peg and hole. The experimental trials gave the manipulator a $125cm^3$ zone of operation which corresponded to a maximum distance between peg and hole of $8.67cm$. This work assumes that other trivial path planning and control techniques would be used for the peg/hole distance to reach this threshold.

The experimental trials used goals drawn from a probability density function with $2mm$ standard deviation. This scale is similar to one from estimating a hole position based on point cloud processing or computer vision. The experimental proof was limited since the trials were conducted with only one uncertainty scale. Nonetheless, the consistency with the simulation results suggested that agents are likely to behave similarly for a range of uncertainties.

The 28% drop in success rate between the simulated PiH agent and the experimental one can be attributed to several factors of which force/torque sensing was a significant contributor. Although low-pass filtering reduced noise, dissimilarities between simulation and experiment such as the coefficient of friction of the material can lead to states far from those observed in simulation. Another cause of lower success rates from "sim-to-real" was the type of controller used during experimental validation. A velocity controller replaced, the position controller used from training. The agent's actions, however, remained a difference in position for each of the elements in the action-space. In order to match the controls, the maximum velocity of the end-effector was tuned for the control speed so that the maximum displacement of the end-effector between control loops matched the maximum displacement allowed in simulation. This process induced uncertainties in the controls due to the changing speed of the control loop. These uncertainties affected significant aspects of the peg insertion, such as the number of steps after which the agent expected the peg to hit the surface.

Nonetheless, the results showed that following the framework led to training DRL

agents for fine manipulation tasks while ensuring that policies transferred to robotic manipulators yielded higher success rates. It eliminates the need to recover goal states precisely and by consequence, widens the range of sensors that can be utilised for a manipulation task. It also increases the cases of fine manipulation applications that can be solved using DRL since computationally expensive filters are no longer needed to recover goal states.

6.5 **Conclusion**

The policies of DRL agents trained with and without goal uncertainties were transferred to a physical robotic manipulator for a round and square PiH task. The results from the experimental trials showed that the agents trained with goal uncertainties had a higher success rate. These results were consistent with the simulation results obtained in Chapter 5. The framework offers a solution for DRL agents to perform better in the real world in the presence of goal uncertainties.

Chapter 7: Conclusion and Future Work

7.1 General Conclusion of the Thesis

The thesis is dedicated to the application of deep reinforcement learning to fine manipulation tasks with goal uncertainties. The general conclusions of the thesis are as follows:

Chapter 1 introduced the background, aims and objectives of the thesis.

Chapter 2 introduced and explained the fundamental mathematical tools used throughout the thesis, which consisted primarily of Deep reinforcement learning. A deep deterministic policy gradient algorithm, a class of actor/critic algorithm, as well as the hindsight experience replay wrapper, instantiated the formalism.

Chapter 3 provided background information and context to the thesis. It began with the literature on mitigating the effects on uncertainties for robotic manipulation and focused on approaches to the Peg-in-Hole problem. It then presented the literature for deep reinforcement learning for fine robotic manipulation while highlighting those that considered Peg-in-Hole. Finally, it showed how the literature mitigated uncertainties for DRL in fine manipulation.

Chapter 4 adopted the deep reinforcement learning formalism and developed a framework to apply it to fine manipulation tasks. The framework aimed to fill the gaps identified in the literature. It had three significant aspects: first, guidelines to formulate DRL agent state-spaces. The guidelines led to a state-space feature vector that contained only the required features for learning. It helped reduce computations; it allowed DRL fine manipulation tasks to have a benchmark, and it ensured that DRL agents could learn with goal uncertainties. Second, a method for DRL agents to create policies that are robust to goal uncertainties. It consisted of artificially adding goal uncertainties during training to create exploratory policies. Finally, a progressive training method that allowed agent trained with an uncertain goal to converge to a solution.

Chapter 5 instantiated the framework from the previous chapter for fine manipulations. The state-space guidelines led to formulating the state-space feature vector of the round PiH manipulation problem. Three agents progressively learnt to complete the task with different, artificially induced, goal uncertainties. The policies were validated in simulation and showed that each of the agents trained with uncertainties performed better than the one trained with definite goals. The framework was then applied to a square PiH task; the validation in simulation again showed that the agent trained with goal uncertainties had a higher insertion rate when goal uncertainties increased.

Chapter 6 validated the policy generated in simulation on a physical robotic manipulator. The robot used the policies to complete both round and square PiH

tasks for seven goal locations drawn from a probability density function for each task. The agent insertion rates were smaller from the ones in simulation however the agents trained with goal uncertainties performed up to 28% better for the round PiH task and 29% for the square PiH task.

Chapter 7 concluded the thesis, highlighted all of the work's contributions and gave direction for future work.

7.2 **Contributions and Main Achievements of the Thesis**

The work adopted a new approach to DRL agent's state-space features formulation for fine manipulation tasks. The approach led to a concise state-space formulation with sufficient information to learn a fine manipulation task. The approach consisted of following guidelines that extracted features representing the goal, constraints and controlled quantities and concatenated them into one state-space vector. The approach allowed the formulation of the state-space features for two fine manipulation tasks: round and square PIH. This contribution of the work is a significant step for using DRL to solve more complex manipulation tasks. Furthermore, It helps to disassociate formulation and solution to a fine manipulation task, so that manipulation can start to have benchmark DRL solutions.

The work further introduced a method to create policies more robust to goal

uncertainties which is a novel approach in the literature. It consisted of training DRL agents with goals sampled from a probability density function that contained the true goal. The work showed that such an approach was bound mathematically to increase the intersection between the task goal set and the training goal set and by consequence was ensuring the agent was familiar with its environment. This contribution will lead to robust DRL solutions that are independent of sensors, controls, actuator, computation processes or environment's uncertainties.

Additionally, the work developed a technique to progressively train DRL agents with goal uncertainties to ensure that they converged to a solution. It consisted of scaling the physical task difficulties based on the agent's success rate. It provided a formalism to ensure that DRL agents could learn fine manipulation and gave indications to a DRL agent architect about when to consider further hyper-parameter tuning. It is a concrete methodology to train and tune DRL agents as opposed to fragmented and sparse approaches that are seen in the literature.

Finally, applying all the methods previously mentioned in the form of a framework to a fine manipulation task was the main achievement of the thesis. Indeed this work not only developed the framework but concretely validated it both in a simulation and experimentally. Its application to two Peg-in-Hole tasks demonstrated how the state-space model guidelines, training goal uncertainties and progressive training led DRL agents to solve fine manipulation problems better than ones trained without all aspects of the framework.

The current applications of model-free, end-to-end deep reinforcement learning in the robotic literature remains confined to robotic navigation and simulated robotic manipulation. Indeed, real-world manipulation is far more challenging to simulate accurately due to contacts while remaining impractical to learn on real hardware. This work brings DRL one step closer to applications to real-world fine manipulation tasks by allowing DRL agents to act in the uncertain world that is ours.

7.3 **Future Work**

The thesis focused on developing a framework to create DRL agents more robust to goal uncertainties. It then validated the framework in simulation and experimentally. Future researches should be directed towards the following aspects:

- Quantifying the scale of uncertainties with which to train agents with respect to the fine manipulation task parameters. It is one of the main limitations of the framework hence the more important to address.
- Normalising goal features. The more complex the manipulation task, the more scales that exist in the goal features. Normalising all features would limit the training uncertainties scales to one and by consequence reduce tuning time.
- Validating the framework on more fine manipulation tasks. The work validated the framework for two tasks with only one uncertainty probability

density function experimentally. More variations of different manipulation tasks should be tested to show the reliability of the framework.

The applications of the proposed method are numerous. A fully trained autonomous agent can be used in conventional manufacturing and assembly environment such as cars assembly lines for mounting external panels. It can also perform in uncontrolled environment for large scale tasks such as down-hole sampling in the mining industry and for more widespread task such as autonomous car recharging or refueling.

List of Publications Arisen from this PhD Study

Journal paper

T. Rouillard, I. Howard, and L. Cui, "Progressive Training of a Deep Reinforcement Learning Agent for a Peg-in-Hole Task with Goal Uncertainties," in *Journal of Intelligent Robotic Systems*. (**Under review**)

Book Chapter

T. Rouillard, I. Howard, and L. Cui, "Autonomous Two-Stage Object Retrieval Using Supervised and Reinforcement Learning," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2019: IEEE, pp. 780-786.

Bibliography

[1] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998 (cit. on pp. 7, 9).

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. ISBN: 0262352702 (cit. on pp. 8, 11, 15, 37).

[3] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". Thesis. 1989 (cit. on pp. 9, 12).

[4] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Vol. 1. Athena scientific Belmont, MA, 1995 (cit. on p. 9).

[5] Richard S Sutton. "Introduction: The challenge of reinforcement learning". In: *Reinforcement Learning*. Springer, 1992, pp. 1–3 (cit. on p. 9).

[6] Marco Wiering and Martijn Van Otterlo. "Reinforcement learning". In: *Adaptation, learning, and optimization* 12 (2012) (cit. on pp. 11, 14).

[7] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013) (cit. on pp. 13, 48).

[8] Shixiang Gu et al. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396. ISBN: 150904633X (cit. on pp. 13, 36, 38).

[9] Francois De La Bourdonnaye et al. “Learning to touch objects through stage-wise deep reinforcement learning”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9. ISBN: 1538680947 (cit. on pp. 13, 35).

[10] Jing Xu et al. “Feedback Deep Deterministic Policy Gradient with Fuzzy Reward for Robotic Multiple Peg-in-hole Assembly Tasks”. In: *IEEE Transactions on Industrial Informatics* (2018). ISSN: 1551-3203 (cit. on pp. 13, 36, 43, 47, 48).

[11] Marcin Andrychowicz et al. “Hindsight experience replay”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058 (cit. on pp. 14, 18, 35, 39, 43, 46, 54).

[12] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015) (cit. on pp. 14, 18, 49).

[13] David Silver et al. “Deterministic policy gradient algorithms”. In: *ICML*. 2014 (cit. on pp. 14, 35, 46).

[14] Sebastian B Thrun. “Efficient exploration in reinforcement learning”. In: (1992) (cit. on p. 17).

[15] Matthias Plappert et al. “Parameter space noise for exploration”. In: *arXiv preprint arXiv:1706.01905* (2017) (cit. on p. 18).

[16] Andrew Y Ng, Daishi Harada, and Stuart Russell. “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *ICML*. Vol. 99. 1999, pp. 278–287 (cit. on pp. 18, 40, 46).

[17] Jens Kober, J Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. ISSN: 0278-3649 (cit. on pp. 18, 40).

[18] Leslie P Kaelbling and Tomas Lozano-Perez. *Integrated robot task and motion planning in the now*. Report. Massachusetts inst of tech Cambridge computer science and artificial intelligence, 2012 (cit. on p. 22).

[19] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005. ISBN: 0262303809 (cit. on p. 23).

[20] Mohamed Elbanhawi and Milan Simic. “Sampling-based robot motion planning: A review”. In: *IEEE Access* 2 (2014), pp. 56–77. ISSN: 2169-3536 (cit. on p. 23).

[21] Amine Abou Moughlbay, Enric Cervera, and Philippe Martinet. “Model Based Visual Servoing Tasks with an Autonomous Humanoid Robot”. In: *Frontiers of Intelligent Autonomous Systems*. Ed. by Sukhan Lee, Kwang-Joon Yoon, and Jangmyung Lee. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 149–162. ISBN: 978-3-642-35485-4. DOI: 10.1007/978-3-642-35485-4_12. URL: http://dx.doi.org/10.1007/978-3-642-35485-4_12 (cit. on p. 23).

[22] Zhi Liu, Ci Chen, and Yun Zhang. “Decentralized robust fuzzy adaptive control of humanoid robot manipulation with unknown actuator backlash”. In: *IEEE Transactions on Fuzzy Systems* 23.3 (2014), pp. 605–616. ISSN: 1063-6706 (cit. on p. 23).

[23] Zhiqiang Sui, Odest Chadwicke Jenkins, and Karthik Desingh. "Axiomatic particle filtering for goal-directed robotic manipulation". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4429–4436. ISBN: 1479999946 (cit. on pp. 24, 26).

[24] Kun Qian et al. "Probabilistic Joint State Estimation of Robot and Non-static Objects for Mobile Manipulation". In: *International journal on smart sensing and intelligent systems* 5.4 (2012), pp. 1081–1096 (cit. on p. 24).

[25] Adam Bry and Nicholas Roy. "Rapidly-exploring random belief trees for motion planning under uncertainty". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 723–730. ISBN: 1612843859 (cit. on p. 24).

[26] Masanao Aoki. "Optimal control of partially observable Markovian systems". In: *Journal of The Franklin Institute* 280.5 (1965), pp. 367–386. ISSN: 0016-0032 (cit. on p. 24).

[27] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Partially observable Markov decision processes for artificial intelligence". In: *KI-95: Advances in Artificial Intelligence: 19th Annual German Conference on Artificial Intelligence Bielefeld, Germany, September 11â13, 1995 Proceedings*. Ed. by Ipke Wachsmuth, Claus-Rainer Rollinger, and Wilfried Brauer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 1–17. ISBN: 978-3-540-44944-7. DOI: 10.1007/3-540-60343-3_22. URL: http://dx.doi.org/10.1007/3-540-60343-3_22 (cit. on p. 24).

[28] Jue Kun Li, David Hsu, and Wee Sun Lee. "Act to See and See to Act: POMDP planning for objects search in clutter". In: *Intelligent Robots and*

Systems (IROS), 2016 IEEE/RSJ International Conference on. IEEE, 2016, pp. 5701–5707. ISBN: 1509037624 (cit. on p. 24).

[29] Ngo Anh Vien and Marc Toussaint. "POMDP manipulation via trajectory optimization". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 242–249. ISBN: 1479999946 (cit. on p. 25).

[30] Ferdian Pratama, Sungmoon Jeong, and Nak Young Chong. "Learning manipulative skills using a POMDP framework". In: *Ubiquitous Robots and Ambient Intelligence (URAI), 2014 11th International Conference on*. IEEE, 2014, pp. 169–175. ISBN: 1479953334 (cit. on p. 25).

[31] Joni Pajarinen and Ville Kyrki. "Robotic manipulation in object composition space". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 1–6. ISBN: 1479969346 (cit. on p. 25).

[32] Joni Pajarinen and Ville Kyrki. "Robotic manipulation of multiple objects as a POMDP". In: *Artificial Intelligence* (2015). ISSN: 0004-3702 (cit. on pp. 25, 26).

[33] Leslie Pack Kaelbling and Tomas Lozano-Perez. "Integrated task and motion planning in belief space". In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1194–1227. ISSN: 0278-3649 (cit. on p. 25).

[34] Pol Monso, Guillem Alenya, and Carme Torras. "Pomdp approach to robotized clothes separation". In: *Intelligent Robots and Systems (IROS),*

2012 IEEE/RSJ International Conference on. IEEE, 2012, pp. 1324–1329. ISBN: 1467317365 (cit. on p. 25).

[35] Masato Nagayoshi, Hajimne Murao, and Hisashi Tamaki. "A State Space Filter for Reinforcement Learning in POMIDPs-Application to a Continuous State Space". In: *SICE-ICASE, 2006. International Joint Conference*. IEEE, 2006, pp. 6037–6042. ISBN: 8995003847 (cit. on p. 25).

[36] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "Grasping pomdps". In: *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 4685–4692. ISBN: 1424406021 (cit. on p. 26).

[37] Tomas Lozano-Perez, Matthew T Mason, and Russell H Taylor. "Automatic synthesis of fine-motion strategies for robots". In: *The International Journal of Robotics Research* 3.1 (1984), pp. 3–24. ISSN: 0278-3649 (cit. on p. 26).

[38] Dinesh K Pai and Ming C Leu. "Uncertainty and compliance of robot manipulators with applications to task feasibility". In: *The International journal of robotics research* 10.3 (1991), pp. 200–213. ISSN: 0278-3649 (cit. on p. 26).

[39] Arash Ajoudani et al. "A manipulation framework for compliant humanoid coman: Application to a valve turning task". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2014, pp. 664–670. ISBN: 147997174X (cit. on p. 27).

[40] Pietro Balatti et al. "A self-tuning impedance controller for autonomous robotic manipulation". In: *2018 IEEE/RSJ International Conference on*

Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 5885–5891. ISBN: 1538680947 (cit. on p. 27).

[41] Hee-Chan Song, Young-Loul Kim, and Jae-Bok Song. "Automated guidance of peg-in-hole assembly tasks for complex-shaped parts". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 4517–4522. ISBN: 1479969346 (cit. on p. 29).

[42] Ker-Jiun Wang. "Fuzzy Sliding Mode Joint Impedance Control for a tendon-driven robot hand performing peg-in-hole assembly". In: *Robotics and Biomimetics (ROBIO), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2087–2092. ISBN: 1509043640 (cit. on p. 29).

[43] Michael Jokesch et al. "Generic Algorithm for Peg-In-Hole Assembly Tasks for Pin Alignments with Impedance Controlled Robots". In: *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 105–117 (cit. on p. 29).

[44] Bo Wei et al. "An Improved Variable Spring Balance Position Impedance Control for a Complex Docking Structure". In: *International Journal of Social Robotics* 8.5 (2016), pp. 619–629. ISSN: 1875-4791 (cit. on p. 29).

[45] Yanjiang Huang et al. "Vision-guided peg-in-hole assembly by Baxter robot". In: *Advances in Mechanical Engineering* 9.12 (2017), p. 1687814017748078. ISSN: 1687-8140 (cit. on p. 30).

[46] Wen-Chung Chang and Chia-Hung Wu. "Automated USB peg-in-hole assembly employing visual servoing". In: *Control, Automation and Robotics*

(ICCAR), 2017 3rd International Conference on. IEEE, 2017, pp. 352–355. ISBN: 150906088X (cit. on p. 30).

[47] Billibon H Yoshimi and Peter K Allen. "Active, uncalibrated visual servoing". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 156–161. ISBN: 0818653302 (cit. on p. 30).

[48] Fares J Abu-Dakka et al. "Solving peg-in-hole tasks by human demonstration and exception strategies". In: *Industrial Robot: An International Journal* 41.6 (2014), pp. 575–584. ISSN: 0143-991X (cit. on p. 30).

[49] LL Lin et al. "Peg-in-Hole assembly under uncertain pose estimation". In: *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*. IEEE, 2014, pp. 2842–2847. ISBN: 1479958255 (cit. on p. 30).

[50] Zhimin Hou et al. "The learning-based optimization algorithm for robotic dual peg-in-hole assembly". In: *Assembly Automation* 38.4 (2018), pp. 369–375. ISSN: 0144-5154 (cit. on p. 30).

[51] Robert Andre, Michael Jokesch, and Ulrike Thomas. "Reliable robot assembly using haptic rendering models in combination with particle filters". In: *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1134–1139. ISBN: 1509024093 (cit. on p. 30).

[52] Jianhua Su et al. "Sensor-less insertion strategy for an eccentric peg in a hole of the crankshaft and bearing assembly". In: *Assembly Automation* 32.1 (2012), pp. 86–99. ISSN: 0144-5154 (cit. on p. 30).

[53] Gregory Kahn et al. "Uncertainty-aware reinforcement learning for collision avoidance". In: *arXiv preprint arXiv:1702.01182* (2017) (cit. on pp. 31, 48).

[54] William R Clements et al. "Estimating risk and uncertainty in deep reinforcement learning". In: *arXiv preprint arXiv:1905.09638* (2019) (cit. on pp. 32, 48).

[55] Frank H Knight. *Risk, uncertainty and profit*. Courier Corporation, 2012. ISBN: 0486147932 (cit. on pp. 32, 47).

[56] Jarryd Martin et al. "Count-based exploration in feature space for reinforcement learning". In: *arXiv preprint arXiv:1706.08090* (2017) (cit. on p. 32).

[57] Haruhiko Asada. "Teaching and learning of compliance using neural nets: Representation and generation of nonlinear compliance". In: *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE, 1990, pp. 1237–1244. ISBN: 0818690615 (cit. on pp. 33, 46).

[58] Andrew G Barto, Richard S Sutton, and Charles W Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846. ISSN: 0018-9472 (cit. on p. 33).

[59] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. "Acquiring robot skills via reinforcement learning". In: *IEEE Control Systems* 14.1 (1994), pp. 13–24. ISSN: 1066-033X (cit. on p. 33).

[60] Yuke Zhu et al. "Reinforcement and imitation learning for diverse visuomotor skills". In: *arXiv preprint arXiv:1802.09564* (2018) (cit. on p. 33).

[61] Aravind Rajeswaran et al. “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations”. In: *arXiv preprint arXiv:1709.10087* (2017) (cit. on p. 34).

[62] Ivaylo Popov et al. “Data-efficient deep reinforcement learning for dexterous manipulation”. In: *arXiv preprint arXiv:1704.03073* (2017) (cit. on pp. 34, 40).

[63] Mrinal Kalakrishnan et al. “Learning force control policies for compliant manipulation”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4639–4644. ISBN: 1612844561 (cit. on p. 34).

[64] LI Zhijun et al. “Reinforcement Learning of Manipulation and Grasping using Dynamical Movement Primitives for a Humanoid-like Mobile Manipulator”. In: *IEEE/ASME Transactions on Mechatronics* (2017). ISSN: 1083-4435 (cit. on p. 34).

[65] Wonchul Kim, Chungkeun Lee, and H Jin Kim. “Learning and Generalization of Dynamic Movement Primitives by Hierarchical Deep Reinforcement Learning from Demonstration”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3117–3123. ISBN: 1538680947 (cit. on p. 34).

[66] Ivan SK Lee and Henry YK Lau. “Adaptive state space partitioning for reinforcement learning”. In: *Engineering applications of artificial intelligence* 17.6 (2004), pp. 577–588. ISSN: 0952-1976 (cit. on p. 35).

[67] Freek Stulp et al. “Learning to grasp under uncertainty”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 5703–5708. ISBN: 1612843859 (cit. on p. 35).

[68] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. “Robot motor skill coordination with EM-based reinforcement learning”. In: *2010 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2010, pp. 3232–3237. ISBN: 1424466768 (cit. on p. 35).

[69] Shixiang Gu et al. “Continuous deep q-learning with model-based acceleration”. In: *International Conference on Machine Learning*. 2016, pp. 2829–2838 (cit. on p. 35).

[70] Ignasi Clavera, David Held, and Pieter Abbeel. “Policy transfer via modularity and reward guiding”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1537–1544. ISBN: 1538626829 (cit. on pp. 35, 37).

[71] Tuomas Haarnoja et al. “Composable deep reinforcement learning for robotic manipulation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6244–6251. ISBN: 1538630818 (cit. on p. 36).

[72] Coline Devin et al. “Learning modular neural network policies for multi-task and multi-robot transfer”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2169–2176. ISBN: 150904633X (cit. on p. 36).

[73] Sergey Levine et al. “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”. In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436. ISSN: 0278-3649 (cit. on p. 37).

[74] Aditya Gudimella et al. “Deep reinforcement learning for dexterous manipulation with concept networks”. In: *arXiv preprint arXiv:1709.06977* (2017) (cit. on pp. 37, 41).

[75] Jan Matas, Stephen James, and Andrew J Davison. “Sim-to-real reinforcement learning for deformable object manipulation”. In: *arXiv preprint arXiv:1806.07851* (2018) (cit. on p. 37).

[76] Fotios Dimeas and Nikos Aspragathos. “Reinforcement learning of variable admittance control for human-robot co-manipulation”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 1011–1016. ISBN: 1479999946 (cit. on p. 37).

[77] Bianca Sangiovanni et al. “Deep reinforcement learning for collision avoidance of robotic manipulators”. In: *2018 European Control Conference (ECC)*. IEEE, 2018, pp. 2063–2068. ISBN: 3952426989 (cit. on pp. 37–39).

[78] Nil Stolt Anso et al. “Deep Reinforcement Learning for Pellet Eating in Agar. io”. In: (2019) (cit. on p. 38).

[79] Zhixiang Wang et al. “Non-local Self-attention Structure for Function Approximation in Deep Reinforcement Learning”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 3042–3046. ISBN: 1479981311 (cit. on p. 38).

[80] Sergey Levine et al. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373. ISSN: 1532-4435 (cit. on pp. 38, 48).

[81] Donghoon Baek et al. “Path planning for automation of surgery robot based on probabilistic roadmap and reinforcement learning”. In: *2018 15th International Conference on Ubiquitous Robots (UR)*. IEEE, 2018, pp. 342–347. ISBN: 1538663341 (cit. on p. 39).

[82] Garrett Thomas et al. “Learning robotic assembly from CAD”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9. ISBN: 1538630818 (cit. on p. 39).

[83] Shengjia Shao et al. “Towards hardware accelerated reinforcement learning for application-specific robotic control”. In: *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–8. ISBN: 1538674793 (cit. on p. 39).

[84] Maja J Mataric. “Reward functions for accelerated learning”. In: *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 181–189 (cit. on pp. 40, 46).

[85] Adam Laud and Gerald DeJong. “The influence of reward on the speed of reinforcement learning: An analysis of shaping”. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 440–447 (cit. on p. 40).

[86] Javier Garcia and Fernando Fernandez. “A comprehensive survey on safe reinforcement learning”. In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480 (cit. on p. 40).

[87] Marek Grzes and Daniel Kudenko. "Plan-based reward shaping for reinforcement learning". In: *2008 4th International IEEE Conference Intelligent Systems*. Vol. 2. IEEE, 2008, pp. 10-22-10–29. ISBN: 1424417392 (cit. on p. 40).

[88] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. Report. 2004 (cit. on p. 40).

[89] John Asmuth, Michael L Littman, and Robert Zinkov. "Potential-based Shaping in Model-based Reinforcement Learning". In: *AAAI*. 2008, pp. 604–609 (cit. on p. 40).

[90] Bhaskara Marthi. "Automatic shaping and decomposition of reward functions". In: *Proceedings of the 24th International Conference on Machine learning*. ACM, 2007, pp. 601–608. ISBN: 1595937935 (cit. on p. 40).

[91] George Konidaris and Andrew Barto. "Autonomous shaping: Knowledge transfer in reinforcement learning". In: *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 489–496. ISBN: 1595933832 (cit. on p. 40).

[92] Sam Michael Devlin and Daniel Kudenko. "Dynamic potential-based reward shaping". In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, 2012, pp. 433–440 (cit. on p. 40).

[93] Eric Wiewiora. "Potential-based shaping and Q-value initialization are equivalent". In: *Journal of Artificial Intelligence Research* 19 (2003), pp. 205–208. ISSN: 1076-9757 (cit. on p. 40).

[94] Pieter Abbeel and Andrew Y Ng. "Apprenticeship learning via inverse reinforcement learning". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1. ISBN: 1581138385 (cit. on p. 41).

[95] Andrew Y Ng and Stuart J Russell. "Algorithms for inverse reinforcement learning". In: *Icml*. Vol. 1. 2000, p. 2 (cit. on p. 41).

[96] Tadanobu Inoue et al. "Deep reinforcement learning for high precision assembly tasks". In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 819–825. ISBN: 1538626829 (cit. on pp. 43, 46–48).

[97] Beltran-Hernandez Cristian Camilo et al. "Variable Compliance Control for Robotic Peg-in-Hole Assembly: A Deep Reinforcement Learning Approach". In: *arXiv preprint arXiv:2008.10224* (2020) (cit. on p. 43).

[98] Zhaolong Gao et al. "Continuous shared control in prosthetic hand grasp tasks by Deep Deterministic Policy Gradient with Hindsight Experience Replay". In: *International Journal of Advanced Robotic Systems* 17.4 (2020), p. 1729881420936851. ISSN: 1729-8814 (cit. on p. 43).

[99] Vincent Francois-Lavet et al. "An introduction to deep reinforcement learning". In: *arXiv preprint arXiv:1811.12560* (2018) (cit. on p. 48).

[100] Isabelle Guyon et al. "Result analysis of the NIPS 2003 feature selection challenge". In: *Advances in neural information processing systems*. 2005, pp. 545–552 (cit. on p. 48).

[101] Nojun Kwak and Chong-Ho Choi. "Input feature selection for classification problems". In: *IEEE transactions on neural networks* 13.1 (2002), pp. 143–159. ISSN: 1045-9227 (cit. on p. 48).

[102] Xin Xu, Dewen Hu, and Xicheng Lu. "Kernel-based least squares policy iteration for reinforcement learning". In: *IEEE Transactions on Neural Networks* 18.4 (2007), pp. 973–992. ISSN: 1045-9227 (cit. on p. 48).

[103] J Zico Kolter and Andrew Y Ng. "Regularization and feature selection in least-squares temporal difference learning". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 521–528 (cit. on p. 48).

[104] Carlos Diuk, Lihong Li, and Bethany R Leffler. "The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 249–256 (cit. on p. 48).

[105] Richard S Sutton. "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In: *Advances in neural information processing systems*. 1996, pp. 1038–1044 (cit. on p. 49).

[106] Hado Van Hasselt and Marco A Wiering. "Reinforcement learning in continuous action spaces". In: *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE, 2007, pp. 272–279. ISBN: 1424407060 (cit. on p. 49).

[107] Matthias Plappert et al. "Parameter space noise for exploration". In: *arXiv preprint arXiv:1706.01905* (2017) (cit. on p. 49).

[108] Tom Schaul et al. “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (2015) (cit. on p. 53).

[109] Kamal Sharma, Varsha Shirwalkar, and Prabir K Pal. “Intelligent and environment-independent peg-in-hole search strategies”. In: *2013 International Conference on Control, Automation, Robotics and Embedded Systems (CARE)*. IEEE, 2013, pp. 1–6. ISBN: 1467361534 (cit. on p. 94).

Appendix: Author Contribution Statements

T. Rouillard, I. Howard, and L. Cui, "Progressive Training of a Deep Reinforcement Learning Agent for a Peg-in-Hole Task with Goal Uncertainties," in *Journal of Intelligent Robotic Systems*. (**Under review**)

	Conception & design	Theoretical study & programming	Prototype development & validation	Manuscript writing, revision & finalisation	Total % contribution
Thibault Rouillard	90%	100%	100%	70%	90%
Thibault Rouillard Acknowledgment: I acknowledge that these represent my contribution to the above research output Signed:					
Ian Howard	0%	0%	0%	10%	2.5%
Ian Howard Acknowledgment: I acknowledge that these represent my contribution to the above research output Signed:					
Lei Cui	10%	0%	0%	20%	7.5%
Lei Cui Acknowledgment: I acknowledge that these represent my contribution to the above research output Signed:					
Total %	100%	100%	100%	100%	100%

T. Rouillard, I. Howard, and L. Cui, "Autonomous Two-Stage Object Retrieval Using Supervised and Reinforcement Learning," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2019: IEEE, pp. 780-786.

	Conception & design	Theoretical study & programming	Prototype development & validation	Manuscript writing, revision & finalisation	Total % contribution
Thibault Rouillard	90%	100%	100%	70%	90%
Thibault Rouillard Acknowledgment: I acknowledge that these represent my contribution to the above research output Signed:					
Ian Howard	0%	0%	0%	10%	2.5%
Ian Howard Acknowledgment: I acknowledge that these represent my contribution to the above research output Signed:					
Lei Cui	10%	0%	0%	20%	7.5%
Lei Cui Acknowledgment: I acknowledge that these represent my contribution to the above research output Signed:					
Total %	100%	100%	100%	100%	100%

Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged.