THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Robot Learning for Manipulation of Deformable Linear Objects

Rita Laezza

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2021

Robot Learning for Manipulation of Deformable Linear Objects

Rita Laezza

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000
www.chalmers.se

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice.
Gothenburg, Sweden, 2021

I wish this was a masterpiece...
an epic, a painting or a symphony.
Perhaps then it would be worthy
of my beloved who rest in peace.

But this is just a thesis,
with facts instead of brush strokes,
with math in place of musical notes
and no poetry, just this.

Dedicated to the memory of my grandmothers,
Maria José (1932-2020) and Matilde (1931-2021).
Forever my ***avó Zezita*** *and* ***nonna Tilde****.*

Abstract

Deformable Object Manipulation (DOM) is a challenging problem in robotics. Until recently there has been limited research on the subject, with most robotic manipulation methods being developed for rigid objects. Part of the challenge in DOM is that non-rigid objects require solutions capable of generalizing to changes in shape and mechanical properties. Recently, Machine Learning (ML) has been proven successful in other fields where generalization is important such as computer vision, thus encouraging the application of ML to robotics as well. Notably, Reinforcement Learning (RL) has shown promise in finding control policies for manipulation of rigid objects. However, RL requires large amounts of data that are better satisfied in simulation while deformable objects are inherently more difficult to model and simulate.

This thesis presents ReForm, a simulation sandbox for robotic manipulation of Deformable Linear Objects (DLOs) such as cables, ropes, and wires. DLO manipulation is an interesting problem for a variety of applications throughout manufacturing, agriculture, and medicine. Currently, this sandbox includes six shape control tasks, which are classified as explicit when a precise shape is to be achieved, or implicit when the deformation is just a consequence of a more abstract goal, e.g. wrapping a DLO around another object. The proposed simulation environments aim to facilitate comparison and reproducibility of robot learning research. To that end, an RL algorithm is tested on each simulated task providing initial benchmarking results. ReForm is one of three concurrent frameworks to first support DOM problems.

This thesis also addresses the problem of DLO state representation for an explicit shape control problem. Moreover, the effects of elastoplastic properties on the RL reward definition are investigated. From a control perspective, DLOs with these properties are particularly challenging to manipulate due to their nonlinear behavior, acting elastic up to a yield point after which they become permanently deformed. A low-dimensional representation from discrete differential geometry is proposed, offering more descriptive shape information than a simple point-cloud while avoiding the need for curve fitting. Empirical results show that this representation leads to a better goal description in the presence of elastoplasticity, preventing the RL algorithm from converging to local minima which correspond to incorrect shapes of the DLO.

Keywords: Robotics, Robot Learning, Reinforcement Learning, Deformable Object Manipulation, Deformable Linear Objects.

List of Publications

This thesis is based on the following publications:

[A] **Rita Laezza**, Yiannis Karayiannidis, "Learning Shape Control of Elastoplastic Deformable Linear Objects". 2021 IEEE International Conference on Robotics and Automation (ICRA).

[B] **Rita Laezza***, Robert Gieselmann*, Florian T. Pokorny, Yiannis Karayiannidis, "ReForm: A Robot Learning Sandbox for Deformable Linear Object Manipulation". 2021 IEEE International Conference on Robotics and Automation (ICRA).

Other publications by the author, not included in this thesis, are:

[C] **R. Laezza**, Y. Karayiannidis, "Shape Control of Elastoplastic Deformable Linear Objects through Reinforcement Learning". *Workshop on Robotic Manipulation of Deformable Objects (ROMADO) at 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* October 25-29, 2020, Las Vegas, NV, USA (Virtual).

[D] **R. Laezza**, R. Gieselmann, F. T. Pokorny, Y. Karayiannidis, "Presenting ReForm: A Robot Learning Sandbox for Deformable Linear Object Manipulation". *Workshop on Representing and Manipulating Deformable Objects (RMDO) at 2021 IEEE International Conference on Robotics and Automation (ICRA)* May 31 - June 7, 2021, Xi'an, China (Hybrid).

Acknowledgments

I would like to begin by expressing the deepest gratitude to my supervisor, Yiannis Karayiannidis, for his patient guidance and support during this first half of my doctoral studies. I am looking forward to our continued collaboration for the remainder of this journey.

From KTH, I would like to thank my co-supervisor, Florian Porkorny, and particularly my PhD collaborator, Robert Gieselmann, alongside whom I have had the pleasure to share the experiences of graduate education. This collaboration of course would not be possible without WASP — Wallenberg AI, Autonomous Systems and Software Program — which brought us all together.

From WASP, I want to acknowledge the Professors involved in the graduate school courses and trips, which provided invaluable knowledge and enabled wonderful experiences that I will always cherish. I would also like to thank my fellow WASP students from the first AI batch, with whom I have had some of the most interesting and entertaining discussions.

Finally, I must thank my family for their moral support and encouragement during the toughest times, as well as joint celebration during the happiest ones. Last but not least, I want to thank my dearest Karl, who has given me confidence to continue when I needed it the most.

Acronyms

AI:	Artificial Intelligence
AL:	Apprenticeship Learning
ANN:	Artificial Neural Network
CNN:	Convolutional Neural Network
CV:	Computer Vision
DDOD:	Dense Depth Object Descriptor
DDPG:	Deep Deterministic Policy Gradient
DL:	Deep Learning
DLO:	Deformable Linear Object
DNN:	Deep Neural Network
DoF:	Degree of Freedom
DOM:	Deformable Object Manipulation
DP:	Dynamic Programming
DRL:	Deep Reinforcement Learning
IL:	Imitation Learning
LfD:	Learning from Demonstrations
MC:	Monte Carlo
MDP:	Markov Decision Process
ML:	Machine Learning
MLP:	Multilayer Perceptron
NLP:	Natural Language Processing
PG:	Policy Gradient
RGB:	Red Green Blue
RL:	Reinforcement Learning
RoL:	Robot Learning
TD:	Temporal Difference

Contents

Part I

Overview

CHAPTER 1

Introduction

At present, most robotic manipulators are confined to industrial settings, where their environment is highly controlled, and humans are generally not allowed near. This is because many of these robots are programmed to operate in a predefined manner and will not perceive an unexpected human in their path, possibly injuring anyone who gets in the way. Recently, the field of collaborative robotics has started to gain momentum, striving for closer interaction between humans and robots. To that effect, a lot of research has focused on making robots more adaptive in order to increase safety, by taking into account the unpredictability of unstructured environments and of human behavior. As this cooperation becomes more common, robots are beginning to spread into other sectors, such as healthcare, agriculture and the service industry. Each sector adds to the variability in tasks and manipulated objects leading to new challenges. This thesis addresses one of them, namely: Deformable Object Manipulation (DOM).

The ability to handle non-rigid objects is particularly important since so many human tasks require skilled manipulation of deformable materials. This includes the handling of suturing threads in healthcare; the harvesting of crops in agriculture; and cooking in the service industry. Note that in each

of these examples, there is a considerable amount of variation both in the objects being handled and in the techniques for executing a certain type of task. Looking closer at the example of surgical suturing, there are differences in tissue properties between individuals, as well as within a single individual. Suturing vessels is significantly different from suturing skin. In addition, the type of needle and thread may also vary in resistance, plasticity and elasticity.

Often, when faced with the job of automating a DOM task, engineers design specialized tools or machines. In agriculture, automated milking systems used in dairy farms are an example of this approach. Even though this leads to efficient results, it requires a unique apparatus for each task. Humans on the other hand, can learn to execute a large set of tasks with their own hands. One goal of robotics research is therefore to create general purpose collaborative robots which are capable of dexterous manipulation akin to that of humans. Contrary to industrial robots which are intended for repetitive production at superhuman speed, strength and precision, collaborative robots have to operate in a safe, gentle and adaptive manner.

Robotic manipulation tasks are routinely solved by first deriving a model of the object dynamics and then using it for control. However, deformable objects have complex nonlinear dynamics, which makes modeling more challenging. Furthermore, due to the wide range of both mechanical and geometrical properties, these objects constitute a large and heterogeneous class. Going back to the cooking example, since the dynamics of an egg cannot describe a bacon strip, one would need to derive a model of each type of food being manipulated. Alternatively, Machine Learning (ML) methods can be used to obtain modeling parameters or even to learn a control policy without the need for an explicit model. Furthermore, ML can be employed either to imitate human manipulation or to go beyond that by learning through experience. These methods fall under the umbrella of Robot Learning (RoL).

The papers presented in Part II lie at the intersection of deformable object manipulation and robot learning. More specifically, DOM is the research problem of interest and RoL is the family of methods which are applied to address it. The goal of Part I is to provide an overview of these two fields, needed to understand the research context. Therefore, it is meant to be a complement to the papers, not a repetition. This chapter starts with the wider context, where Section 1.1 presents the field of DOM and Section 1.2 continues with RoL. Section 1.3 follows with an outline of this thesis.

1.1 Deformable Object Manipulation

Deformable Object Manipulation (DOM) is a rapidly growing field of robotics. It encompasses manipulation problems in which the object being manipulated undergoes non-negligible deformation. To date, the majority of robotics research has been limited to rigid object manipulation, thus requiring the assumption that objects would not change shape. Naturally, this has narrowed the scope of robotic applications, leaving many industrial tasks involving deformable objects still to be performed by humans. One of the most compelling evidence to this fact is an example from the automotive industry, which started to become automated already in the early 1970s, but until recently nearly 100% of the installment of car wiring systems was done manually [1].

Besides benefiting a variety of industrial applications, there are other sectors which stand to gain from advances in DOM, as highlighted in the previous section. However, deformation brings several technical challenges, which according to Zhu et al. [2] can be summarized as:

- deformation is complex to model
- it is difficult to sense deformations
- deformable objects have an infinite number of degrees of freedom

Though these challenges are shared by all deformable objects, depending on the type of object they can have varying degrees of impact on the manipulation problem. For example, sensing deformation of a rope is particularly difficult because of its *low compression strength*, i.e. low resistance when two opposing points are pushed together. While grasping one end of the rope, very little information can be deduced about the rest of its shape through force-torque measurements alone. The same is not true for a metal wire with *high strain*, i.e. high resistance when it starts to deform. Since the material opposes deformation, there will be high forces which can be measured through force-torque sensors. This difference led Sanchez et al. [3] to categorize deformable objects according to these two physical properties[1], as shown in Figure 1.1.

Another property by which deformable objects were grouped in [3] is their approximate geometry. The rope and wire mentioned above are examples of

[1] In [3], low compression strength is referred to as *no compression strength*, and high strain is referred to as *large strain*.

Deformable Linear Objects (DLOs)[2], which are characterised by having one dimension considerably larger than the other two. More types of *linear* objects include cables and threads. Objects can also be considered *planar*, when one dimension is considerably smaller than the other two, e.g. metal sheets, clothes and paper. Finally, *volumetric* objects have no dimension significantly larger/smaller than the others, much like the sponge in Figure 1.1.

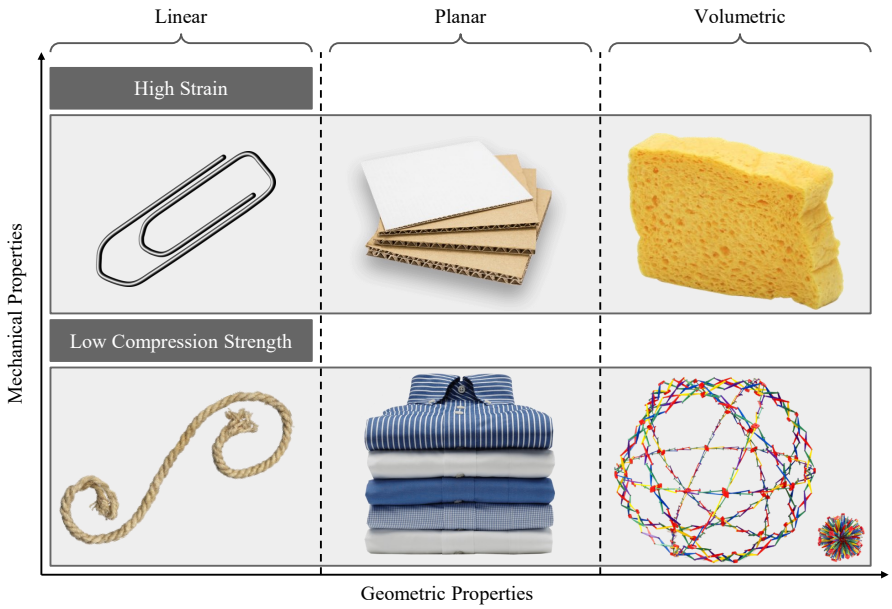


Figure 1.1: Classification of deformable objects proposed by Sanchez [3]. Based on mechanical properties, objects may have *high strain* or *low compression strength*. Based on an approximated geometry, these may be *linear*, *planar* or *volumetric*[3].

Attempting to find a manipulation strategy which generalizes to all classes presented in Figure 1.1, would require nothing short of human level intelligence and dexterity. Most research has therefore been focused on specific classes and even more often, on a particular task [3]. This thesis will focus on DLO manipulation, which will be covered in Chapter 2.

[2] Sometimes referred to as Deformable One-dimensional Objects (DOOs).
[3] Though a Hoberman sphere is given as the volumetric example, it is technically not a deformable object but rather an articulated structure, with low compression strength.

1.2 Robot Learning

Robot learning can be defined as the application of machine learning to robotic tasks. However, this is a very broad definition since robotics is an interdisciplinary field, and ML can be applied to many different problems: **i.** for human-robot interaction through speech one can use advances from Natural Language Processing (NLP); **ii.** to endow robots with visual comprehension, there are methods from Computer Vision (CV); **iii.** to find model parameters based on sampled data, there is model learning; **iv.** to be able to copy a human skill, Learning from Demonstrations (LfD) can be implemented; **v.** and in order to have robots learn directly by interacting with the environment, there is Reinforcement Learning (RL). Note that while the first two research areas are relevant for robotics, the last three are the most closely associated with RoL [4]. Figure 1.2 illustrates the RoL landscape with its three central approaches which will be described below.

Model Learning When ML is used for system identification, this is referred to as model learning. While classical approaches use statistical methods to learn specific classes of mathematical models, ML algorithms aim to learn more general mappings from inputs to outputs, which do not require the same amount of laborious hand-tuning. There are two main types of models which can be learned: *forward* and *inverse*. The former aim at predicting the evolution of the system based on the current observation or a history of past observations. These predictions can then be used for control, which is what the latter type of model does directly. That is, inverse models attempt to predict the input required to achieve a desired output [4].

Learning from Demonstrations LfD includes two main strategies, namely Imitation Learning (IL) and Apprenticeship Learning (AL) [4]. In IL, which is also sometimes referred to as behavioral cloning, the robot estimates a policy from a teacher's demonstration in order to reproduce it. In AL, on the other hand, a reward function is used to assign scores to the demonstrations, with the intent to encode the true objective and go beyond pure imitation. This reward function needs to be chosen such that a perfect score is attributed to an optimal demonstration. Based on this reward function it is then possible to find the optimal policy, through RL. Consequently, this approach is also known as inverse reinforcement learning [4].

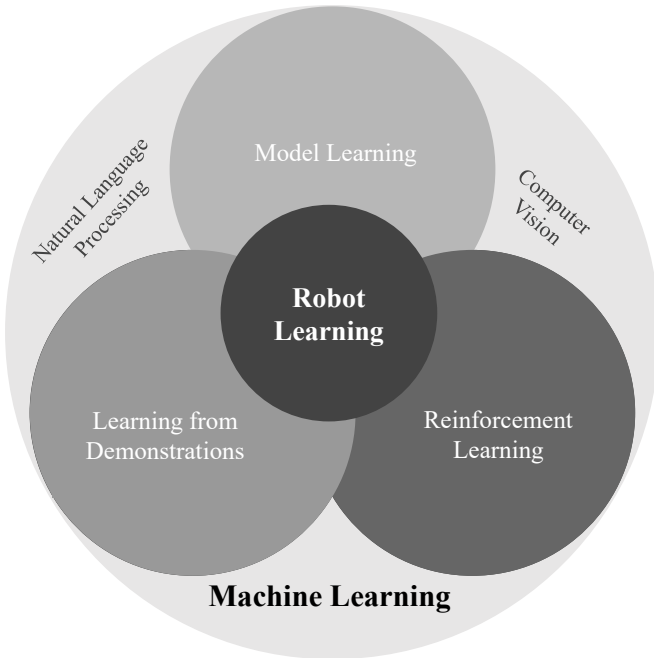


Figure 1.2: Overview of the RoL landscape, with three main research areas: learning from demonstrations, model learning and reinforcement learning. ML has also found many useful applications in robotics, from other research fields such as CV and NLP.

Reinforcement Learning Model learning and IL approaches mainly use algorithms from the supervised learning branch of ML, i.e. learn a mapping function based on ground truth data. RL, on the other hand, is considered to be a separate branch which aims to learn by trial-and-error. Contrary to supervised learning, RL is centered around an agent which interacts with the environment and makes decisions that influence what data it receives from future experience. Moreover, while in supervised learning there are ground truth labels, in RL there is only a scalar reward signal which the agent attempts to maximize. This reward is also what sets it apart from unsupervised learning, where no signal of correctness is given. Although all RoL techniques can be applied to DOM problems, this thesis will focus on reinforcement learning in particular. Hence, Chapter 3 will introduce RL theory in greater detail.

After this overview of the three central RoL approaches, it is important to ask the question: why should one use any of them? According to Connell and Mahadevan [5], RoL is particularly important for problems in which:

- the environment is nonstationary
- it may be prohibitively hard to program a robot
- not all situations, as well as goals, may be foreseeable

Taking a particular example from DOM, let us consider a task we are all familiar with: folding laundry. Programming a robot to fold our clothes is extremely challenging. Since these are low compression strength materials, sensing has to rely mostly on vision data. This, coupled with the fact that the clothes we own change over time, make it a nonstationary environment. Furthermore, there is high variability in the geometrical and fabric properties of garments, which potentially requires a unique robot program for each individual item, i.e. the steps to fold a cotton t-shirt are not the same as to fold a pair of leather pants. Therefore, hard-coded approaches may quickly become impractical. Finally, the state of our laundry when taken straight out of the dryer is truly unforeseeable.

Unsurprisingly, the first complete pipeline for autonomous folding of clothes using a dual-armed robot proposed in [6] made use of several ML techniques to achieve a 79% success rate, for a small subset of garments. It should be noted that the grippers used in this work were specialized for handling clothes [7]. Besides RoL software, it is believed that another major bottleneck in DOM is the hardware [2]. This includes not only the robot, but also the grippers and sensors chosen for a particular task. Learning itself may become much simpler depending on the type of sensing and actuation used while interacting with the environment.

The work by Doumanoglou et al. [6] is evidence that ML plays an important role in DOM. Nevertheless, their approach applied mostly CV algorithms to solve subproblems of the task, leaving the main RoL techniques to be explored. Generally, robotic manipulation approaches fall in-between two extremes: either a single RoL method is applied in an *end-to-end* approach, or a *modular* framework is implemented using different algorithms to solve each individual subproblem. This work focuses on applying RL to learn robot-agnostic control policies, assuming a modular implementation with CV techniques for tracking the DLO and inverse kinematics algorithms for control.

1.3 Thesis Outline

This Licentiate thesis presents progress towards the graduate project funded by WASP **to develop Artificial Intelligence (AI) and learning-based approaches for robotic manipulation of deformable objects**. The project can be divided into two separate components, one which refers to the *core technology* of AI, and the other to the DOM *application*. Initially, this was to be achieved by combining:

- data-driven modeling of robot-object interaction, based on Deep Neural Networks (DNNs) and vision/force data.
- design of control policies based on reinforcement learning principles and testing them in real-world and simulated robotic setups.

This helped reduce the focus for the *core technology* to RL. As highlighted in Section 1.1, DOM is a large and heterogeneous problem which would be difficult to tackle all at once, so the *application* was narrowed to deformable linear object manipulation. Therefore, the rest of Part I is organized into two main chapters: Chapter 2 introduces relevant related work on DLO manipulation, and Chapter 3 presents the theoretical background of RL. Finally, some concluding remarks are provided in Chapter 4.

Contributions After this overview, Part II consists of two contributed papers. Paper A addresses the challenges of state representation and reward definition when applying RL methods to a DLO shape control problem, with elastoplastic properties. In particular, a discrete curvature representation is used to define a reward that better encodes details of the desired shape. Paper B presents ReForm, a new RoL sandbox to facilitate research on DLO manipulation. For this, the simulation environment used in the previous paper is compiled together with five additional tasks. A more detailed description of the contributions is provided in Section 4.1.

CHAPTER 2

Deformable Linear Object Manipulation

DLOs were one of the first classes of deformable objects to be studied within robotic manipulation. This is due in part to their comparatively simple geometry, with respect to planar and volumetric objects, which makes them computationally less expensive to simulate [3]. More importantly, DLOs are crucial components in numerous industrial, healthcare and service applications. Consider the countless electrical cables which are installed in electronic devices, and the variety of hoses and tubes found across so many industrial machines. Previous work on robotic DLO manipulation has tackled specific tasks, such as USB cable insertion [8], hot-wire cutting [9], knot tying [10] as well as untangling [11], wire harness assembly [12], surgical suturing [13], etc. Other works have focused on important subproblems such as state estimation [14], modeling [15] and shape/deformation control [16].

In this chapter, four key subproblems of robotic manipulation are introduced: **i.** modeling, **ii.** simulation, **iii.** sensing and **iv.** control. Section 2.1 gives an overview of modeling techniques used for DLO manipulation. Section 2.2 covers simulation of DLOs with an emphasis on software. This is followed by Section 2.3 where three major sensing modalities are summarized. Finally, the problem of control is addressed in Section 2.4.

2.1 Modeling

In DOM problems, modeling can be applied to describe either the instantaneous geometry of the objects or their evolving dynamics. The former is useful for defining a shape representation, while keeping dimensionality low. Dynamical models on the other hand, attempt to describe the behavior of the object when external forces are applied and are mostly used for prediction and control. Note that both geometric and dynamical models may be analytical or learned via ML methods. This section begins by presenting the problem of shape representation in Section 2.1.1, followed with an overview of modeling techniques for deformation physics in Section 2.1.2.

2.1.1 Shape Representation

In robotic manipulation, it is often sufficient to define the state of a rigid object by its pose vector in $\mathbb{R}^6$, i.e. position and orientation, as illustrated in Figure 2.1. If the geometry and weight of the object are also known, that provides a complete description. Conversely, there is no obvious choice of representation for a DLO since the state must also include information about its shape [17]. One straightforward option is to discretize the continuous object as a set of points in $\mathbb{R}^{N\times 3}$ i.e. a point cloud, also shown in the figure. Despite simple, this has a couple of drawbacks: Firstly, it is an approximation which will be increasingly accurate as $N \to \infty$, also leading to a larger state space; Secondly, it cannot describe torsion if needed, in which case the orientation of each point needs to be included in the state, further increasing its space to $\mathbb{R}^{N\times 6}$.

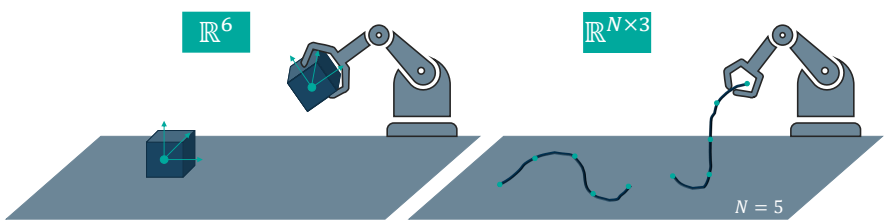


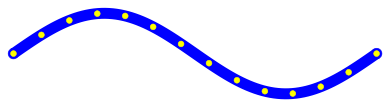
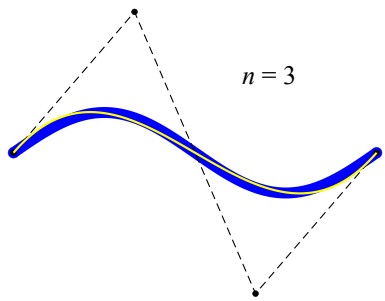
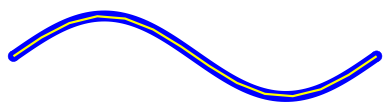
Figure 2.1: Comparison of state representations for a rigid object and a DLO. While the rigid cube can be represented by its pose, the DLO is approximated as a point cloud with N points.

Shape representation is intimately related to the state estimation technique used for a given task. For example, Tang et al. [10] developed a framework for DLO manipulation using Coherent Point Drift (CPD) for tracking the object. Since CPD is a point set registration method i.e. it aligns a point cloud with another, they discretized the DLO as a set of uniformly spaced points. Zea et al. [18] on the other hand, chose Bézier curves to represent the DLO for a Bayesian state estimation method. Yet another approach was used by Wnuk et al. [19] who represented the DLO's state by its skeleton line, from which a kinematic multi-body model was derived for control. These three representations are shown in Table 2.1, though there are many more summarized in [20], [21].

All aforementioned methods rely on vision data and therefore require several preprocessing steps of RGB and/or depth measurements. Segmentation, i.e. separating objects from the background, is particularly important for the success of DLO tracking. CPD only requires this image processing phase to obtain the point cloud representation. However, the other two methods employ continuous shape representations, further requiring a curve fitting algorithm. In [18] they construct a rectangle chain approximation of the Bézier curves, to simplify the fitting process. Much like many other algorithms, their work also relies on a dynamical model to better predict the object's state. Furthermore, in [19] the skeleton curve is modeled as a weighted sum of Radial Basis Functions (RBFs), for which they compute weights minimizing the error between the point cloud and the RBF. Conversely, in Paper A the point cloud is assumed to be the starting point for the proposed discrete representation, which does not require curve fitting.

Besides analytical modeling approaches, there have also been attempts to learn shape representations through black box models, e.g. Artificial Neural Networks (ANNs). Yan et al. [14] proposed a self-supervised learning approach for estimating the DLO as an ordered sequence of points. Alternatively, Sundaresan et al. [22] learned Dense Depth Object Descriptors (DDODs). Once a shape representation is obtained, it can be used either as **i.** a feedback signal or as **ii.** state variables. In the first case, the representation should be robust to noise. In the second case, the goal is to find a good trade-off between lower dimensionality and higher accuracy [2]. Finally, in end-to-end approaches a latent shape representation is implicitly learned directly from sensory data. Sensing modalities are discussed in Section 2.3.

Table 2.1: Examples of DLO state representations used in related work. This table presents the formal definition on the left and an illustrative plot on the right, where the DLO is drawn in blue and the representations in yellow. The cubic Bézier curve ($n = 3$), includes the set of $n + 1$ control points $\mathbf{p}_i$ and the respective Bézier polygon as the dashed line, in black.

A **point cloud** [10] represents the DLO as a discrete set of N points: $$X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \in \mathbb{R}^{N\times 3}$$ where $\mathbf{x}_n \in \mathbb{R}^3$ is the n-th point's position in Cartesian space.	
A **Bézier curve** [18] of order n is defined as a weighted sum of Bernstein polynomials, $B_i^n(u)$, with a set of $n+1$ control points, $\mathbf{p}_i$: $$\mathbf{B}(u) = \sum_{i=0}^{n} \mathbf{p}_i B_i^n(u)$$ where, $$B_i^n(u) = \binom{n}{i}(1-u)^{n-i}u^i$$ with $u \in [0, 1]$.	
A **skeleton line** [19] is modeled as a continuous spatial curve in Euclidean space, $$f(s) : \mathbb{R} \mapsto \mathbb{R}^3$$ where $s \in [0], [1]$ are local coordinates running along the DLO's length, L. The curve in turn is described by the Frenet-Ferret frame: $$\mathbf{R}(s) = [f'(s) \quad f''(s) \quad f'(s) \times f''(s)]$$	

2.1.2 Deformation Physics

Deformation occurs when an object changes shape due to the application of external forces. In addition, there are internal forces that neighbouring particles of a continuous material exert on each other, which are expressed as the stress, σ. When a force F is applied, this results in a dimensional change along the force direction, namely the strain ε. There is a wide range of deformation behavior depending on the material properties of an object. Stress tests are typically performed to analyse such properties, by applying tensile, compressive, bending, torsion or shear forces. Stress and strain are computed differently depending on the type of force and geometry of the object. For a tensile force, the stress and strain of a rod are given by:

$$\sigma = \frac{F}{A_0} \quad \text{and} \quad \varepsilon = \frac{L - L_0}{L_0}$$

where A_0 and L_0 are the original cross-sectional area and length, respectively and L is the measured length. Note that as the rod extends, its cross-sectional area will shrink since the volume of the object remains constant. The relationship between such axial and lateral strains is given by **Poisson's ratio**, ν [3]. Figure 2.2 shows an illustration of tensile stress tests applied to three rods with different material properties and the corresponding stress-strain curves.

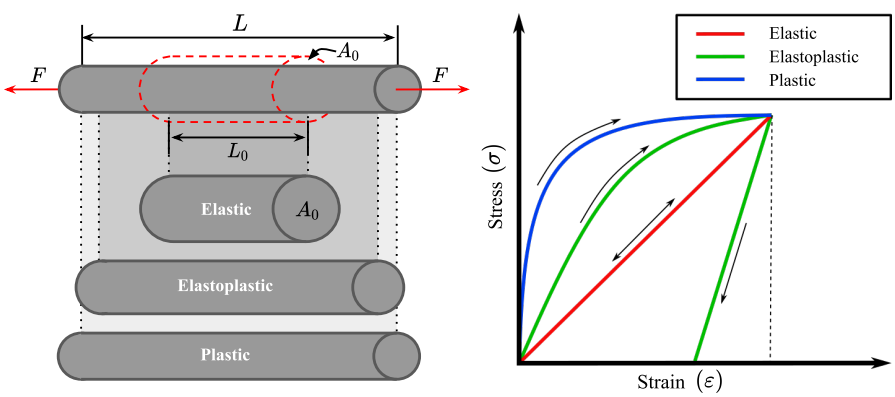


Figure 2.2: Illustration of different types of deformation. The stress-strain curves on the right show the behavior up to a certain strain, followed by the removal of the force. The resulting deformations are shown on the left.

When an object undergoes *elastic* deformation it returns to its original shape once the force is removed. An isotropic material, i.e. uniform in all orientations, with **Young's modulus** E, it is said to be linearly elastic if the strain is proportional to the stress, $\sigma = E\varepsilon$. In contrast, when an object undergoes *plastic* deformation it will become permanently deformed. An intermediate behavior is exhibited by *elastoplastic* materials which become permanently deformed after a certain threshold but are able to partially recover once the force is removed [3], as shown in Figure 2.2. These may act elastic within a certain range of stresses and plastic when a **yield point** is crossed. Once this happens, the resulting shape becomes strongly history-dependent. The effect of these properties are studied in Paper A. Finally, while for the aforementioned types of deformation the stress rate is of no importance, the same is not true for viscous materials. When the material is *viscoelastic*, the strain is a function of the stress rate. If this time dependent behavior is accompanied by permanent deformation, then the material is *viscoplastic* [23].

Due to this large variability, it becomes necessary to choose a modeling technique which can adequately describe an object's behavior. Other factors to consider are computational complexity, physical accuracy, and photo-realism [20]. Given their simplicity and computational efficiency, *discrete* models have been used extensively, of which mass-spring systems are the most common formulation. In [24], a DLO was modeled as a series of masses connected by linear, bending and torsion springs. However, discrete models are not as physically accurate as *continuum* models, which are more complex and therefore also more computationally heavy. As a consequence, these tend to be used outside of robotics for detailed material analysis e.g. finite element modeling of Warrington-Seale rope [25]. Finite Element Methods (FEM) are numerical techniques commonly used to solve partial differential equations of continuum-based models, by splitting the object into discrete elements that approximate its geometry [3]. Refer to [20] for a comprehensive overview.

Other works have used simpler energy-based models [15], [26]–[28], for simulating and controlling DLOs. Despite being physically inspired and computationally efficient, they do not explicitly model material properties. Recently, high-fidelity physics engines which support real-time execution have become readily available. That has enabled the use of more advanced models of DLOs which can be easily utilized out of the box. An overview of simulation using physics engines is presented in the next section.

2.2 Simulation

In the field of robotics, there are several simulators which are commonly used such as Gazebo [29], CoppeliaSim (f.k.a. V-REP) [30], PyBullet [31], and MuJoCo [32]. It is important to distinguish between a simulation software and the underlying physics engine. For example, both Gazebo and CoppeliaSim support Open Dynamics Engine (ODE) or Bullet as the physics engine, among others. On the other hand, MuJoCo and (Py)Bullet are standalone physics engines. Since they all offer rigid body kinematics and dynamics it is possible to approximate a DLO as a series of rigid links connected by ball-joints, which can be viewed as an underactuated robot.

Unfortunately, simulating more complex deformation is still limited [2]. MuJoCo 2.0 supports simulation of `composite` objects like particle systems, ropes, cloth and other soft bodies. A DLO may be simulated either as a 1D `grid` or a `rope` object. The former are strings of spheres connected by tendons with soft equality constraints for the length, which do not allow rotation. Alternatively, a `rope` is defined by a kinematic tree of bodies, for which the orientation can change. Compared to 1D grids, the rope simulation suffers from less jitter and can also use capsule and ellipsoid geometries [33]. Bullet supports similar soft body dynamics for cloth, rope and deformable volumes, using `btSoftBody` objects. To create a DLO, Bullet offers the `CreateRope` function in `btSoftBodyHelpers`, but more complex DLOs require a specialized implementation [34]. For both of these engines the mechanical properties of a DLO need to be set through constraint parameters connecting rigid bodies. Even though such parameters can be tuned to approximate the behavior of a real DLO, they are not based on deformation properties and are better suited to represent low compression strength materials.

Alternative simulators allow to set actual material properties, with parameters such as Young Modulus and Poisson's Ratio. This is true for the Simulation Open Framework Architecture (SOFA) [35] which was initially developed for medical applications and supports both mass-spring and FEM deformable models. SOFA has also been used for soft robotics, e.g. to control elastic soft robots [36]. AGX Dynamics, the simulator used in Part II, offers similar possibilities. There are two main DLO classes provided with this software: `Cable` and `Wire`. The former is aimed for fixed-length DLOs, which may exhibit elastoplastic behavior, while the latter is better fitted for DLOs where torsion is not relevant and length may vary [37].

Though the aforementioned simulators are the most represented in robotics research, there are many other options. For example, Blender is a 3D animation software that also supports Bullet for more advanced physics simulation. Sundaresan et al. [22] use Blender to generate synthetic depth data for rope manipulation through DDODs. Simulators used as game engines, such as NVIDIA PhysX and FleX, have also been applied to robotic contexts, with simulated environments like SoftGym [38], SAPIEN [39] and ThreeDWorld [40]. Section 3.6 covers simulation environments in the context of RL for robotic manipulation. Finally, there are also smaller scale simulators for specific applications, such as the Elastica [41] library based on Cosserat rod theory, which has been used with RL but is not intended for robotics.

2.3 Sensing

Sensing is considered as one of the most important areas of research when it comes to DOM [2]. There are several sensing modalities which may be used for robotic manipulation, e.g. vision, tactile, olfactory, thermal and force. According to Yin et al. [21] vision seems to be the predominant sensing modality in DOM applications. However, the choice of modality depends on the task, as well as the state representation which needs to be estimated. For example, with low compression strength DLOs which incur large deformations, vision is preferable to force-torque sensing since the latter only provides local information about the object's stiffness [2]. Conversely, for contact rich task where there may be occlusion by the gripper, perhaps tactile sensing is a better choice [42]. There are three main sensing modalities which have been used for DLO manipulation: vision, force-torque, and tactile. These will be presented below together with some related work.

Vision Naturally, when it comes to DOM applications, vision is an almost indispensable sensing modality. However, since vision data is high dimensional it is common to use tracking algorithms to obtain much smaller state representations of the object. Deformation increases the difficulty of object tracking considerably, but some algorithms have shown promising results for DLO manipulation, e.g [10], [12]. Alternatively, RGB and depth maps have been used directly in end-to-end implementations [43]. A unique challenge for DLO manipulation is the need to perceive topological information, such as knots and loops which lead to self-occlusion [11].

Force-torque One of the most commonly used sensing modalities in rigid object manipulation problems is force-torque sensing. Tough it does not provide much usable information for low compression strength objects, they may be useful for high strain DLOs [44]. Force-torque measurements can be obtained directly using sensors, or estimated based on proprioceptive signals, i.e. the joint torques of the robot.

Tactile The third and least mature modality is tactile sensing [45]. Even though there are many types of tactile inputs, in general they encode local information on the contact surface, such as forces, pressure or texture. There is large variability on the sensing technology used, and consequently also in the capabilities of the sensors. The most common approaches are based on optical e.g. GelSight [42] or capacitive technologies e.g. tactile sensors installed on PR2 [46], among others. The former have been successfully applied to DLO manipulation in [8], and shown to improve dexterity.

Vision, force-torque and tactile sensing may be combined for state estimation or identification of model parameters, through sensor fusion techniques.

2.4 Control

Robotic manipulators generally consist of a mechanical structure with a set of rigid links connected by a set of revolute and/or prismatic joints, each associated with a joint variable q which has to be controlled. Manipulators can be classified as Cartesian, cylindrical, SCARA (Selective Compliance Assembly Robot Arm), spherical or anthropomorphic. Typically a manipulator is also fitted with an end-effector specific for the task, e.g. gripper, suction plate, etc. [47]. Therefore, a control strategy for DLO manipulation has to address both the manipulation and grasping problems, which depend on the mechanical structure of the manipulator and end-effector. For the sake of simplicity, in this work the problem was reduced to the manipulation of one or two points on a DLO rigidly attached to end-effector(s), thus removing the grasping problem. Furthermore, to increase generality no particular manipulator was assumed and all control inputs are defined in *operational space*, i.e. in terms of positions, velocities or accelerations of the end-effector required to execute a task. This is in contrast with end-to-end approaches which attempt to embed the geometry of the robot into the learned controller, given that sensory inputs are mapped directly to joint variables.

Problem Formulation In this thesis, the control problem is formulated in terms of the desired velocity of a given controlled point on the DLO. This point is assumed to be fixed to the end-effector, thus sharing the same velocity $\boldsymbol{v}_e$. For an arbitrary n-DoF manipulator, the joint velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^n$, can be mapped to the end-effector velocity as:

$$\boldsymbol{v}_e = \begin{bmatrix} \dot{\boldsymbol{p}}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$$

where $\boldsymbol{J}(\boldsymbol{q})$ is the Jacobian matrix, and $\dot{\boldsymbol{p}}_e, \boldsymbol{\omega}_e \in \mathbb{R}^3$ are the end-effector's linear and angular velocity, respectively. Therefore the goal is to find a manipulator agnostic policy π that maps a given state $\boldsymbol{s}$, to a reference velocity $\boldsymbol{v}_e^{\text{ref}}$, in order to control the shape of a DLO:

$$\boldsymbol{v}_e^{\text{ref}} = \pi(\boldsymbol{s})$$

where $\boldsymbol{s}$ contains information about both the DLO and the end-effector. This control policy can then be mapped to the joint velocities of an arbitrary manipulator, such as the anthropomorphic arm of Figure 2.3. The reference joint velocities can be calculated by minimizing the norm $||\boldsymbol{v}_e^{\text{ref}} - \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}||$, using simple inverse differential kinematics or Hierarchical Quadratic Programming (HQP) in case equality and inequality constraints are desired and can be ordered into a strict hierarchy [48].

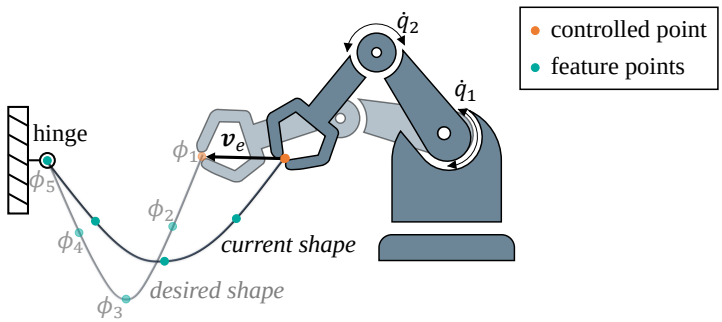


Figure 2.3: Illustration of problem formulation. For an explicit shape control problem, the goal is to deform the DLO into a *desired shape*, by controlling the velocity of the end-effector $\boldsymbol{v}_e$. Inverse kinematics can then be implemented to map $\boldsymbol{v}_e$ to joint velocities $\dot{q}_1$ and $\dot{q}_2$, as well as to control feature points ϕ_i on the DLO, with $i \in \{1, \ldots, 5\}$.

Given the ubiquity of inverse kinematic control in robotics, several DLO manipulation strategies have followed a similar approach, by modeling the object with a deformation Jacobian $\boldsymbol{J}_{\text{DLO}}(\boldsymbol{\phi})$, where $\boldsymbol{\phi}$ is some representation of the DLO's shape [49]–[52]. A simple representation is given by coordinates of feature points, as shown in Figure 2.3. However, this Jacobian matrix depends on the deformation model of the DLO requiring some calibration procedure to obtain initial modeling parameters. On the other hand, if the model is unknown, this matrix can be directly estimated through system identification. Furthermore, this type of implementation has to be adaptive since the nonlinear behavior of the DLO quickly makes the initial estimate inaccurate [49]. Therefore, these methods have only been applied to DLOs which are assumed to be elastic and are likely to fail in objects with elastoplastic properties.

One of the key obstacles for such *model-based* approaches is the modeling complexity of deformable objects, as highlighted in Section 2.1. Since models are bound to misrepresent the behavior of the object, the controller needs to be robust to model inaccuracies. Alternatively, a learned model can be used, however this requires sufficiently varied training data, otherwise overfitting may lead to poor generalization in unseen states. Yet another approach is to learn a policy directly, without learning an explicit deformation model. However, such *model-free* methods tend to be less data efficient, since finding the adequate mapping between states and actions implicitly requires that the underlying deformation model must also be learned [2]. The problem is only exacerbated by the larger state and action spaces associated with DOM.

The work presented in this thesis addresses multiple DLO shape control problems in a model-free fashion. RL is applied to learn a policy in operational space, using different state representations and reward definitions which can be obtained from vision data. The goal of shape control may be either to deform the DLO into a specific shape, or to manipulate it so that a more general condition is satisfied, e.g. tying a knot, wrapping DLO around a cylinder, etc. In Part II, shape control problems are classified based on this goal difference. The first type of problems are considered **explicit shape control**, and the latter are **implicit shape control**. This distinction is particularly useful in the context of RL, since adequately defining the goal of the task is key to its success. In particular, this affects the reward definition, as is highlighted in Paper A, which focuses on an explicit shape control problem. Paper B, includes three environments of each class.

While RL is a very general framework for trial-and-error learning which includes a great variety of algorithms, in this thesis a single algorithm was used: Deep Deterministic Policy Gradient (DDPG). Therefore, in order to provide the necessary context for the papers presented in Part II it might be sufficient to describe DDPG and move on. Let us try to do this now:

DDPG is a model-free, off-policy, actor-critic method. A policy is updated using the deterministic policy gradient theorem, with an estimate of the action-value function. In turn the action-value estimate is updated based on the mean squared error between the current value estimate and the TD(0) target.

For readers familiar with reinforcement learning the provided description may have been clear, but for others the terminology of RL can be a steep hill to climb. The following chapter provides the necessary context to understand DDPG and the motivation behind choosing it over other algorithms. Before that, the RL landscape is introduced starting with a broad view of the subject. This is followed by the challenges of applying RL specifically to robotics. Finally, a few related works where RL has been used for deformable object manipulation are enumerated.

CHAPTER 3

Reinforcement Learning

RL was briefly introduced in Section 1.2, within the context of robot learning. However, reinforcement learning can be applied to a wide range of sequential decision making problems. Indeed one of the most mediatic successes of RL has been the achievement of superhuman performance in the game of Go. First in 2016 when DeepMind's AlphaGo [53] algorithm defeated grandmaster Lee Sedol, and replicated one year later when it came out victorious over Ke Jie, the highest ranked player at the time. RL has been exceptionally efficient at learning to play a variety of board games like chess [54] and backgammon [55] as well as video games like Atari [56] and StarCraft II [57].

But why did these methods suddenly become so effective? RL had been around for decades, and using ANNs as the main function approximation technique was common. The answer lies partially within advances in DL, which enabled more powerful Deep Reinforcement Learning (DRL) algorithms. A key turning point can be traced to 2012, when the Convolutional Neural Network (CNN) architecture later named AlexNet [58] far outperformed past results of the annual ImageNet Large-Scale Visual Recognition Challenge. Besides algorithmic improvements, the increase of available compute power made possible by GPU acceleration was also a catalyst.

Nevertheless, there are other important factors that contributed to the success of DRL in games which unfortunately do not hold true for robotic manipulation. Firstly, these environments are mostly fully observable and deterministic[1], with often discrete and fairly low-dimensional state and action spaces. Secondly, they enable the possibility of self-play, effectively learning to improve on past policies by competing against them. Thirdly, winning a game consists of a simple goal definition which provides an obvious reward signal. Finally, it is important to note that AlphaGo combined RL with planning by Monte Carlo Tree Search (MCTS) methods [53].

In contrast, the state and action spaces in robotic manipulation problems are continuous and *high-dimensional*. Furthermore, due to limitations in perception they are mostly partially observable. Indeed the majority of the obstacles to applying RL in robotics is due to interaction with the *real world*, where there are numerous sources of stochasticity and noise. While game agents can remain in a virtual world and therefore play through thousands of matches in seconds, robotic agents have to control physical systems which are limited to real-time execution. This problem is only exacerbated by conservative velocity constraints meant to prevent damage on such an expensive piece of equipment. There have been attempts to use multiple robots in parallel to collect more data [59], however the costs are prohibitive for most academic research institutions. Even the simplest task of resetting the environment is a challenge, given that it either requires some automated approach capable of handling unpredictable states or a human tediously standing by to do it [60]. Two main approaches have been used to overcome this issue: **i.** use LfD to initialize the RL algorithm with a good starting control policy and try to improve from there, or **ii.** learn in simulation first and then transfer the learned policy to the real robot. This thesis focuses on the latter, by developing DOM simulation environments such as the related works presented in Section 3.6.

In addition, for many robotic tasks there is no clear reward function which fully describes the *goal*. Though sparse rewards similar to the ones used in games can be used for simpler robotic problems such as peg-in-hole, e.g. assign positive reward once the task is completed, more complex tasks tend to require **reward shaping** to lead the agent to the goal [60]. This problem is addressed in Paper A, for an explicit shape control task.

[1] Backgammon is not fully deterministic since it includes dice rolls and Starcraft II is only partially observable with some randomness.

To summarize, the key challenges of reinforcement learning in robotic applications have been succinctly described by Kober et al. [60] as four curses:

- Curse of dimensionality
- Curse of real-world samples
- Curse of goal specification
- Curse of under-modeling and model uncertainty

The last curse specifically referred to *modeling* accuracy in simulators. Given that these consist of simplifications of the real world, policies learned in such a setting may not work on the real robot. Consequently, addressing simulation-to-reality transfer is an open research topic, also termed Sim2Real, that will be discussed in Section 4.2.1. Despite these curses, the authors compiled a comprehensive list of robotic tasks to which RL has been successfully applied. Since the survey's publication in 2013, there has been a growing interest in this field, fueled by the achievements of DRL.

RL has also been tested on a few DOM applications such as the ball-paddling example from [60] where an elastic string connects the two objects. More recently, low compression strength materials such as cloth and rope have been successfully manipulated without the use of demonstrations [61]. This is in contrast with previous work on cloth folding and hanging which combined RL and LfD [43]. Reinforcement learning has also been used to solve modified classical robotics tasks, such as a peg-in-hole task where the insertion is made of foam [62], and its converse where a soft cable is inserted into a rigid hole [63]. There have even been examples from robotic surgery applications, such as pattern cutting in gauze with DRL policies for tensioning [64].

In this chapter, the goal will be to introduce the theory behind reinforcement learning. At its core, RL is a computational approach for an **agent** to learn how to achieve a goal by interacting with the **environment**. This agent-environment interaction, illustrated in Figure 3.1, can be modeled as a Markov Decision Process (MDP). Furthermore, RL is intimately related to optimal control theory and Dynamic Programming (DP), since both aim to find optimal control policies that optimize an objective function or cumulative reward. However, they assume perfect knowledge of the environment in the form of a transition model and potentially also a disturbance model.

3.1 MDP Formulation

Markov decision processes provide a useful formalism to describe sequential decision making problems. For this reason, it is common practice to frame RL problems as MDPs. Major textbooks on RL mainly focus on MDPs with finite state and action spaces [65]–[67], however robotic control is better characterized by continuous spaces. Since both cases will be considered throughout this chapter, Definition 1 provides a general description of an MDP.

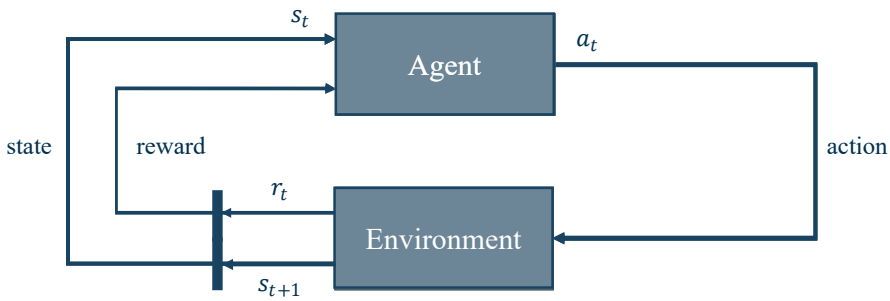


Figure 3.1: Illustration of agent-environment interaction. The agent is in state s_t and takes action a_t, leading to a reward r_t and new state s_{t+1}.

Definition 1: Markov Decision Process *An MDP is defined as a tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$*, where: The set* $\mathcal{S}$ *represents the state space and the set* $\mathcal{A}$ *the action space, which can be discrete or continuous. The action space may in some cases be a function of the state,* $\mathcal{A}(s)$ *for each* $s \in \mathcal{S}$*. For continuous spaces, it is assumed that* $\mathcal{S} \subseteq \mathbb{R}^{D_s}$ *and* $\mathcal{A} \subseteq \mathbb{R}^{D_a}$*, where* $D_s, D_a \in \mathbb{N}$ *are the dimensionalities. The aforementioned spaces characterize the MDP as finite or infinite, e.g. if both spaces are finite the MDP is finite. When the state space is continuous the dynamics of the environment are represented by the probability density function* $p(s_{t+1}|s_t, a_t)$*, which describes the probability of transitioning to state* s_{t+1}*, when in state* s_t *the agent takes action* a_t*. For a discrete state space, this is given by a probability mass function. The objective of the task is encoded by a scalar reward function, which may depend on the current state* $r(s_t)$*, both state and action* $r(s_t, a_t)$*, and even the successive state* $r(s_t, a_t, s_{t+1})$*. Finally,* $\gamma \in [0, 1]$ *is the discount factor which defines how much weight is placed on future rewards in the objective, where* $\gamma = 0$ *implies none, and* $\gamma = 1$ *implies equal weighting for all future rewards.*

For simplicity, discrete-time MDPs are considered, though much of the theory which will be introduced can be extended to continuous-time [68]. Therefore, the agent-environment interaction leads to a discrete *trajectory*:

$$h_{0:T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \ \dots \ , s_t, a_t, r_t, s_{t+1}, \ \dots \ , s_T\}$$

where T is the time of termination. While the convention throughout this thesis is that a reward at time t is the result of taking action a_t when in state s_t, as shown in Figure 3.1, this differs from the convention used in [66].

MDPs may also be classified with respect to their time horizon. An MDP is said to be finite-horizon if $T < \infty$, otherwise the MDP is said to be infinite-horizon. Sutton and Barto [66] further classify tasks as *episodic* if they can be broken down into a sequence of episodes, such as finite-horizon cases, or *continuing* for infinite-horizon MDPs. It is possible to unify these classes if termination is assumed to be equivalent to reaching an absorbing state s_T, for which any action leads to a transition to itself, without any reward, i.e. $s_t = s_T, r_t = 0, \ \forall t > T$ [66]. The objective in MDP problems can then be generally defined in terms of the *return*, R_t:

$$R_t \doteq \sum_{k=t}^{T} \gamma^{k-t} r_k = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \tag{3.1}$$

where r_k is the immediate reward at time k and $T \in [0, \infty]$, with the caveat that when $T = \infty$ the discount factor must satisfy $\gamma < 1$, otherwise the sum becomes undefined. The return expresses the sum of future rewards at time step t, and can be easily related to the successive return, for all $t < T$, as:

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &= r_t + \gamma \left(r_{t+1} + \gamma r_{t+2} + \dots \right) \\ &= r_t + \gamma R_{t+1} \end{aligned} \tag{3.2}$$

leading to a recursive property which is exploited in DP algorithms.

Finally, for a problem to be modeled as an MDP, the Markov property must hold. Namely the state needs to be a sufficient statistic for predicting the future, independently from past observations. Furthermore, MDPs are assumed to be stationary, i.e. the transition probabilities and reward function are not time-dependent.

3.2 Dynamic Programming

Dynamic programming refers to a group of algorithms used to compute optimal policies, based on a known MDP. For simplicity, the algorithms presented in this section assume finite MDPs with discrete state and action spaces. A *policy* defines how the agent interacts with the environment. Therefore, a stochastic policy $\pi(a|s) = \mathbb{P}(a_t = a|s_t = s)$ describes the probability of taking action a in state s, at time step t. Policies can also be deterministic, i.e. $\pi(s) = a$, in which case they simply map states to actions, $\pi : \mathcal{S} \to \mathcal{A}$.

DP makes use of value functions to guide the search for an optimal policy, i.e. the policy which maximizes the expected return. In order to evaluate a state, the *state-value function* expresses the expected return conditioned on starting in a state s, and following policy π thereafter:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[R_t|s_t = s\right] \tag{3.3}$$

Similarly, to evaluate a state-action pair, the *action-value function* expresses the expected return further conditioned on taking a specific action a:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[R_t|s_t = s, a_t = a\right] \tag{3.4}$$

A result of the recursive property from equation (3.2), is that it is possible to derive the *Bellman equation* for v_π as:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[r_t + \gamma v_\pi(s_{t+1})|s_t = s] && (3.5)\\ &= \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \left[r_t + \gamma v_\pi(s')\right] && \text{stochastic policy}\\ &= \sum_{s'} p\left(s'|s, \pi(s)\right) \left[r_t + \gamma v_\pi(s')\right] && \text{deterministic policy} \end{aligned}$$

where $s' \in \mathcal{S}$ indicates any given successive state. These expressions can be used as update rules to evaluate a policy. Algorithm 1 takes as input the policy to be evaluated π and a small threshold β which defines the estimation accuracy. Note that $\boldsymbol{v}$ is a vector with the value of all states $s \in \mathcal{S}$, and $\boldsymbol{v}(s)$ is a specific element of that vector. At the end of a full sweep through the state space, the largest observed change Δ is used to evaluate the stopping condition. Since in-place updates are applied for each evaluated state, the values $\boldsymbol{v}(s')$ may be from the previous sweep or the current one.

Algorithm 1 Iterative Policy Evaluation (deterministic policy)

Set desired accuracy parameter β
Randomly initialize $\boldsymbol{v}$, except $\boldsymbol{v}(s_T) = 0$
function IPE(π, $\boldsymbol{v}$, β)
  $\Delta \leftarrow \beta$
  while $\Delta \geq \beta$ **do**
    $\Delta \leftarrow 0$
    for all $s \in \mathcal{S}$ **do**
      $v \leftarrow \boldsymbol{v}(s)$
      $\boldsymbol{v}(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \left[r + \gamma \boldsymbol{v}(s')\right]$
      $\Delta \leftarrow \max(\Delta, |v - \boldsymbol{v}(s)|)$
  return $\boldsymbol{v}$

The action-value function can be expressed with respect to the state-value function. Namely, if an agent is in state s and takes action a leading to successive state s', and from there onward follows the policy π:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[r_t + \gamma v_\pi(s_{t+1}) | s_t = s, a_t = a] \\ &= \sum_{s'} p\left(s'|s,a\right)\left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{3.6}$$

Therefore, if $q_\pi(s,a) \geq v_\pi(s)$, then the action taken in state s when following policy π was worse than a, which is a special case of Theorem 1.

Theorem 1: Policy Improvement *Let π and π' be deterministic policies such that*

$$q_\pi\left(s, \pi'(s)\right) \geq v_\pi(s), \ \forall s \in \mathcal{S}$$

Then the policy π' is at least as good as π, i.e. $v_{\pi'}(s) \geq v_\pi(s), \ \forall s \in \mathcal{S}$.

This idea can be extended from a single state-action pair to all possible states and actions by always choosing the best action according to $q_\pi(s,a)$, which leads to the new greedy policy π':

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\, q_\pi(s,a) \tag{3.7}$$

By iterating this process, it is possible to monotonically improve until the new greedy policy is as good but not better than the previous one, i.e. $v_\pi(s) = v_{\pi'}(s), \ \forall s \in \mathcal{S}$. Once this occurs, then v_π must be v_* implying that π was already an optimal policy π_*.

Theorem 1 results in the first DP algorithm for control, namely *policy iteration*. It works by interleaving policy evaluation and policy improvement steps, and is guaranteed to converge to the optimal solution, as in Algorithm 2. The general idea of alternating between these two interacting processes of evaluation and improvement is referred to as Generalized Policy Iteration (GPI), illustrated in Figure 3.2. GPI can be used to describe the RL methods which will be presented throughout the rest of the chapter.

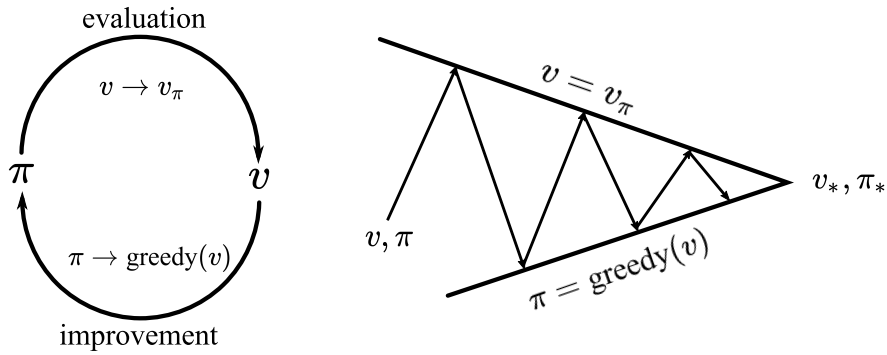


Figure 3.2: Illustration of generalized policy iteration: policy evaluation and improvement processes interact until the system converges to optimal policy, π_*. GPI can also be expressed in terms of action-values, and the evaluation step does not necessarily have to converge to v_π [66].

Algorithm 2 Policy Iteration (deterministic policy)

Set desired accuracy parameter β
`policy_stable` $\leftarrow$ `false`
Randomly initialize π and $\boldsymbol{v}$, except $\boldsymbol{v}(s_T) = 0$
while `policy_stable` = `false` **do**
  $\boldsymbol{v} \leftarrow \text{IPE}(\pi, \boldsymbol{v}, \beta)$ ▷ Policy Evaluation
  `policy_stable` $\leftarrow$ `true` ▷ Policy Improvement
  for all $s \in \mathcal{S}$ **do**
    `old_action` $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \text{argmax}_a \sum_{s'} p(s'|s,a)\,[r + \gamma \boldsymbol{v}(s')]$
    if `old_action` $\neq \pi(s)$ **then**
      `policy_stable` $\leftarrow$ `false`

Since each iteration in Algorithm 2 includes an evaluation loop, the convergence speed to the optimal policy depends on the number of steps required to reach the desired estimation accuracy β. Furthermore, the greedy policy may actually converge before the state value function converges, which leads to several strategies to truncate the IPE loop. The most extreme case of GPI, is when the evaluation is stopped after just one sweep of the state space, leading to the other major DP algorithm that is based on the *Bellman optimality equation* for v_*:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[r_t + \gamma v_*(s_{t+1}) | s_t = s, a_t = a] \\ &= \max_a \sum_{s'} p(s'|s,a) \left[r + \gamma v_*(s') \right] \end{aligned} \tag{3.8}$$

The *value iteration* algorithm is formulated by using equation (3.8) as an update rule. This leads to a loop similar to the IPE algorithm but where the value estimate is based on the best action, as can be seen in Algorithm 3.

Algorithm 3 Value Iteration (deterministic policy)

```
Set desired accuracy parameter β
Randomly initialize v, except v(s_T) = 0
function VI(v, β)
    Δ ← β
    while Δ ≥ β do
        Δ ← 0
        for all s ∈ S do
            v ← v(s)
            v(s) ← max_a Σ_{s'} p(s'|s,a) [r + γ v(s')]
            Δ ← max(Δ, |v − v(s)|)
    return π = argmax_a Σ_{s'} p(s'|s,a) [r + γ v(s')]
```

Policy and value iteration provide the theoretical basis for RL. However, since DP is *model-based* it can only be applied when the MDP is known and relatively small. Alternatively, *model-free* RL can be explored through methods where the value function is approximated, as in Section 3.3, or a policy is directly optimized, as in Section 3.4. Nevertheless, both alternatives are less sample efficient than model-based algorithms. On the other hand, learning a model is challenging [69], as will be discussed in Section 4.2.2.

3.3 Value Function Approximation

Monte Carlo (MC) methods provide a simple strategy to approximate value functions by sampling complete trajectories from the agent-environment interaction, which can be real or simulated. MC algorithms work by estimating the average return based on the rewards observed throughout an episode until termination. Therefore, they can only be applied to episodic tasks, for which the return can be explicitly computed by moving backwards in time, i.e. $R_t = r_t + \gamma(r_{t+1} + \gamma(\dots(r_{T-1} + \gamma R_T))$, with $t < T - 2$.

Temporal Difference (TD) learning, is an alternative approach which does not require sampling full episodes. While MC methods use the return $R_t \equiv R_t^{(T)}$ to update the value estimate, TD methods *bootstrap* using current value estimates as the target, e.g. $R_t^{(1)} = r_t + \gamma v(s_{t+1}) = r_t + \gamma q(s_{t+1}, a_{t+1})$ or $R_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 v(s_{t+2})$, where the superscript (n) denotes the bootstrap depth. A one-step bootstrap update is referred to as TD(0), and is similar to DP updates. The Sarsa algorithm uses such an update to approximate the action-value function. This method is presented below (in red) together with the Q-learning algorithm (in blue) proposed by Watkins [70].

Algorithm 4 TD Control (Sarsa & Q-learning)

Randomly initialize $\boldsymbol{Q}(s, a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$, except $\boldsymbol{Q}(s_T, \cdot) = \mathbf{0}$
for `episode` $= 1 : M$ **do**
  Obtain initial state s_0
  Choose a_0 from current state s_0 using policy derived from $\boldsymbol{Q}$ S
  for $t = 0 : T - 1$ **do**
    Choose a_t from current state s_t using policy derived from $\boldsymbol{Q}$ Q
    Execute action a_t, observe reward r_t and new state s_{t+1}
    $\boldsymbol{Q}(s_t, a_t) \leftarrow \boldsymbol{Q}(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} \boldsymbol{Q}(s_{t+1}, a') - \boldsymbol{Q}(s_t, a_t)\right]$ $\boldsymbol{Q}$ Q
    Choose a_{t+1} from next state s_{t+1} using policy derived from $\boldsymbol{Q}$ S
    $\boldsymbol{Q}(s_t, a_t) \leftarrow \boldsymbol{Q}(s_t, a_t) + \alpha \left[r_t + \gamma \boldsymbol{Q}(s_{t+1}, a_{t+1}) - \boldsymbol{Q}(s_t, a_t)\right]$ S
    $a_t \leftarrow a_{t+1}$ S
    $s_t \leftarrow s_{t+1}$

Algorithm 4 facilitates the comparison of TD control methods based on action-values. Sarsa uses soft updates $\boldsymbol{Q}(s_t, a_t) \leftarrow \boldsymbol{Q}(s_t, a_t) + \alpha\delta_t$, where $\delta_t = R_t^{(1)} - \boldsymbol{Q}(s_t, a_t)$ is the TD error and α is the step-size. While Sarsa has $R_t^{(1)}$ as the bootstrapping target, Q-learning has the return of the optimal policy $r_t + \gamma \max_{a'} \boldsymbol{Q}(s_{t+1}, a')$. Q-learning is an *off-policy* algorithm because it uses action-value estimates from π_* while following another policy π, whereas Sarsa is an *on-policy* method which uses estimates from the same policy π.

Contrary to DP updates that consider the whole distribution of possible states, MC and TD methods only update values of visited states. Therefore, they must guarantee sufficient exploration of the state space. Consequently, the greedy policies implemented in DP methods which exploited value estimates need to be modified to include exploratory actions. This introduces a trade-off referred to as the *exploration-exploitation dilemma*. A simple approach is given by the ε-greedy policy, with $\varepsilon \in (0, 1)$:

$$\pi(s) = \begin{cases} a^* = \operatorname{argmax}_{a \in \mathcal{A}} \boldsymbol{Q}(s, a) & \text{with probability } 1 - \varepsilon \\ a \in \mathcal{A} & \text{with probability } \varepsilon \end{cases} \tag{3.9}$$

which means the total probability of choosing an optimal action is $\pi(a^*|s) = \frac{\varepsilon}{|\mathcal{A}|} + (1 - \varepsilon)$ and any suboptimal action is $\pi(a|s) = \frac{\varepsilon}{|\mathcal{A}|}$, with $a \neq a^*$.

This type of approach effectively embeds exploration into the algorithm and on-policy methods, e.g. Sarsa, rely on it for exploration. Conversely, off-policy algorithms have a *behavior policy* μ, which selects actions and a separate *target policy* π, which is actually updated. This allows the use of deterministic target policies, e.g. greedy, since the behavior policy will continue to sample random actions. For example, in the Q-learning algorithm, actions may be selected based on a separate behavior policy like ε-greedy.

It is possible to unify TD learning with MC methods given that as $n \to T$, the TD updates become equivalent to Monte Carlo updates. However, the optimal value of n is not known *a priori* because it is problem and algorithm dependent. The TD(λ) approach addresses this issue by averaging over several n-step returns to compute an update, where $\lambda \in [0, 1]$ refers to the eligibility trace-decay parameter [66, Chapter 12]. The target of this type of update is the λ-return, defined as the geometrically weighted average of all n:

$$R_t^\lambda \doteq (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} = (1 - \lambda) \sum_{n=1}^{T-t} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t} R_t \tag{3.10}$$

For the methods presented so far, the state- and action-value functions have been estimated as a vector $\boldsymbol{v} \in \mathbb{R}^{|\mathcal{S}|}$ and a matrix $\boldsymbol{Q} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$, respectively. Nevertheless, this is only possible if the state and action spaces are bounded and discrete. Otherwise, they must either be quantized or *function approximation* needs to be employed. There are many alternative function approximation techniques however, this work focuses on DNNs that will be briefly introduced in Section 3.5. To apply these methods, the state-value function denoted $v(s|\mathbf{v})$, is parameterized with $\mathbf{v} \in \mathbb{R}^{D_\mathbf{v}}$, where $D_\mathbf{v} \in \mathbb{N}$ is the dimensionality of the parameter vector. For action-values, the parameterized function may be *action-in*, denoted $q(s, a|\mathbf{w})$ which outputs a scalar or *action-out*, denoted $q(s, \cdot|\mathbf{w})$ which instead outputs a vector. Choosing between these two formulations depends on whether the RL algorithm requires the q value for a particular action a or for all possible actions in a given state s. Note that the *action-out* formulation implies a finite action space. Action-value function parameters are denoted $\mathbf{w} \in \mathbb{R}^{D_\mathbf{w}}$ with dimensionality $D_\mathbf{w} \in \mathbb{N}$.

3.4 Policy Approximation

The previous two sections covered *value-based* methods, in which policies are only implicitly defined through the value function, i.e. the *critic*, requiring maximization over actions to either select an action or update the value estimates. Consequently, the algorithms presented so far are not directly applicable to problems with continuous action spaces. An alternative approach is found in *policy-based* methods, which search for an explicit policy i.e. an *actor*, and do not suffer from the same limitation. This section will introduce the key idea behind policy search, with a focus on gradient methods. These will also be combined with value-based strategies in what is referred to as *actor-critic* methods, which improve efficiency by reducing variance.

Policy-based methods are a natural choice for robotic applications, not only due to their applicability to continuous MDPs but also because learning in policy space often requires fewer parameters than in value space. Furthermore, they offer a direct approach to incorporate prior knowledge through the choice and initialization of the policy representation, as well as the inclusion of constraints. Moreover, since in value-based methods a small change in the value function may result in large discontinuous changes to the policy, these can lead to dangerous actions [71].

To approximate an optimal policy, one needs to define a parameterized policy $\pi_{\boldsymbol{\theta}}$, with parameters $\boldsymbol{\theta} \in \mathbb{R}^{D_{\boldsymbol{\theta}}}$ of dimensionality $D_{\boldsymbol{\theta}}$. This section will focus on **Policy Gradient** (PG) methods, which further require the policy to be differentiable with respect to its parameters. However, there are also gradient-free optimization methods such as evolutionary algorithms, which may be used otherwise. Depending on the algorithm, this policy can be stochastic $\pi(a|s, \boldsymbol{\theta})$ or deterministic $\pi(s|\boldsymbol{\theta})$. Once a policy parameterization has been defined, it is possible to treat this as an optimization problem, by defining a performance measure $J(\pi_{\boldsymbol{\theta}})$. This measure is defined differently for episodic tasks where the *starting-state* formulation is used and for continuing tasks which require the *average-reward* formulation.

Starting-state formulation The goal is to maximize the total reward over an episode, from a non-random starting-state s_0 to a terminal state s_T:

$$J_{s_0}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R_0] = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{k=0}^{T-1} \gamma^k r_k \middle| s_0\right] \tag{3.11}$$

which is just the state-value function $v_{\pi_{\boldsymbol{\theta}}}(s_0)$.

Average-reward formulation The goal is to maximize the expected average reward per time step:

$$J_{\bar{r}}(\pi_{\boldsymbol{\theta}}) = \frac{1}{T}\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{k=0}^{T-1} r_k\right] \tag{3.12}$$

which for infinite horizon MDPs can be evaluated as a limit:

$$J_{\bar{r}}(\pi_{\boldsymbol{\theta}}) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{k=0}^{T-1} r_k\right] \tag{3.13}$$

For simplicity, this section considers the starting-state formulation for continuous state and action spaces. In either case, the goal is to improve the parameterized policy $\pi_{\boldsymbol{\theta}}$ by updating its parameters $\boldsymbol{\theta}$ through (stochastic) *gradient ascent*:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_a \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$$

where α_a is the step-size parameter and the subscript a stands for actor.

The problem with such PG objectives is that the policy $\pi_{\boldsymbol{\theta}}$ affects not only the actions taken in each state, but it also indirectly affects the state distribution $\rho_{\pi_{\boldsymbol{\theta}}}(s)$, through interaction with the unknown environment dynamics $p(s'|s,a)$. This makes gradient computation problematic since the effect of policy parameters on the state distribution is not known.

A simple numerical approach to compute the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is by **finite-difference methods**. These approximate derivatives with finite differences by perturbing each parameter θ_k with a small amount ϵ, in each step:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_k} \approx \frac{J(\boldsymbol{\theta} + \epsilon \mathbb{1}_k) - J(\boldsymbol{\theta})}{\epsilon} \tag{3.14}$$

where $\mathbb{1}_k$ is a unit vector with 1 in the k-th component and 0 elsewhere [71].

A more powerful strategy is used by **likelihood ratio methods**, which allow for an analytical computation of the gradient using Theorem 2.

Theorem 2: Policy Gradient *For the objective function $J(\pi_{\boldsymbol{\theta}})$ and any differentiable **stochastic** policy, the gradient is given by*

$$\begin{aligned}\nabla_\theta J(\pi_{\boldsymbol{\theta}}) &\propto \int_{\mathcal{S}} \rho_{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}) q_{\pi_{\boldsymbol{\theta}}}(s,a) \mathrm{d}a \mathrm{d}s \\ &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi(a|s,\boldsymbol{\theta}) q_{\pi_{\boldsymbol{\theta}}}(s,a)]\end{aligned}$$

where $\rho_{\pi_{\boldsymbol{\theta}}}(s)$ is the on-policy distribution under $\pi_{\boldsymbol{\theta}}$. The proportionality sign $\propto$ becomes an equality for the continuing case, while in the episodic case the proportionality factor is the average episode length.

*Recently, the theorem was extended to **deterministic** policies by Silver et. al. [72], assuming the underlying MDP satisfies some regularity conditions:*

$$\begin{aligned}\nabla_\theta J(\pi_{\boldsymbol{\theta}}) &\propto \int_{\mathcal{S}} \rho_{\pi_{\boldsymbol{\theta}}}(s) \nabla_{\boldsymbol{\theta}} \pi(s|\boldsymbol{\theta}) \nabla_a q_{\pi_{\boldsymbol{\theta}}}(s,a)|_{a=\pi_{\boldsymbol{\theta}}(s)} \mathrm{d}s \\ &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \pi(s|\boldsymbol{\theta}) \nabla_a q_{\pi_{\boldsymbol{\theta}}}(s,a)|_{a=\pi_{\boldsymbol{\theta}}(s)}]\end{aligned}$$

which was shown to be a special (limiting) case of the stochastic policy gradient.

The key feature of the PG theorem is that $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ does not include the gradient of the on-policy state distribution $\rho_{\pi_{\boldsymbol{\theta}}}(s)$, despite the fact that this depends on the policy parameters $\boldsymbol{\theta}$.

Therefore, PG methods based on the likelihood ratio only need to find an estimate of the action-value $q_{\pi_{\boldsymbol{\theta}}}(s,a)$. One of the first PG algorithms used the sample return R_t for that purpose, making it an MC implementation. This algorithm, appropriately named REINFORCE, was proposed by Williams [73] and is presented in Algorithm 5.

Algorithm 5 REINFORCE (with baseline)

Randomly initialize policy $\pi(a|s,\boldsymbol{\theta})$, with parameters $\boldsymbol{\theta}$
Randomly initialize differentiable value function $v(s|\mathbf{v})$, with parameters $\mathbf{v}$
for `episode` $= 1 : M$ **do**
  Generate an episode h following $\pi(a|s,\boldsymbol{\theta})$
  for $t = 0 : T-1$ **do**
    $R_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    $A \leftarrow R_t - v(s_t|\mathbf{v})$
    $\mathbf{v} \leftarrow \mathbf{v} + \alpha_b \nabla_{\mathbf{v}} v(s_t|\mathbf{v}) A$
    $R_t \leftarrow A$ ▷ i.e. use advantage function in place of return
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_a \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t,\boldsymbol{\theta}) R_t$ ▷ PG update

Since REINFORCE is an MC method, it suffers from high variance and is slow to learn. To help reduce variance, Williams proposed a variant of the algorithm with a *baseline function*, $b(s)$. This can be any function which does not depend on the actions a. The baseline is subtracted from the state-action value in the policy gradient theorem, so that the expectation remains intact:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi(a|s,\boldsymbol{\theta}) \left(q_{\pi_{\boldsymbol{\theta}}}(s,a) - b(s) \right) \right] \tag{3.15}$$

This difference is called an *advantage function*, $A_{\pi_{\boldsymbol{\theta}}}(s,a)$, typically denoted by uppercase letter A to differentiate it from actions. A good baseline is the state-value function $b(s) = v_{\pi_{\boldsymbol{\theta}}}(s)$ which results in:

$$A_{\pi_{\boldsymbol{\theta}}}(s,a) = q_{\pi_{\boldsymbol{\theta}}}(s,a) - v_{\pi_{\boldsymbol{\theta}}}(s) \tag{3.16}$$

Algorithm 5 shows REINFORCE with baseline in blue. Note that the step-size of the value function has subscript b for baseline and not c for critic. According to [66, Chapter 13.5], the value function is only considered a critic when it is used for bootstrapping from estimated values of successive states.

Instead of the MC return R_t, it is possible to choose a different estimate of $q_{\pi_{\boldsymbol{\theta}}}(s,a)$, using a TD learning critic to reduce variance, i.e. $q_{\pi_{\boldsymbol{\theta}}}(s,a|\mathbf{w})$. This leads to the final group of algorithms which will be presented in this chapter, namely **actor-critic** methods. A baseline may still be used to further improve stability, leading to advantage actor-critic methods. However, replacing the true $q(s,a)$ by an approximation does not guarantee that it will represent the true gradient, unless Theorem 3 holds.

Theorem 3: Compatible Function Approximation *If the following two conditions are satisfied:*

1. *The value function approximator $q(s,a|\mathbf{w})$ is compatible to the policy $\pi_{\boldsymbol{\theta}}$:*

$$\begin{aligned} \nabla_{\mathbf{w}} q(s,a|\mathbf{w}) &= \nabla_{\boldsymbol{\theta}} \log \pi(a|s,\boldsymbol{\theta})^\intercal \mathbf{w} && \text{stochastic policy} \\ \nabla_a q(s,a|\mathbf{w})|_{a=\pi_{\boldsymbol{\theta}}(s)} &= \nabla_{\boldsymbol{\theta}} \pi(s|\boldsymbol{\theta})^\intercal \mathbf{w} && \text{deterministic policy} \end{aligned}$$

2. *The value function parameters $\mathbf{w}$ minimize the mean-squared error:*

$$\begin{aligned} MSE_{sto} &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\left(q_{\pi_{\boldsymbol{\theta}}}(s,a) - q(s,a|\mathbf{w})\right)^2\right] \\ MSE_{det} &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\left(\nabla_a q_{\pi_{\boldsymbol{\theta}}}(s,a)|_{a=\pi_{\boldsymbol{\theta}}(s)} - \nabla_a q(s,a|\mathbf{w})|_{a=\pi_{\boldsymbol{\theta}}(s)}\right)^2\right] \end{aligned}$$

then the policy gradient is unbiased:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_\theta \log \pi(a|s,\boldsymbol{\theta}) q(s,a|\mathbf{w})] && \text{stochastic policy} \\ \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla_\theta \pi(s|\boldsymbol{\theta}) \nabla_a q(s,a|\mathbf{w})|_{a=\pi_{\boldsymbol{\theta}}(s)}] && \text{deterministic policy} \end{aligned}$$

What this theorem expresses is that function approximators have to be linear in the policy features: $\nabla_{\boldsymbol{\theta}} \pi(s|\boldsymbol{\theta})$ or $\nabla_{\boldsymbol{\theta}} \log \pi(a|s,\boldsymbol{\theta})$. Furthermore, the parameters should be the solution to the linear regression problem minimizing the *MSE*. However, in practice this second condition is relaxed to include value estimation methods such as TD learning [72].

It is now possible to introduce the precursor to DDPG, namely the Compatible Off-Policy Deterministic Actor-Critic (COPDAC-Q) algorithm proposed in [72]. Since this is an off-policy method with behavior policy $\mu(a|s) \neq \pi(s|\boldsymbol{\theta})$, the gradient becomes slightly different:

$$\begin{aligned} \nabla J_\mu(\pi_{\boldsymbol{\theta}}) &\approx \int_{\mathcal{S}} \rho_\mu(s) \nabla_{\boldsymbol{\theta}} \pi(s|\boldsymbol{\theta}) q_{\pi_{\boldsymbol{\theta}}}(s,a) \mathrm{d}s \\ &= \mathbb{E}_\mu\left[\nabla_{\boldsymbol{\theta}} \pi(s|\boldsymbol{\theta}) \nabla_a q_{\pi_{\boldsymbol{\theta}}}(s,a)|_{a=\pi_{\boldsymbol{\theta}}(s)}\right] \end{aligned} \tag{3.17}$$

where $\rho_\mu(s)$ is the state distribution under policy μ. Moreover, the Q in the name refers to the critic implementation which uses the action-value TD(0) update, as presented in Algorithm 6. Hence, condition 2 is technically not satisfied, since the critic parameters $\mathbf{w}$ are updated by TD learning. In practice, they implement the critic as a linear function approximator $q(s, a|\mathbf{w}) = \phi(s, a)^\intercal \mathbf{w}$, from state-action features defined as $\phi(s, a) = a^\intercal \nabla_{\boldsymbol{\theta}} \pi(s|\boldsymbol{\theta})$ to satisfy condition 1. The paper mentions a state-value function baseline which is also defined as a linear function approximator $v(s|\mathbf{v}) = \mathbf{v}^\intercal \phi(s)$. Nevertheless, it is not clear how the parameters $\mathbf{v}$ affect the PG update and the features $\phi(s)$ are never explicitly defined.

Algorithm 6 COPDAC-Q (Deterministic PG)

Randomly initialize policy $\pi(s|\boldsymbol{\theta})$, with parameters $\boldsymbol{\theta}$
for `episode` $= 1 : M$ **do**
  Obtain initial observation state s_0
  for $t = 0 : T - 1$ **do**
    $a_t \leftarrow \mu(a|s_t)$
    Execute action a_t, observe reward r_t and new state s_{t+1}
    $\delta_t \leftarrow r_t + \gamma q\left(s_{t+1}, \pi(s_{t+1}|\boldsymbol{\theta})\middle|\mathbf{w}\right) - q(s_t, a_t|\mathbf{w})$ ▷ TD-error
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_a \nabla_{\boldsymbol{\theta}} \pi(s_t|\boldsymbol{\theta}) \left(\nabla_{\boldsymbol{\theta}} \pi(s_t|\boldsymbol{\theta})^\intercal \mathbf{w}\right)$ ▷ PG update
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha_c \delta_t \ \phi(s_t, a_t)$
    $\mathbf{v} \leftarrow \mathbf{v} + \alpha_b \delta_t \ \phi(s_t)$

For the PG update the gradient of the action-value with respect to a is given by: $\nabla_a q(s, a|\mathbf{w}) = \nabla_{\boldsymbol{\theta}} \pi(s_t|\boldsymbol{\theta})^\intercal \mathbf{w}$. Furthermore, for the gradient ascent steps, the gradients of $q(s, a|\mathbf{w})$ and $v(s|\mathbf{v})$ are just the features $\phi(s, a)$ and $\phi(s)$, respectively. Unlike the REINFORCE algorithm, the PG update is based on the current value of the critic, and then the critic is updated towards the direction of the TD-error, δ_t.

Already in [72], the COPDAC-Q algorithm was tested with ANN function approximation, using Multilayer Perceptrons (MLPs) for both the actor and the critic. This was tested on a goal-reaching task with a simulated 6-segment octopus arm, where $D_s = 50$ and $D_a = 20$. The MLPs had relatively low capacity since the networks contained a single hidden layer where the policy had 8 neurons with a sigmoidal activation, and the critic had 40 neurons with a linear activation, i.e. $D_{\boldsymbol{\theta}} = 8, D_{\mathbf{w}} = 40$. The next section introduces DRL.

3.5 Deep Reinforcement Learning

Deep reinforcement learning refers to RL methods where DNNs are used to represent a value function, the policy or even the model of the dynamics.

Deep Neural Networks DNNs are artificial neural networks with a multilayer architecture where each layer contains a set of neurons. Much like their biological counterparts, *neurons* are the functional units of an ANN. The perceptron [74] is an artificial neuron model which learns a mapping from input vectors $\mathbf{x}$ to binary outputs $y \in \{0, 1\}$, as a composition of two functions $H(f(\mathbf{x}))$, where $f(\mathbf{x}) = \boldsymbol{w} \cdot \mathbf{x}_i + b$ with weight vector $\boldsymbol{w}$ and bias b, and H is the activation function. For the perceptron algorithm, H is a Heaviside step, but other activation functions include the sigmoid or hyperbolic tangent, and more recently the Rectified Linear Unit (ReLU). As a supervised learning method, the perceptron algorithm updates the parameters $\boldsymbol{w}, b$ in the direction which minimizes the error between the desired output and the current one $y_i - H(\boldsymbol{w} \cdot \mathbf{x}_i + b)$, using a set of sampled input-output pairs $\{(\mathbf{x}_i, y_i)\}$.

MLPs are networks made up of layers of perceptrons, typically with nonlinear activation functions. They are referred to as *fully connected*, because each neuron in one layer is connected to all neurons in the following layer, as shown in Figure 3.3. CNNs are another type of *feedforward* architecture, which applies parameterized convolutional filters by strides along the input data, thus reducing each local region into a scalar value. For data with a sequential nature, Recurrent Neural Networks (RNNs) have been the architecture of choice. These are not considered feedforward, since they have an internal state that is used as input to the following layers. More recently, Transformer networks [75] have outperformed previous architectures in numerous applications, thanks to their attention mechanism.

Though architectures may vary, the training procedure for most ANNs is relatively similar. Data is used to update the weights through *backpropagation*, in a direction minimizing some error which encodes the objective. In supervised learning, the error is typically defined for either prediction or classification tasks. Updates may be done for each data point, i.e. stochastic gradient descent, or in batches, i.e. minibatch gradient descent. Besides the development of better architectures, many algorithmic improvements have also contributed to more efficient training, such as the Adam algorithm for optimization [76] or batch normalization [77] and dropout [78] for regularization.

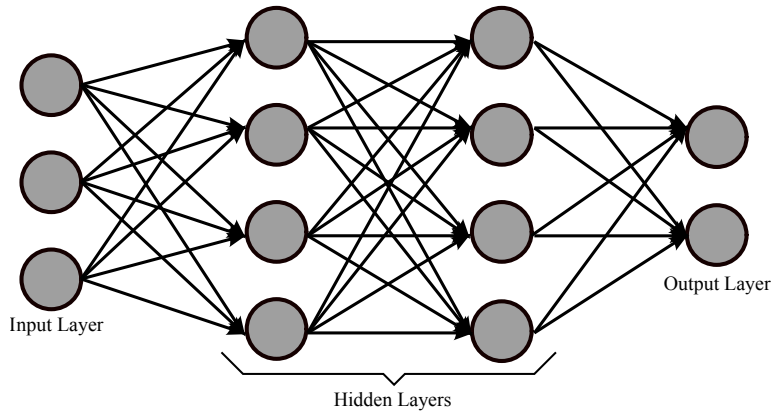


Figure 3.3: Illustration of MLP, i.e. feedforward fully connected neural network.

Despite being powerful function approximators, DNNs can lead to instabilities and even divergence. This is because convergence of learned parameters to a fixed point is not guaranteed when a nonlinear function approximator is used for value-based algorithms, presented in Section 3.3. Even with simpler linear function approximators, off-policy algorithms such as Q-learning, are no longer guaranteed to converge. This is a consequence of **the deadly triad**, which refers to the combination of *function approximation*, *bootstrapping* and *off-policy training*. Whenever these three conditions are present, even DP algorithms presented in Section 3.2 become unstable. For more details on this subject, consult [66, Chapter 11.3].

One of the earlier successes of deep reinforcement learning was the proposal of the Deep Q-Network (DQN) algorithm [56]. As the name indicates, it consists of a modification of Algorithm 4, where the q value is represented by a DNN. A key challenge in training neural networks using sampled RL data, is that optimization algorithms commonly assume the samples are independently and identically distributed (iid). Naturally, this assumption does not hold for an RL trajectory, since the state and action distributions are correlated by the interaction between the policy and the dynamics of the environment. To address this issue and make use of the increased efficiency of minibatch gradient methods, Mnih et al. [56] proposed that sampled experience tuples (s_t, a_t, r_t, s_{t+1}) be stored in a *replay buffer* $\mathcal{D}$. For each step, a minibatch $\mathcal{B}$ of uniformly sampled tuples is then used to update the Q-network. The buffer was designed as a first in, first out system, with a fixed memory. Another

challenge that needed to be addressed, is that implementing the Q-learning critic as an ANN was unstable. This is due to the bootstrapping nature of the algorithm which means that the network being updated $q(s, a|\mathbf{w})$ by gradient descent $\mathbf{w} \leftarrow \mathbf{w} + \alpha_c \nabla_{\mathbf{w}} L$ is also used to calculate the target value of the *MSE* loss:

$$L \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \left(y_i - q(s_i, a_i|\mathbf{w})\right)^2$$

where $y_i \leftarrow r_i + \max_{a'} q(s_t, a'|\mathbf{w})$ is the target. To overcome this problem, Mnih et al. [56] proposed using a *target Q network*, with separate parameters $\mathbf{w}'$, so that $y_i \leftarrow r_i + \max_{a'} q(s_t, a'|\mathbf{w}')$. The target network can then be updated at a slower rate, by setting $\mathbf{w}' \leftarrow \mathbf{w}$ only every C steps. Furthermore, the authors also found that clipping the *MSE* term from the update to be between -1 and 1, helped improve stability. However, this was fairly specific to the test domain, as discussed in [79].

Based on the algorithmic changes described above, DQN was shown to reach human-level performance in 49 of the Atari 2600 games. The critic network was modelled as an action-out CNN architecture, which was trained using pixels and games scores as inputs. However, if DQN is an off-policy, TD method which uses function approximation, how does the deadly triad affect its performance? Van Hasselt et al. attempt to answer that question in [80].

3.5.1 Deep Deterministic Policy Gradient

Unfortunately, DQN is only suitable for discrete and low-dimensional action spaces. Building on the COPDAC-Q algorithm and the observations from the DQN training strategy, Lillicrap et al. [81] proposed the deep deterministic policy gradient algorithm, for high-dimensional continuous action spaces. As shown in Algorithm 7, DDPG also makes use of the replay buffer and target network ideas. They found that in order to avoid divergence, target networks were required both for the critic and the actor. While this slows down the learning, it is highly compensated by the learning stability. Unlike the DQN strategy, the target networks are updated through soft updates defined by parameter τ, in order to have them slowly track the learned networks, as shown in item 3 of Algorithm 7. In the original implementation, the exploratory behavior policy μ was obtained by adding noise $\omega_t \sim \mathcal{W}$ to the current policy $\pi(s_t|\boldsymbol{\theta})$. For their experiments, $\mathcal{W}$ was an Ornstein-Uhlenbeck process which enables temporally correlated exploration.

Algorithm 7 Deep Deterministic PG

Initialize actor network $\pi(s|\boldsymbol{\theta})$, with weights $\boldsymbol{\theta}$
Randomly initialize critic network $q(s, a|\mathbf{w})$, with weights $\mathbf{w}$
Initialize target networks q' and π', with weights $\mathbf{w}' \leftarrow \mathbf{w}$, $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
Initialize replay buffer $\mathcal{D} \leftarrow \varnothing$
for `episode` $= 1 : M$ **do**
  Initialize a random process $\mathcal{W}$ for action exploration
  Obtain initial observation state s_0
  Set termination variable, $d_t \leftarrow 0$
  for $t = 0 : T - 1$ **do**
    $a_t \leftarrow \pi(s_t|\boldsymbol{\theta}) + \omega_t$ where $\omega_t \sim \mathcal{W}$
    Execute action a_t, observe reward r_t and new state s_{t+1}
    if $t + 1 = T$ **then** $d_t \leftarrow 1$
    Store transition in replay buffer, $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1}, d_t)\}$
    Randomly sample minibatch of transitions $\mathcal{B} \subset \mathcal{D}$
    1. Update critic network using target networks:

$$y_i \leftarrow r_i + \gamma(1 - d_i)q'\left(s_{i+1}, \pi'\left(s_{i+1}\middle|\boldsymbol{\theta}'\right)\middle|\mathbf{w}'\right), \ \forall i \in [1, |\mathcal{B}|]$$

$$L \leftarrow \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}\left(y_i - q(s_i, a_i|\mathbf{w})\right)^2$$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha_c \nabla_{\mathbf{w}} L \qquad \triangleright \text{ gradient descent step}$$

    2. Update actor network using critic network:

$$J \leftarrow \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|} q\left(s_i, \pi(s_i|\boldsymbol{\theta})\middle|\mathbf{w}\right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_a \nabla_{\boldsymbol{\theta}} J \qquad \triangleright \text{ gradient ascent step}$$

    3. Update target networks, with $\tau \ll 1$:

$$\boldsymbol{\theta}' \leftarrow \tau\boldsymbol{\theta} + (1 - \tau)\boldsymbol{\theta}'$$
$$\mathbf{w}' \leftarrow \tau\mathbf{w} + (1 - \tau)\mathbf{w}'$$

Not included in the pseudocode, is the addition of *batch normalization* [77] to update the networks. The motive to include this deep learning strategy, which normalizes a minibatch so that each dimension across samples has zero mean and unit variance, is related to the application domain. While the DQN was tested on the Atari games, the DDPG was tested on a suite of continuous control environments. The former all have the same input type and therefore were easily normalized as part of a preprocessing step, whereas in the latter each task has a specific set of inputs and outputs, with different physical units, e.g. positions (m) vs velocities (m/s).

Several policy gradient methods have been proposed and shown comparatively better results in some of tested environments. A few of the most successful algorithms are listed in Table 3.1 for reference. Other algorithmic improvements have also been proposed such as Prioritized (PER) [82] and Hindsight Experience Replay (HER) [83], Generalized Advantage Estimation (GAE) [84], TD-regularization [85] and Phasic Policy Gradient (PPG) [86]. In order to scale up RL, there have been variants of actor-critic algorithms with a focus on parallel or distributed computing, such as the Asynchronous Advantage Actor Critic (A3C) algorithm [87] and Importance Weighted Actor-Learner Architectures (IMPALA) [88].

Table 3.1: Reference table for state-of-the-art policy gradient methods. DDPG has been considered one of the more efficient off-policy DRL methods [89].

[X]	Algorithm	X-Policy	X Actor	Critic
[81]	Deep Deterministic Policy Gradient (DDPG)	Off	Deterministic	Yes
[90]	Twin-Delayed DDPG (TD3)	Off	Deterministic	Yes
[87]	Advantage Actor Critic (A2C)	On	Stochastic	Yes
[91]	Actor Critic with Experience Replay (ACER)	Off	Stochastic	Yes
[92]	Trust Region Policy Optimization (TRPO)	On	Stochastic	No
[93]	Proximal Policy Optimization (PPO)	On	Stochastic	No
[94]	Actor Critic using Kronecker-Factored Trust Region (ACKTR)	On	Stochastic	Yes
[95]	Soft Actor-Critic (SAC)	Off	Stochastic	Yes

Nevertheless, DDPG was used as presented in Algorithm 7 for all experiments in Part II. The reasoning behind using DDPG and not any of the algorithms in Table 3.1 is based on the observations made by Henderson et al. [96] about the difficulties in comparing DRL algorithms across different publications. Specifically, they attempt to address the following questions:

- How much do *hyperparameter settings* influence the reported algorithm performance?
- How does the *network architecture* of the policy and value functions affect the performance?
- How does *reward scaling* affect results and why is it used?
- How do *random seeds* affect the reported performance? Can results be distorted by averaging an improper number of trials?
- How do the *environment* properties affect variability in performance?
- Are commonly used *codebase* implementations comparable?

It is a well-known fact that *hyperparameter settings* and *network architecture* have significant effects on performance of any deep learning algorithm. Henderson et al. further show that *reward scaling* can have a large impact, and recommend a more principled approach, e.g. Pop-Art [79]. They also demonstrate that the variance between trials with different *random seeds* is enough to create statistically different performance distributions. Moreover, algorithm performance can vary across *environments* and it is not clear which is the universally best performing method. Most DRL papers present results in the form of learning curves, as some function of the return. However, without understanding what the returns actually indicate, these curves can be misleading since the algorithm may converge to local optima without ever reaching the desired goal. Consequently, the authors recommend the inclusion of a qualitative analysis of the results. This is observed in Paper A, where some reward definitions lead to suboptimal solutions. Finally, they find that implementation differences between *codebases* of the same algorithm can have drastic effects on performance, making it important to either use standardized codebases, or make the new code public together with all hyperparameter details [96]. In this work, the default DDPG implementation from the rlpyt library [97] was used.

3.6 RL Simulation Environments

Much like the ImageNet challenge for CV methods mentioned at the start of this chapter, several simulation environments were created in order to compare RL algorithms. To facilitate this comparison, OpenAI released the Gym [98] toolkit for developing RL simulation environments following the same formalism. This standardization makes testing RL algorithms in different applications much easier, as all tasks have the same Python interface. Notably, the environments used for benchmarking DRL in continuous control tasks by Duan et al. [89], were made available as part of Gym [98]. These included classic control problems from RL literature, as well as more complex locomotion tasks. Though the agents in these environments can be viewed as simplified robots, they do not actually correspond to any real-world hardware. It was with the publication of HER [83] that environments including real robotic systems like Fetch [99] and the Shadow Hand [100] were added to Gym.

Besides the tasks provided by OpenAI, many other third-party environments were created using the same standard. That is the case for the work introduced in Paper B which presents ReForm, a robot learning sandbox for DLO manipulation. Also included in the paper is an overview of environments for robotic manipulation, although some related work was missed such as robo-gym, which simulates multiple commercially available industrial robots using Gazebo [101]. A particularly important omission in the context of deformable object manipulation is the concurrent work by Huang et al. [102] which published the PlasticineLab environments after the final version deadline for Paper B. These, together with ReForm and SoftGym [38], consist of the only RL benchmarking environments currently available which address the challenges of DOM.

SoftGym, was released while the work on ReForm was underway. This library uses NVIDIA FleX as the physics engine, which can simulate soft objects such as clothes and ropes, as well as liquids. However, elastoplasticity is not available in any of the environments. Conversely, PlasticineLab does include an elastoplastic material model implemented using Taichi [103]. Nevertheless, all deformable objects in the provided environments behave like Plasticine. That includes the DLO manipulation task, where a rope is modeled as a long Plasticine piece. This is in contrast with the AGX Dynamics engine used for ReForm, which supports DLO models with a range of material properties.

CHAPTER 4

Concluding Remarks

Part I provided an overview of the research context in which the contributed papers of Part II are inserted. Chapter 1 started with a birds-eye view of deformable object manipulation and robot learning. This was followed by a more in-depth treatment of DLO manipulation in Chapter 2 and RL theory in Chapter 3. Finally, this chapter provides a summary of the paper contributions, in Section 4.1. To conclude, future directions currently being considered for the remainder of this graduate project are discussed in Section 4.2.

Before that, one might stop to reflect about how we approach DOM problems in our daily lives. Is trial-and-error learning such as RL, responsible for all human dexterity? Or are many manipulation skills learned by imitation? How are the goals for each task defined? When folding laundry, does the exact position of each fold matter, or is the purpose to flatten clothes while reducing storage area? How many tasks would humans be able to solve if instead of having compliant five-fingered hands, they had rigid two-fingered parallel grippers? Would tying shoelaces be easy or even possible? Which tasks are strictly dependent on vision or tactile feedback? These are a few of the many questions that arise when attempting to navigate the field of DOM from a robot learning perspective.

4.1 Contributions

The product of the work completed so far is summarized by the literature study included in Part I and the acceptance of two papers to the 2021 International Conference on Robotics and Automation (ICRA). The following sections describe the contributions of each paper included in Part II.

4.1.1 Paper A

This paper introduces a new shape control task for DLOs with elastoplastic properties. The goal is to deform a DLO into a desired shape, with the added difficulty that a permanent deformation may occur. In such a case, it becomes challenging to define a distance measure which uniquely guides the manipulation objective. RL offers a data-driven approach to tackle this problem without needing a model. However, it still requires a reward definition that adequately describes the goal. The key contribution in this work, is the proposal of a reward function based on a shape representation using discrete curvature and torsion. DDPG is applied to solve the manipulation task, in simulation experiments. Results show that to converge to the correct shape, the reward must include the proposed shape representation.

4.1.2 Paper B

This paper presents ReForm, a novel robot learning sandbox for deformable linear object manipulation. It is meant as a tool for testing and benchmarking DLO manipulation strategies. It consists of six environments representing important characteristics of deformable objects, with material properties ranging from pure elasticity to elastoplasticity. The sandbox also includes problems such as self-collisions. ReForm is a modular framework which allows choice of parameters such as end-effector DoFs, reward function and type of observation. Since vision data is supported, CV methods can also be tested, for example to address self-occlusion in DLO tracking.

In conclusion, an RL algorithm has been applied to learn control policies based on DNNs, for a set of deformable linear object manipulation tasks, thus partially fulfilling the objectives described in Section 1.3. Though all contributed work has been confined to simulation, force and vision data have also been utilized in this virtual setting.

4.2 Future work

As mentioned in Chapter 1, there is a hardware bottleneck when it comes to DOM. Therefore, soft manipulators are becoming commonplace for DOM tasks since they can adapt to the shape of deformable objects without needing dedicated software [104]. Tactile sensing, on the other hand is lagging considerably behind, and is viewed as one of the most important areas requiring further development, for DOM to succeed [2]. Nevertheless, there are several active lines of research on sensing technologies, such as GelSight [42] and many others [105]–[107]. Although hardware improvements have an important role to play in DOM, the research goal defined in Section 1.3 shifted the focus to software solutions. Based on observations made during the initial phase of this graduate project, the aim of upcoming work will be twofold: **i.** transfer policies learned in simulation to the real world and **ii.** attempt to reduce sample inefficiency by exploring model-based methods.

4.2.1 Simulation-to-Reality

Now that a sandbox for DLO manipulation tasks is available and has been tested with RL algorithms, the next challenge is transferring policies learned in simulation to a real robot. Sim2Real is an active research topic, meriting workshops at the Robotics: Science and Systems (RSS) conference both in 2019 and 2020 [108]. In RSS 2021, there was yet another workshop exclusively dedicated to simulation of deformable objects. Nevertheless, simulation accuracy is still quite limited, resulting in a reality gap which must be overcome by: *closing the gap*, through improved simulations and system identification; or *bridging the gap*, through methods which make use of simulated data that is not completely accurate.

Domain Randomization [109] is a simple approach for the latter, that consists of training DNNs with data generated from a distribution of simulation parameters, such as object shapes, rendering colors, etc. Similarly, Dynamics Randomization [110] has been shown to improve generalization of policies learned in simulation. However, such randomization methods significantly slow down training of the RL agent. Bengio et al. [111] proposed Curriculum Learning to to speed up training of DNNs by starting small, with simpler tasks and slowly increasing difficulty [112]. This notion was extended to RL for robotics in [113]. Future work will explore the effectiveness of such strategies.

4.2.2 Model-Based RL

The motivation to start with model-free RL was due to the modeling complexity of deformable objects, discussed in Section 2.1. However, there are many benefits to having a model and future work will include the exploration of model-based methods. This may be through model learning algorithms, briefly introduced in Section 1.2. Alternatively, RL algorithms which simultaneously learn a model of the environment dynamics and estimate value functions can be employed. Wang et al. [69] provide a comprehensive benchmark of model-based RL algorithms. However, biased models significantly limit the effectiveness of RL, leading to policies that exploit the model-bias.

While this thesis has addressed learning-based control with a focus on RL, there are many other promising approaches to solve DOM problems. For example, techniques from adaptive [114] and robust control [115] can be applied. Furthermore, Model Predictive Control (MPC) is an optimal strategy closely related to DP which can be used when a model is available. While DP consists of offline optimization methods to obtain an explicit closed-loop control policy, MPC is an online method which iteratively solves for the optimal open-loop trajectory over a finite time window, at each time step. Such model-based approaches can be combined with model-free methods to achieve better results, as demonstrated in [116], where MPC was combined with TRPO. All these methods are currently being considered as potential future directions.

References

[1] P. Heisler, P. Steinmetz, I. S. Yoo, and J. Franke, "Automatization of the cable-routing-process within the automated production of wiring systems," in *Energy Efficiency in Strategy of Sustainable Production III*, ser. Applied Mechanics and Materials, vol. 871, Trans Tech Publications Ltd, Nov. 2017, pp. 186–192.

[2] J. Zhu, A. Cherubini, C. Dune, D. Navarro-Alarcon, F. Alambeigi, D. Berenson, F. Ficuciello, K. Harada, X. Li, and J. Pan, "Challenges and outlook in robotic manipulation of deformable objects," *arXiv preprint arXiv: 2105.01767*, 2021.

[3] J. Sanchez, J.-A. Corrales, B.-C. Bouzgarrou, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 688–716, 2018.

[4] J. Peters, R. Tedrake, N. Roy, and J. Morimoto, "Robot learning," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 865–869, ISBN: 978-0-387-30164-8.

[5] J. H. Connell and S. Mahadevan, "Introduction to robot learning," in *Robot Learning*, Springer, 1993, pp. 1–17.

[6] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrik, A. Kargakos, L. Wagner, V. Hlaváč, T.-K. Kim, and S. Malassiotis, "Folding clothes autonomously: A complete pipeline," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1461–1478, 2016.

[7] M. J. Thuy-Hong-Loan Le, A. Landini, M. Zoppi, D. Zlatanov, and R. Molfino, "On the development of a specialized flexible gripper for garment handling," *Journal of Automation and Control Engineering Vol*, vol. 1, no. 3, 2013.

[8] Y. She, S. Wang, S. Dong, N. Sunil, A. Rodriguez, and E. H. Adelson, "Cable manipulation with a tactile-reactive gripper," in *Robotics: Science and Systems (RSS)*, 2020.

[9] S. Duenser, R. Poranne, B. Thomaszewski, and S. Coros, "Robocut: Hot-wire cutting with robot-controlled flexible rods," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 98–1, 2020.

[10] T. Tang, C. Wang, and M. Tomizuka, "A framework for manipulating deformable linear objects by coherent point drift," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3426–3433, 2018.

[11] J. Grannen, P. Sundaresan, B. Thananjeyan, J. Ichnowski, A. Balakrishna, M. Hwang, V. Viswanath, M. Laskey, J. E. Gonzalez, and K. Goldberg, "Untangling dense knots by learning task-relevant keypoints," in *Conference on Robot Learning (CoRL)*, 2020.

[12] T. Tang and M. Tomizuka, "Track deformable objects from point clouds with structure preserved registration," *The International Journal of Robotics Research*, p. 0 278 364 919 841 431, 2018.

[13] S. Sen, A. Garg, D. V. Gealy, S. McKinley, Y. Jen, and K. Goldberg, "Automating multi-throw multilateral surgical suturing with a mechanical needle guide and sequential convex optimization," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 4178–4185.

[14] M. Yan, Y. Zhu, N. Jin, and J. Bohg, "Self-supervised learning of state estimation for manipulating deformable linear objects," *IEEE Robotics and Automation Letters*, 2020.

[15] H. Wakamatsu and S. Hirai, "Static modeling of linear object deformation based on differential geometry," *The International Journal of Robotics Research*, vol. 23, no. 3, pp. 293–311, 2004.

[16] D. Berenson, "Manipulation of deformable objects without modeling and simulating deformation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2013, pp. 4525–4532.

[17] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, "Learning predictive representations for deformable objects using contrastive estimation," *arXiv preprint arXiv:2003.05436*, 2020.

[18] A. Zea, F. Faion, and U. D. Hanebeck, "Tracking elongated extended objects using splines," in *2016 19th International Conference on Information Fusion (FUSION)*, IEEE, 2016, pp. 612–619.

[19] M. Wnuk, C. Hinze, A. Lechler, and A. Verl, "Kinematic multibody model generation of deformable linear objects from point clouds," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 9545–9552.

[20] V. E. Arriola-Rios, P. Guler, F. Ficuciello, D. Kragic, B. Siciliano, and J. L. Wyatt, "Modeling of deformable objects for robotic manipulation: A tutorial and review," *Frontiers in Robotics and AI*, vol. 7, p. 82, 2020.

[21] H. Yin, A. Varava, and D. Kragic, "Modeling, learning, perception, and control methods for deformable object manipulation," *Science Robotics*, vol. 6, no. 54, 2021.

[22] P. Sundaresan, J. Grannen, B. Thananjeyan, A. Balakrishna, M. Laskey, K. Stone, J. E. Gonzalez, and K. Goldberg, "Learning rope manipulation policies using dense object descriptors trained on synthetic depth data," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 9411–9418.

[23] A. C. Ugural and S. K. Fenster, *Advanced mechanics of materials and applied elasticity*. Pearson Education, 2011.

[24] N. Lv, J. Liu, X. Ding, J. Liu, H. Lin, and J. Ma, "Physically based real-time interactive assembly simulation of cable harness," *Journal of Manufacturing Systems*, vol. 43, pp. 385–399, 2017.

[25] V. Fontanari, M. Benedetti, and B. D. Monelli, "Elasto-plastic behavior of a warrington-seale rope: Experimental analysis and finite element modeling," *Engineering Structures*, vol. 82, pp. 113–120, 2015.

[26] A. Remde and D. Henrich, "Direct and inverse simulation of deformable linear objects," in *Robot manipulation of deformable objects*, Springer, 2000, pp. 43–70.

[27] H. Wakamatsu, K. Takahashi, and S. Hirai, "Dynamic modeling of linear object deformation based on differential geometry coordinates," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2005, pp. 1028–1033.

[28] H. Wakamatsu, T. Yamasaki, A. Tsumaya, E. Arai, and S. Hirai, "Dynamic modeling of linear object deformation considering contact with obstacles," in *2006 9th International Conference on Control, Automation, Robotics and Vision*, IEEE, 2006, pp. 1–6.

[29] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep. 2004, pp. 2149–2154.

[30] E. Rohmer, S. P. N. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 1321–1326.

[31] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, `http://pybullet.org`, 2016–2020.

[32] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2012, pp. 5026–5033.

[33] E. Todorov, *Mujoco: Modeling, simulation and visualization of multi-joint dynamics with contact*, Roboti Publishing, Seattle, USA, 2021 [Online Access].

[34] E. Coumans, *Bullet 2.38 physics sdk manual*, Bullet Physics Library, 2021 [Online Access].

[35] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik, and S. Cotin, "SOFA: A Multi-Model Framework for Interactive Physical Simulation," in *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, ser. Studies in Mechanobiology, Tissue Engineering and Biomaterials, Y. Payan, Ed., vol. 11, Springer, Jun. 2012, pp. 283–321.

[36] C. Duriez, "Control of elastic soft robots based on real-time finite element method," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2013, pp. 3982–3987.

[37] Algoryx, *Agx dynamics user manual*, Umeå, Sweden, 2021 [Online Access].

[38] X. Lin, Y. Wang, J. Olkin, and D. Held, "Softgym: Benchmarking deep reinforcement learning for deformable object manipulation," in *Conference on Robot Learning (CoRL)*, 2020.

[39] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, *et al.*, "Sapien: A simulated part-based interactive environment," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 097–11 107.

[40] C. Gan, J. Schwartz, S. Alter, D. Mrowca, M. Schrimpf, J. Traer, J. D. Freitas, J. Kubilius, A. Bhandwaldar, N. Haber, M. Sano, K. Kim, E. Wang, M. Lingelbach, A. Curtis, K. T. Feigelis, D. Bear, D. Gutfreund, D. D. Cox, A. Torralba, J. J. DiCarlo, J. B. Tenenbaum, J. Mcdermott, and D. L. Yamins, "ThreeDWorld: A platform for interactive multi-modal physical simulation," 2021.

[41] M. Gazzola, L. Dudte, A. McCormick, and L. Mahadevan, "Forward and inverse problems in the mechanics of soft filaments," *Royal Society open science*, vol. 5, no. 6, p. 171 628, 2018.

[42] W. Yuan, S. Dong, and E. H. Adelson, "Gelsight: High-resolution robot tactile sensors for estimating geometry and force," *Sensors*, vol. 17, no. 12, p. 2762, 2017.

[43] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning (CoRL)*, 2018, pp. 734–743.

[44] A. X. Lee, H. Lu, A. Gupta, S. Levine, and P. Abbeel, "Learning force-based manipulation of deformable objects from multiple demonstrations," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 177–184.

[45] Z. Kappassov, J.-A. Corrales, and V. Perdereau, "Tactile sensing in dexterous robot hands," *Robotics and Autonomous Systems*, vol. 74, pp. 195–220, 2015.

[46] J. M. Romano, K. Hsiao, G. Niemeyer, S. Chitta, and K. J. Kuchenbecker, "Human-inspired robotic grasp control with tactile sensing," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1067–1079, 2011.

[47] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[48] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming:: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[49] M. Yu, H. Zhong, F. Zhong, and X. Li, "Adaptive control for robotic manipulation of deformable linear objects with offline and online learning of unknown models," *arXiv preprint arXiv:2107.00194*, 2021.

[50] J. Zhu, B. Navarro, P. Fraisse, A. Crosnier, and A. Cherubini, "Dual-arm robotic manipulation of flexible cables," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 479–484.

[51] R. Lagneau, A. Krupa, and M. Marchal, "Automatic shape control of deformable wires based on model-free visual servoing," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5252–5259, 2020.

[52] M. Ruan, D. Mc Conachie, and D. Berenson, "Accounting for directional rigidity and constraints in control for manipulation of deformable objects without physical simulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 512–519.

[53] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[54] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[55] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.

[56] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[57] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[59] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[60] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[61] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel, "Learning to Manipulate Deformable Objects without Demonstrations," in *Robotics: Science and Systems (RSS)*, Corvalis, Oregon, USA, Jul. 2020.

[62] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, and A. M. Agogino, "Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 2062–2069.

[63] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[64] B. Thananjeyan, A. Garg, S. Krishnan, C. Chen, L. Miller, and K. Goldberg, "Multilateral surgical pattern cutting in 2d orthotropic gauze with deep reinforcement learning policies for tensioning," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 2371–2378.

[65] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[66] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[67] D. Bertsekas, *Reinforcement and Optimal Control*. Athena Scientific, 2019.

[68] K. Doya, "Reinforcement learning in continuous time and space," *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[69] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," *arXiv preprint arXiv:1907.02057*, 2019.

[70] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[71] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2006, pp. 2219–2225.

[72] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[73] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[74] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[75] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[76] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[77] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, PMLR, 2015, pp. 448–456.

[78] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[79] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," *Advances in Neural Information Processing Systems*, vol. 29, pp. 4287–4295, 2016.

[80] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," *arXiv preprint arXiv:1812.02648*, 2018.

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

[82] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[83] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *31st International Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 5055–5065.

[84] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[85] S. Parisi, V. Tangkaratt, J. Peters, and M. E. Khan, "Td-regularized actor-critic methods," *Machine Learning*, vol. 108, no. 8, pp. 1467–1501, 2019.

[86] K. W. Cobbe, J. Hilton, O. Klimov, and J. Schulman, "Phasic policy gradient," in *International Conference on Machine Learning (ICML)*, PMLR, 2021, pp. 2020–2027.

[87] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, PMLR, 2016, pp. 1928–1937.

[88] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International Conference on Machine Learning (ICML)*, PMLR, 2018, pp. 1407–1416.

[89] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning (ICML)*, PMLR, 2016, pp. 1329–1338.

[90] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning (ICML)*, PMLR, 2018, pp. 1587–1596.

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

[92] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, PMLR, 2015, pp. 1889–1897.

[93] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[94] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *Advances in neural information processing systems*, vol. 30, pp. 5279–5288, 2017.

[95] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, PMLR, 2018, pp. 1861–1870.

[96] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[97] A. Stooke and P. Abbeel, "Rlpyt: A research code base for deep reinforcement learning in pytorch," *arXiv preprint arXiv:1909.01500*, 2019.

[98] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[99] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *Workshop on autonomous mobile service robots*, 2016.

[100] S. Robotics, "Shadow dexterous hand technical specifications," *Shadow Robot Company, London. www.shadowrobot.com*, 2013.

[101] M. Lucchi, F. Zindler, S. Mühlbacher-Karrer, and H. Pichler, "Robo-gym– an open source toolkit for distributed deep reinforcement learning on real and simulated robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[102] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan, "Plasticinelab: A soft-body manipulation benchmark with differentiable physics," in *International Conference on Learning Representations (ICLR)*, 2021.

[103] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, "Taichi: A language for high-performance computation on spatially sparse data structures," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–16, 2019.

[104] W. Wang, Y. Tang, and C. Li, "Controlling bending deformation of a shape memory alloy-based soft planar gripper to grip deformable objects," *International Journal of Mechanical Sciences*, vol. 193, p. 106 181, 2021.

[105] G. H. Büscher, R. Koiva, C. Schürmann, R. Haschke, and H. J. Ritter, "Flexible and stretchable fabric-based tactile sensor," *Robotics and Autonomous Systems*, vol. 63, pp. 244–252, 2015.

[106] G. Cheng, E. Dean-Leon, F. Bergner, J. R. G. Olvera, Q. Leboutet, and P. Mittendorfer, "A comprehensive realization of robot skin: Sensors, sensing, control, and applications," *Proceedings of the IEEE*, vol. 107, no. 10, pp. 2034–2051, 2019.

[107] H. Lee, K. Park, J. Kim, and K. J. Kuchenbecker, "Internal array electrodes improve the spatial resolution of soft tactile sensors based on electrical resistance tomography," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 5411–5417.

[108] S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, F. Golemo, M. Mozifian, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, *et al.*, "Perspectives on sim2real transfer for robotics: A summary of the rss 2020 workshop," *arXiv preprint arXiv:2012.03806*, 2020.

[109] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2017, pp. 23–30.

[110] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2018, pp. 3803–3810.

[111] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *International Conference on Machine Learning (ICML)*, PMLR, 2009, pp. 41–48.

[112] J. L. Elman, "Learning and development in neural networks: The importance of starting small," *Cognition*, vol. 48, no. 1, pp. 71–99, 1993.

[113] L. Hermann, M. Argus, A. Eitel, A. Amiranashvili, W. Burgard, and T. Brox, "Adaptive curriculum generation from demonstrations for sim-to-real visuomotor control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 6498–6505.

[114] J. Qi, W. Ma, D. Navarro-Alarcon, H. Gao, and G. Ma, "Adaptive shape servoing of elastic rods using parameterized regression features and auto-tuning motion controls," *arXiv preprint arXiv:2008.06896*, 2020.

[115] S. Hirai, T. Tsuboi, and T. Wada, "Robust grasping manipulation of deformable objects," in *Proceedings of the 2001 IEEE International Symposium on Assembly and Task Planning (ISATP2001). Assembly and Disassembly in the Twenty-first Century.(Cat. No. 01TH8560)*, IEEE, 2001, pp. 411–416.

[116] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 7559–7566.

Part II

Papers

PAPER A

Learning Shape Control of Elastoplastic Deformable Linear Objects

Rita Laezza, Yiannis Karayiannidis

Published in *International Conference on Robotics and Automation (ICRA)*,
 DOI: 110.1109/ICRA48506.2021.9561984

The layout has been revised.

Abstract

Deformable object manipulation tasks have long been regarded as challenging robotic problems. However, until recently very little work has been done on the subject, with most robotic manipulation methods being developed for rigid objects. Deformable objects are more difficult to model and simulate, which has limited the use of model-free Reinforcement Learning (RL) strategies, due to their need for large amounts of data that can only be satisfied in simulation. This paper proposes a new shape control task for Deformable Linear Objects (DLOs). More notably, we present the first study on the effects of elastoplastic properties on this type of problem. Objects with elastoplasticity such as metal wires, are found in various applications and are challenging to manipulate due to their nonlinear behavior. We first highlight the challenges of solving such a manipulation task from an RL perspective, particularly in defining the reward. Then, based on concepts from differential geometry, we propose an intrinsic shape representation using discrete curvature and torsion. Finally, we show through an empirical study that in order to successfully solve the proposed task using Deep Deterministic Policy Gradient (DDPG), the reward needs to include intrinsic information about the shape of the DLO.

1 Introduction

In recent years, there has been a growing interest in deformable object grasping and manipulation problems by the robotics community [1], [2]. This is due in part to their widespread across diverse applications as well as their increased complexity, when compared to rigid object tasks. These problems have been shown to be difficult to solve with classical approaches [1]. Consequently, learning-based methods are being explored as a more powerful alternative [2]. Intuitively, if a robot is to reach human-level dexterity, there may be a need for human-inspired learning. Reinforcement Learning is a particularly promising family of methods which seeks to make robots capable of learning through

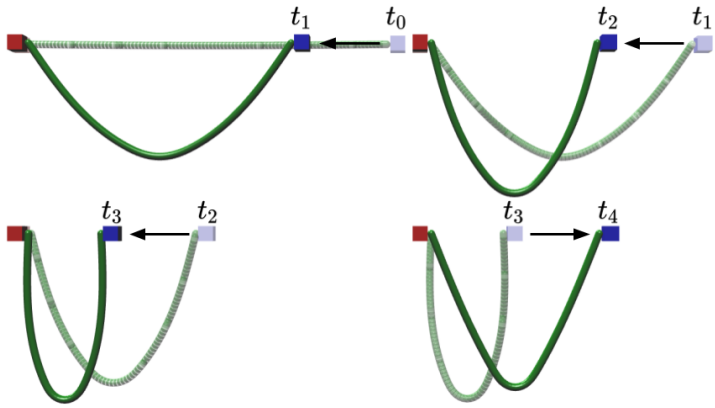


Figure 1: Simulation of DLO with plastic properties. By bending the object first inwards and then outwards, when returning the gripper to the same position at t_2 and t_4, the shape of the DLO is different due to (permanent) plastic deformation. This motion is executed along a single DoF.

experience [3]. RL has been proven successful in solving complex games, such as Go [4], as well as robotic control tasks, such as pick and place [5].

Contrary to grasping and manipulation of rigid objects, which have been extensively addressed in the robotics literature, non-rigid objects have been largely overlooked [1]. Though some of the same methods can be extended to particular types of deformable objects, there are still many problems left unsolved [1]. Most notably, while manipulation of rigid objects focuses mainly on controlling their pose, when manipulating deformable objects it is often their shape which needs to be controlled [2]. Furthermore, dealing with materials which are highly deformable or with elastoplastic properties, makes modeling and sensing of these objects a difficult challenge. In addition, most work on deformable object manipulation has focused on specialized tasks, from applications like robotic surgery, food processing and fabric manufacturing [1]. While this is a practical choice to solve real-life problems, the solutions are often not general [6]. With this work we propose a strategy for explicit shape control of elastic and/or plastic objects, which could potentially be applied to a large range of problems.

According to classification criteria suggested by Sanchez et al. [1], deformable objects can be categorized based on their mechanical properties, i.e. low or high compression strength and their geometric properties, i.e. linear, planar or solid shapes. In this work we focus on Deformable Linear Objects with elastoplastic properties. Objects that fit into this category include metal wires, rods and cables, found across multiple applications including medical, industrial, and household services. DLOs are an appealing choice for their relative geometric simplicity, making them more efficient to simulate but still complex to manipulate. Within this class, we found that the manipulation of objects with elastoplastic properties is yet to be studied, with most of the literature focusing on purely elastic DLOs or low compression strength DLOs such as ropes, which exhibit plastic behavior [1]. The reason elastoplastic materials make for a particularly difficult class of objects is due to their non-linear behavior, starting as purely elastic up to a yield point, after which transitioning to a plastic domain. This is illustrated with an example of DLO manipulation in Figure 1, where plastic deformation occurs after the initial elastic deformation, leading to potentially irreversible changes. A practical application of elastoplastic wires can be found in the production of orthodontic braces. Our work opens up the potential for RL-based automation of the shaping process, since this is still done manually to a large extent [7].

To address the problem of robotic manipulation of elastoplastic DLOs, we are interested on the ability to learn velocity control policies in a model-free fashion. To that end, we have implemented a simulation environment with a new shape control problem. The control policy is learned in task space and controlled by a Cartesian gripper with varying Degrees of Freedom (DoFs). The gripper grasps the object either with a fixed grasp or a flexible pinch, allowing rotation around one axis (i.e. hinge constraint). Since we aim to learn continuous actions, a policy gradient method is used, namely DDPG [8].

As our main contribution, we present a detailed evaluation of the proposed shape control problem from a Reinforcement Learning perspective. We begin by presenting the challenges of designing a reward signal. We propose shape representations using concepts from discrete differential geometry, namely curvature and torsion. Based on these, we evaluate three dense reward functions in a rigorous empirical study. Further, the impact of parameters such as mechanical properties of the DLO, number of controlled DoFs and type of grasp is also studied.

2 Related Work

To date, ropes or rope-like objects are the most researched group of DLOs in robotic manipulation. Common problems involving ropes include knot tying, untangling, threading and reaching goal-configurations on a flat surface [1]. While all of these present interesting challenges, only the latter represents an **explicit shape control** problem. For the others, what matters is not the final geometric deformation, but the configuration of the DLO, relative to itself, or other objects. Within these **implicit shape control** problems, the work by Berenson [9], recently extended in [10], proposes promising methods which preclude the use of physical simulation.

Deformable objects simulation is still an active research topic. A great part of advances in the field come from the computer graphics community, such as Pai [11], that used a Cosserat formulation to develop fast simulation algorithms. There have also been efforts to model DLOs with the intent to solve deformation tasks, more notably by Wakamatsu et al. [12], where a method based on differential geometry was used for motion planning. Other common DLO modeling approaches include Finite Element Methods (FEM) and Mass-Spring-Damper (MSD) systems [1].

Sensing and state estimation of non-rigid objects also presents a challenge which is often tackled separately [13], [14]. However, recent work on robotic shape control of DLOs combines different vision-based state estimation methods with control strategies. Yan et al. [15] used self-supervised learning to estimate the state of a rope resting on a tabletop, controlled by a single-arm. The manipulation was done by successive grasping and planning, after each state estimation step. Zhu et al. [6] used Fourier series to model the DLO shape and successfully deformed flexible cables into desired shapes, using a velocity controlled dual-armed robot. There have also been different approaches for state representations of a DLO's shape, including node-graphs, Frenet coordinate frames and Kirchoff elastic rods [1], [12], [16]. More recently, using deep learning techniques has opened up the possibility to learn directly from the high-dimensional raw image data [17]. This can be used in end-to-end strategies, where robot joint velocities are obtained directly from pixels.

We conclude this section by highlighting works which applied RL for deformable object manipulation tasks. Clomé et al. [18] first implemented a policy improvement strategy with path integrals to manipulate a scarf around a mannequin's neck. More recently, RL was used in robot-assisted endovas-

cular catheterization [19]. Both of these works employ Dynamic Movement Primitives (DMPs) and Learning from Demonstration (LfD). Matas et al. [17] produced promising results in cloth-manipulation using a state-of-the-art RL algorithm. Their work was formulated both in an end-to-end manner and for sim-to-real transfer. They used a variation of Deep Deterministic Policy Gradient, named DDPG from Demonstrations (DDPGfD) which seeds the learning with expert demonstrations. Conversely, Wu et al. [20] proposed to solve pick-and-place tasks of deformable objects completely from scratch. In contrast with our work, [17], [18] and [19] covered implicit shape control problems, while [20] did not address permanent deformation.

3 Background

Before presenting the proposed shape control problem, we introduce the necessary technical background on RL in 3.1 and DLO simulation in 3.2.

3.1 Reinforcement Learning

In RL, control problems are framed as Markov Decision Processes. We consider an infinite-horizon discounted MDP, defined as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where γ is the discount factor and $\mathcal{S}$ and $\mathcal{A}$ are continuous state and action spaces, respectively. For most real-life problems this MDP is unknown since the probability density function $p(s_{t+1}|s_t, a_t)$, depends on an environment which is difficult to model. This function represents the probability of transitioning to state s_{t+1}, given the current state s_t and action a_t, with $s_t, s_{t+1} \in \mathcal{S}$ and $a_t \in \mathcal{A}$. Further, in practical applications, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, is defined based on the desired task, taking the environment into consideration. The reward at each transition, r_t, is assumed to be a bounded scalar [21]. To provide a measure of success, the return at time t is defined as the sum of discounted future rewards:

$$G_t = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k) \tag{A.1}$$

In policy gradient methods the objective is to find an optimal stochastic policy, $\pi_\vartheta : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, which maps states to action probabilities. The optimal parameterized policy maximizes the expected return, i.e. $J(\pi) = \mathbb{E}[G_0|\pi]$. For

a specific state s_t and action a_t, the expected return is defined as the action-value function Q^π:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_k, s_k \sim \rho^\pi, a_k \sim \pi}[G_t | s_t, a_t] \tag{A.2}$$

with $k \geq t$ and ρ^π the state distribution under policy π.

As the name indicates, DDPG learns a deterministic policy $\mu_\vartheta : \mathcal{S} \rightarrow \mathcal{A}$, instead of a stochastic one. This algorithm is considered an actor-critic method, because the policy (actor) parameters are updated based on an estimated value function (critic) [8]. Both actor and critic are modeled as Deep Neural Networks (DNNs). Parameters $\vartheta \in \mathbb{R}^n$, are updated via stochastic gradient ascent, to maximize the Q_φ value, with $\varphi \in \mathbb{R}^m$. The policy update is calculated based on the current estimate of the Q value:

$$\nabla_\vartheta J(\mu_\vartheta) \approx \mathbb{E}_{s_t \sim \rho^\mu}[\nabla_\vartheta \mu_\vartheta(s) \nabla_a Q^\mu(s, a)|_{s=s_t, a=\mu_\vartheta(s_t)}] \tag{A.3}$$

Parameters of the Q_φ network are updated according to the Bellman equation, by minimizing the loss $\mathcal{L}(\varphi)$:

$$\begin{aligned} \mathcal{L}(\varphi) &= \mathbb{E}_{s_t \sim \rho^\mu, a_t \sim \mu, r_t}\left[(Q_\varphi(s_t, a_t) - y_t)^2\right] \\ y_t &= r(s_t, a_t) + \gamma Q_\varphi(s_{t+1}, \mu(s_{t+1})) \end{aligned} \tag{A.4}$$

To ensure sufficient exploration while experience is being sampled, noise is added to the actor policy $\beta(s_t) = \mu_\vartheta(s_t) + \mathcal{N}$, effectively making this an off-policy method. Practically, this means that the states in equations (A.3) and (A.4) are sampled from ρ^β instead of ρ^μ. Moreover, experience sampled by following the exploration policy β is stored in a replay buffer, as tuples (s_t, a_t, r_t, s_{t+1}). Actor and critic networks are updated by uniformly sampling mini-batches from the replay buffer, which helps mitigate problems such as learning from temporally correlated data (environment steps are not i.i.d.) and catastrophic forgetting [8]. The replay ratio defines the number of gradient updates per environment step (i.e. how much experience is trained on before being discarded). To increase stability, two sets of networks are kept so that actor and critic updates are done with respect to slow-changing target networks, with parameters ϑ' and φ'. To that end, soft updates $\varphi' \leftarrow \lambda\varphi + (1 - \lambda)\varphi'$ and $\vartheta' \leftarrow \lambda\vartheta + (1 - \lambda)\vartheta'$ are used for each parameterized function, with $\lambda << 1$.

3.2 DLO Simulation

Although our aim is to implement robot learning in real-life experiments, it can be intractable to train RL algorithms directly in a real robot, since they require a lot of sampled experience. This is especially true when learning from scratch and using model-free methods, such as DDPG, which are notoriously sample inefficient [21]. It is therefore common-practice to tackle problems first in simulation, and later apply sim-to-real transfer. With this in mind, we have implemented a virtual environment, to evaluate the potential of these methods for deformable object manipulation.

When choosing a physics engine, there are many factors to consider, such as accuracy, speed and development time. One requirement added by our particular application is the need for deformable object simulation capabilities. The robotics and classical control environments available in Gym [22] were implemented using MuJoCo (Multi-Joint dynamics with Contact) [23], which seems to be the predominant choice in the Reinforcement Learning community. It offers support for three types of soft bodies, namely rope, cloth and sponge-like 3D objects. Bullet is the preferred open source alternative, which is supported both by Gazebo and V-REP [24]. It provides limited functionalities, although it was used for cloth simulation in [17]. SOFA (Simulation Open Framework Architecture) [25] on the other hand is a framework targeted at interactive computational medical simulation, with good support for soft tissues. Nevertheless, we found that for DLO simulations, AGX Dynamics provides the best set of tools.

AGX Dynamics also offers real-time rendering and a `Cable` class which consists of a lumped element model with support for elastoplastic deformations [26], [27]. Further, it provides the possibility to define the object's `Material` with physically motivated mechanical properties such as Poisson's ratio, Young's modulus and the yield point, where there is the transition between elastic and plastic deformation, as illustrated in Figure 2. On the other hand, purely elastic DLOs exhibit linear behavior which makes robotic shape control tasks significantly simpler.

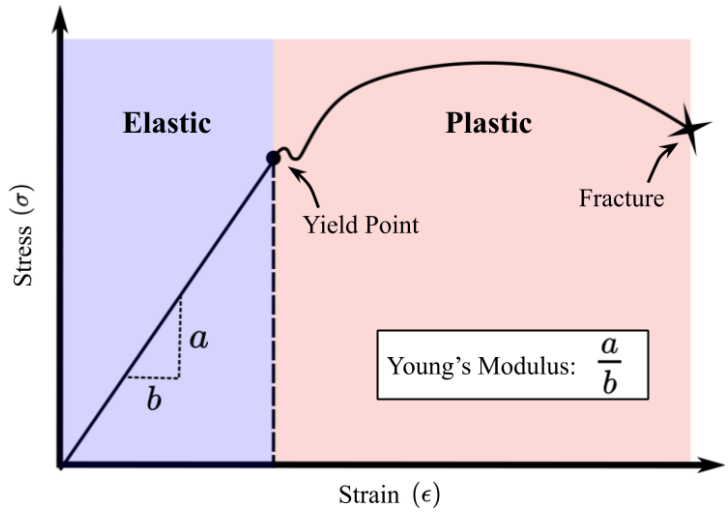
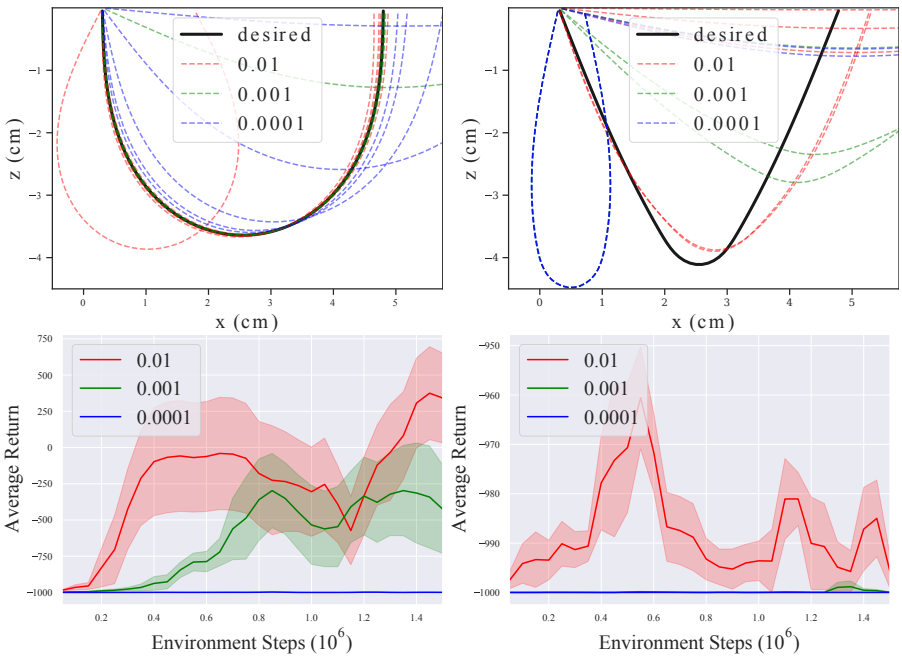


Figure 2: Illustration of typical stress-strain curve of material with elastic and plastic properties. Blue region consists of the (linear) elastic domain, while the red region consists of the (nonlinear) plastic domain.

4 Problem Statement

We propose a shape control problem of an elastoplastic DLO, with two grippers holding it in free space (without obstacles), as shown in Figure 1. The goal is to deform the DLO into a desired shape from an undeformed starting state. For simplicity, one of the grippers is static (red) while the other is able to move (blue). The control input is the linear velocity of a Cartesian gripper along each controlled DoF. This can be seen as task space velocity-resolved control of a robotic arm. We consider different number of controlled DoFs, which affects the size of both the state and action spaces. A perfect grip without translational slippage is assumed, with two modalities: hinge or lock. The former passively allows rotation about one axis of the gripping point, while the latter is completely fixed, leading to more pronounced deformations.

In order to successfully apply RL to any application, a reward signal must be designed such that it encodes the actual goal, without inadvertently leading to high rewards in non-goal states [28]. For the proposed shape control problem, the goal can be described as a perfect overlap between the state of



(a) Elastic DLO **(b)** Elastoplastic DLO

Figure 3: Learning was possible for the purely elastic DLO but failed for the elastoplastic case, as shown on the top row with the final shapes obtained for 5 trials with different success thresholds. Bottom row shows the average return during training, highlighting the effect of threshold selection, with complete failure of learning for higher accuracies. Three threshold values were tested and shaded area shows standard deviation.

the achieved and the desired DLO. This is a challenging task compared to rigid body problems, where the state of an object can be summarized by its pose in $\mathbb{R}^6$. On the other hand, a perfect match for a DLO requires a state representation at least in $\mathbb{R}^{3N}$, where N is the number of discrete points used to describe the DLO's shape as a point cloud. If we consider a sparse reward where a positive scalar is attributed only when the goal is reached, this can be problematic due to two main reasons:

i. When is the shape reached? This requires some distance measure which is intimately related with the deformable linear object's shape representation.

The simplest choice is to take the Euclidean distance between discrete points of the desired and achieved deformations. However, this also requires a choice of success threshold, which affects both the learning process and the accuracy of the achieved shape.

ii. With an increased state space, exploration becomes more challenging, particularly with sparse rewards. Indeed, this may result in what is called the *plateau problem*, in which the agent never experiences a positive reward during training, leading to failure to learn [28]. Note that the more complex the shape, or the greater accuracy is desired, the larger $N \gg 6$ must be, resulting in a larger state space.

To demonstrate these challenges we consider the proposed shape control problem, with 1 DoF control. A DDPG agent is trained with a reward of 1 attributed only when the Euclidean distance between the desired and the achieved shape is within a given threshold; otherwise, the agent receives a reward of -1 at every step, encouraging the agent to reach the goal as fast as possible. Positive rewards are therefore sparse. Figure 3 (a) shows the results for an elastic DLO where the 0.001 threshold leads to good results but the larger threshold leads to inaccurate shapes and the lower one hinders learning. Figure 3 (b) shows that for the elastoplastic DLO, even with the highest threshold i.e. lowest accuracy, the agent was unable to learn, with all trials leading to wrong shapes.

5 Shape Representation

Sensing of deformable objects is a challenging research topic. In Section 2, we list some of the state estimation methods that have been used to track the shape of DLOs. However, here we do not focus on estimation, but rather on the choice of shape representation. Given that we work in simulation, the state of the DLO in Euclidean space can be easily summarized as a point cloud with the coordinates of the lumped elements making up the object. We leave the task of extracting this point cloud from vision data as future work. From this simple representation we present useful concepts from differential geometry that can better describe the intrinsic shape of a DLO.

If we consider the point could of a DLO to be a discrete curve $c : \mathbb{N} \to \mathbb{R}^3$ with $N \geq 4$ points, it is possible to find a shape representation based on the notions of curvature and torsion. As shown in Figure 4, the discrete

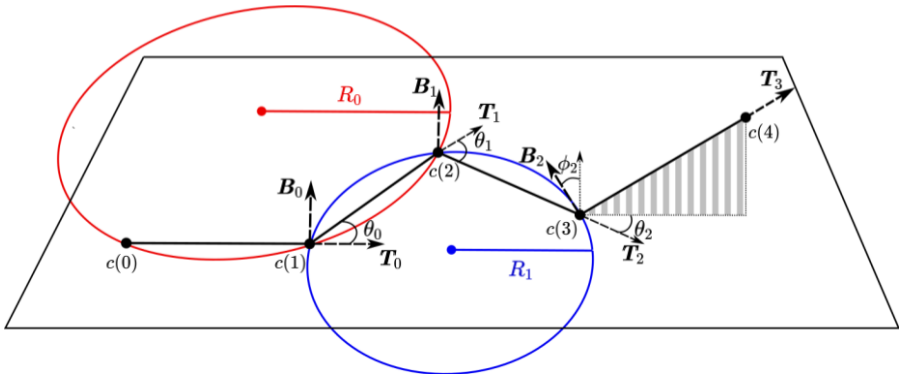


Figure 4: Discrete curve described by points $c(i)$, tangent vectors $\boldsymbol{T}_i$, binormal vectors $\boldsymbol{B}_i$ and angles θ_i, ϕ_i. Two osculating circles are illustrated to show the inverse relationship between radius R_i and curvature κ_i.

curvature κ_i can be described through the circumscribed osculating circle. For three consecutive (noncollinear) points, there is a unique circle circumscribing them, with radius $R_i > 0$. Curvature is defined as $\kappa_i = 1/R_i$, and for discrete curves it can be approximated as:

$$\kappa_i = \frac{2}{l} \tan \frac{\theta_i}{2} \approx \frac{\theta_i}{l}, \quad \text{with } \theta_i \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \tag{A.5}$$

where l is the segment length, and θ_i is the angle between tangent vectors of two consecutive segments. For collinear points the curvature is zero. Further the discrete torsion can be approximated as,

$$\tau_i = \frac{2}{l} \tan \frac{\phi_i}{2} \approx \frac{\phi_i}{l}, \quad \text{with } \phi_i \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \tag{A.6}$$

where ϕ_i is the angle between two consecutive binormal vectors. It is assumed that all segments have equal length [29]. To obtain the exterior angles, for each pair of adjacent points the tangent vector needs to be computed:

$$\boldsymbol{T}_i = \frac{c(i+1) - c(i)}{l}, \quad i = 0, \ldots, N-1 \tag{A.7}$$

Then, for each pair of consecutive tangent vectors, the curvature angle can be

obtained,

$$\theta_i = \arccos\left(\frac{\boldsymbol{T}_i \times \boldsymbol{T}_{i+1}}{||\boldsymbol{T}_i|| \cdot ||\boldsymbol{T}_{i+1}||}\right), \quad i = 0, \ldots, N-2 \tag{A.8}$$

which is enough to approximate the discrete curvature κ_i, as in equation (A.5). For the torsion, it is further necessary to compute the binormal vectors, which are orthogonal to the plane defined by the tangent and normal vectors. This can also be computed based on the plane characterized by two consecutive tangent vectors:

$$\boldsymbol{B}_i = \boldsymbol{T}_i \times \boldsymbol{T}_{i+1}, \quad i = 0, \ldots, N-2 \tag{A.9}$$

Finally, the torsion angle can be computed as,

$$\phi_i = \arccos\left(\frac{\boldsymbol{B}_i \times \boldsymbol{B}_{i+1}}{||\boldsymbol{B}_i|| \cdot ||\boldsymbol{B}_{i+1}||}\right), \quad i = 0, \ldots, N-3 \tag{A.10}$$

Based on the definitions presented in this section we move on to Section 6 where we formulate different MDP state and reward definitions which can be used to solve the proposed shape control problem.

6 RL Formulation

We define the action space $\mathcal{A}$, such that $a \in [-1, 1]$ and its dimensionality depends on the number of controlled degrees of freedom. Outputs from the policy DNN are then rescaled into viable velocity commands. The state s includes the position and velocity of the end-effector, denoted by $\boldsymbol{p}_{ee}, \boldsymbol{v}_{ee} \in \mathbb{R}^3$ respectively. Further, depending on the reward definition $r(s)$, the achieved curve $\boldsymbol{c} = [c(i), \ldots, c(N)]$, the achieved curvature $\boldsymbol{\kappa} = [\kappa_i, \ldots, \kappa_{N-2}]$ and/or torsion $\boldsymbol{\tau} = [\tau_i, \ldots, \tau_{N-4}]$ may also be included in the state definition.

As mentioned in Section 4, the simplest distance measure is the Euclidean distance $L(\boldsymbol{c})$ between the achieved curve $\boldsymbol{c}$ and the desired curve $\bar{\boldsymbol{c}}$. This leads to our first reward function:

$$r_t(\boldsymbol{c}_t) = -L_t(\boldsymbol{c}_t) = -||\boldsymbol{c}_t - \bar{\boldsymbol{c}}||_2 \tag{A.11}$$

For planar deformations, an alternative reward function can be defined based on the desired curvature $\bar{\boldsymbol{\kappa}}$:

$$r_t(\boldsymbol{\kappa}_t) = -L_t(\boldsymbol{\kappa}_t) = -||\boldsymbol{\kappa}_t - \bar{\boldsymbol{\kappa}}||_2 \tag{A.12}$$

This reward can also be extended with the distance between the achieved $\boldsymbol{\tau}$ and desired $\bar{\boldsymbol{\tau}}$ torsion, for 3D deformations. Finally, we formulate a weighted reward function which combines the rewards from equations (A.11) and (A.12):

$$r_t(\boldsymbol{c}_t, \boldsymbol{\kappa}_t) = -(1-\alpha)L_t(\boldsymbol{c}_t) - \alpha L_t(\boldsymbol{\kappa}_t) \tag{A.13}$$

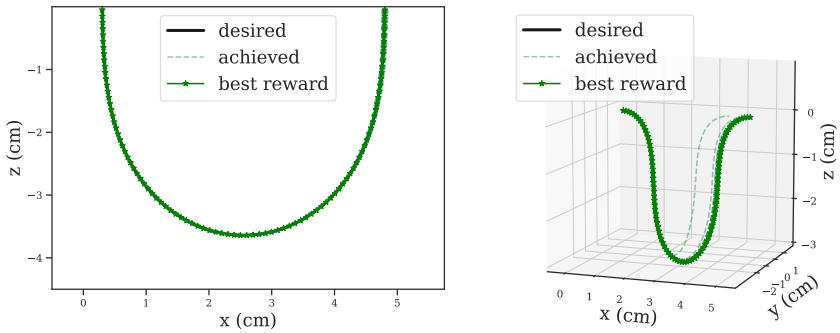
with $\alpha \in [0, 1]$. We denote variables dependent on the current environment step with subscript t.

In Section 7 we evaluate the performance of the proposed dense reward functions. To that end, we concatenate the shape representations used in the reward i.e. $\boldsymbol{c}_t$, $\boldsymbol{\kappa}_t$ and/or $\boldsymbol{\tau}_t$, with $\boldsymbol{p}_{ee}$ and $\boldsymbol{v}_{ee}$ in a one-dimensional state vector.

7 Experimental Results

To generate the goal shapes, a 5th order polynomial trajectory is used such that plastic deformation occurs, when simulating an elastoplastic DLO. The same trajectory leads to different shapes depending on the type of grip and material properties of the object. The gripping points on the DLO are simulated as being attached on each extremity to a rigid object, by either a `Hinge` or a `Lock` constraint. We further use three `Prismatic` constraints, to simulate a Cartesian gripper. The deformable linear object is modeled as an Aluminum cable, which is 10cm long and has a radius of 1mm. The resolution of `Cable` is set to 1000 segments per meter, while its Young's Modulus is set to 69MPa and its Poisson's ratio to 0.35. For the elastoplastic behavior, a yield point of 50MPa is defined in the bend direction of the DLO. Furthermore, the gripper is constrained by limiting its force range. The simulation time-step is set to 0.01s, while actions are applied every second step leading to a control frequency of 50 Hz.

In this work we evaluate the proposed shape control problem and the challenges of reward signal design using DDPG. For our experiments, the open source rlpyt [30] codebase is used. The actor and critic DNNs have the same architecture, namely two hidden layers with $[400, 200]$ neurons. The Adam optimization algorithm is used for gradient updates with learning rates of 1×10^{-4} and 1×10^{-3} for actor and critic, respectively. A batch size is set to 1024, sampled uniformly from the replay buffer with 5×10^5 tuples, and



(a) Hinge **(b)** Lock

Figure 5: Results of ten trained policies to reach a desired shape with a purely elastic DLO. The same 1 DoF trajectory was implemented to generate both goals, although in (a) a hinge grip was used allowing for a planar deformation while in (b) the grip was fixed, leading to a 3D deformation.

replay ratio of 64. Other important hyperparameters include the discount factor, $\gamma = 0.99$ and soft update rate, $\lambda = 0.01$.

For each algorithm, 10 trials are performed and the results averaged. For each trial, 20 parallel agents were used to gather experience, each with a different random seed. We present the final shapes obtained for each trial, trained with 1 million environment samples. Note however that this fixed from the start and in simpler tasks the algorithm converged earlier. Conversely, for the higher dimensional tasks training may have been insufficient.

To highlight the challenge of elastoplasticity, we first evaluate the performance of DDPG for the same shape control problem applied to a purely elastic DLO. Results are shown in Figure 5. In this case the agent is able to easily learn the control policy, using reward (A.11).

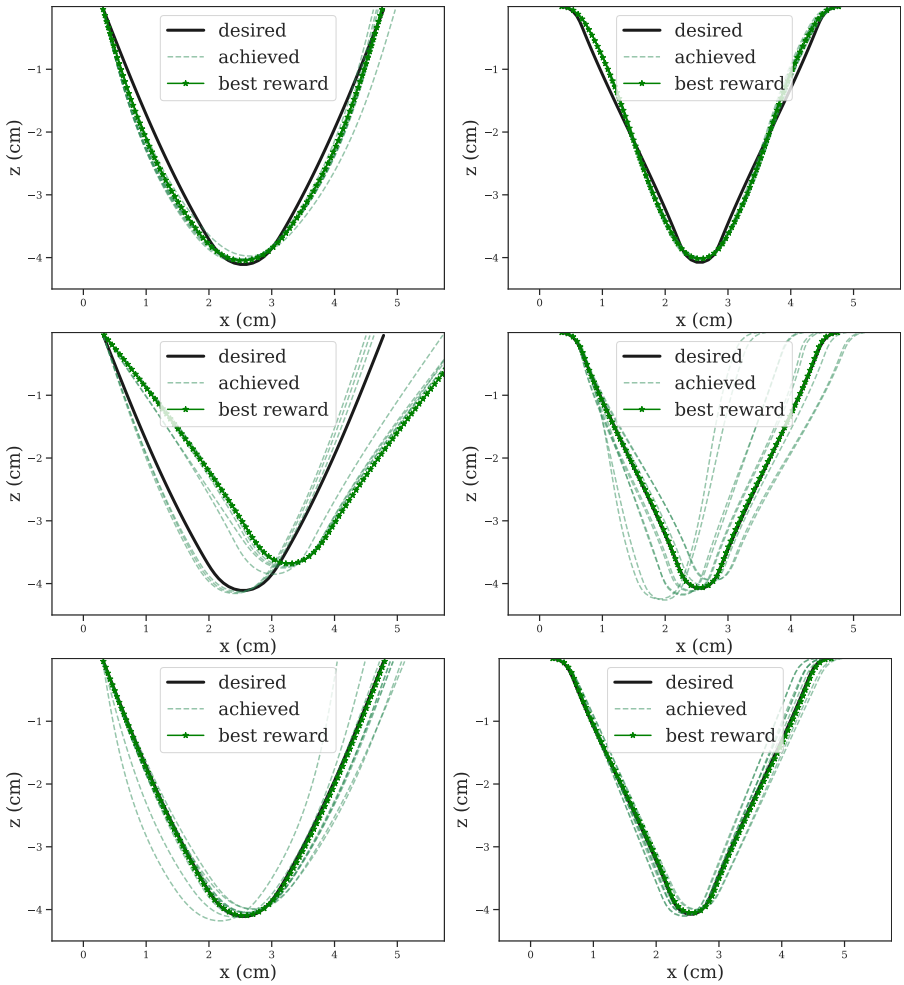
Table 1 summarizes the results obtained with different reward functions, for an elastoplastic DLO. The Euclidean distance $L_T(\boldsymbol{c}_T)$ was measured for each shape achieved after executing the learned control policies, with T denoting the final time step. Reward (A.11) resulted in the best average distance however, as seen in Figure 6 (top row), the learned policies do not reach the desired shapes with permanent deformation. Instead the DDPG agent gets stuck in a local reward maximum, where the gripper is in the correct position, but the DLO has an incorrect shape. On the other hand, using

Table 1: Results for each reward proposed in 6, showing the final distance, $L_T(\boldsymbol{c}_T)$. Mean, standard deviation and best results are listed.

r_t	**Hinge**		**Lock**	
	mean $\pm$ σ	best	mean $\pm$ σ	best
(A.11)	0.0132 ± 0.0022	0.0100	0.0077 ± 0.0003	0.0073
(A.12)	0.0561 ± 0.0298	0.0087	0.0298 ± 0.0231	**0.0010**
(A.13)	0.0143 ± 0.0106	**0.0019**	0.0077 ± 0.0043	0.0026

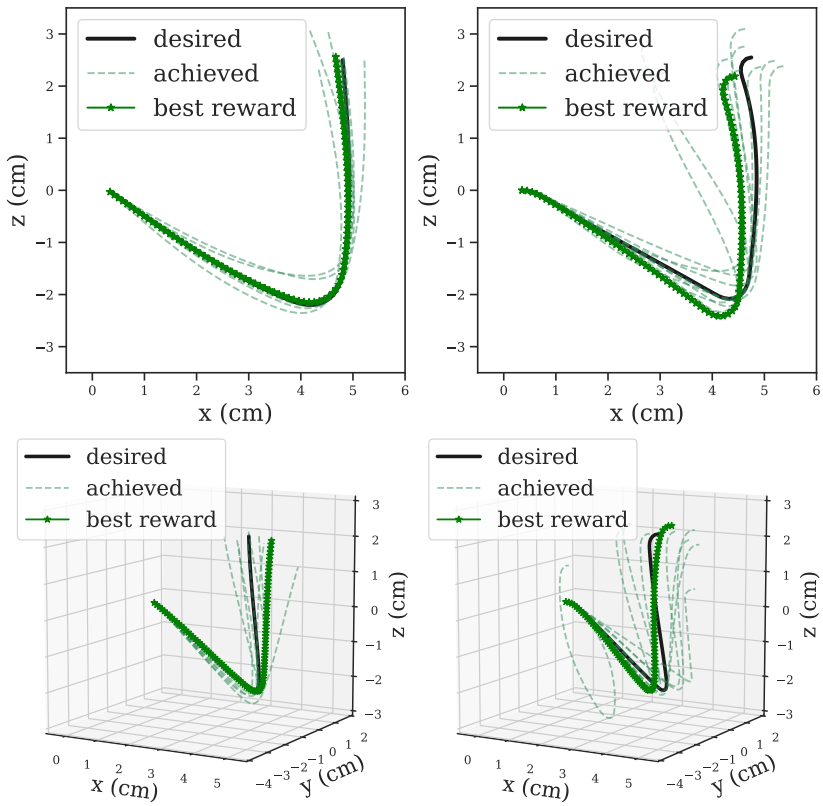
reward function (A.12) leads to the worst average distance between achieved and desired shapes. This is also evident in Figure 6 (middle row) however, the policies successfully learn to reach the desired permanent deformation. Note that for the pinch grip, this reward function has another local maximum which is far from the goal shape. Finally, we weight the two previous functions in reward (A.13). With $\alpha = 0.5$, this results in an average distance similar to the one obtained with reward function (A.11), while also reaching the correct plastic deformation, as shown in 6 (bottom row).

Finally, we evaluate the performance of DDPG with reward function (A.13) for 2 and 3 DoFs. Results show how the shape complexity affects the learning outcome. As seen in Figure 7, the lock grip shape is more challenging to achieve. Note that for the 3D deformation, torsion $\boldsymbol{\tau}$ was included in the state representation and reward function. As expected, the shapes are more difficult to reach, due to the increased state and action spaces, however the learned policies still learn to plastically deform the DLO.



(a) Hinge **(b)** Lock

Figure 6: Final shapes achieved for 1 DoF problems. Three state representations and respective rewards are shown, namely reward (A.11) - top, reward (A.12) - middle and reward (A.13) - bottom. All ten trained policies are shown and shape leading to best reward is highlighted.



(a) Hinge **(b)** Lock

Figure 7: Results with higher DoFs. The best $L_T(\boldsymbol{c}_T)$ distances for the 2 DoF control (top) were: $[0.0030, 0.0138]$; while for the 3 DoF control (bottom) they were: $[0.0183, 0.0228]$.

8 Concluding Remarks

We have presented a new shape control problem for elastoplastic DLOs, highlighting its implementation challenges in the context of RL. We first presented the difficulties of designing a reward that encodes the correct goal, while also being resistant to local maxima. To that end we introduced a DLO shape representation, based on curvature and torsion of a discrete curve. This led to three alternative dense reward functions which were empirically compared.

Our results showed that for an elastic DLO, the 1 DoF shape control problem can be easily solved based on the Euclidean distance of the desired and achieved curves. However, with elastoplastic DLOs, the reward must also include a distance measure to the desired curvature and torsion. Finally, we evaluated the proposed weighted reward function with 2 and 3 controlled DoFs, leading to more challenging exploration, but still converging to reasonable shapes.

For future work, we will use the proposed reward in multi-goal RL, to obtain a general solution to this type of shape control problems. Further, the problem of re-grasping and more complex shapes will be addressed. Ultimately the greatest challenge ahead is to bring our results from simulation into the real-world. This is planned to be done in a sim-to-real approach.

References

[1] J. Sanchez, J.-A. Corrales, B.-C. Bouzgarrou, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 688–716, 2018.

[2] F. F. Khalil and P. Payeur, "Dexterous robotic manipulation of deformable objects with multi-sensory feedback-a review," in *Robot Manipulators Trends and Development*, IntechOpen, 2010.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *31st International Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 5055–5065.

[6] J. Zhu, B. Navarro, P. Fraisse, A. Crosnier, and A. Cherubini, "Dual-arm robotic manipulation of flexible cables," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 479–484.

[7] Z. Xia, H. Deng, S. Weng, Y. Gan, J. Xiong, and H. Wang, "Development of a robotic system for orthodontic archwire bending," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 730–735.

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

[9] D. Berenson, "Manipulation of deformable objects without modeling and simulating deformation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2013, pp. 4525–4532.

[10] M. Ruan, D. Mc Conachie, and D. Berenson, "Accounting for directional rigidity and constraints in control for manipulation of deformable objects without physical simulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 512–519.

[11] D. K. Pai, "Strands: Interactive simulation of thin solids using cosserat models," in *Computer Graphics Forum*, Wiley Online Library, vol. 21, 2002, pp. 347–352.

[12] H. Wakamatsu and S. Hirai, "Static modeling of linear object deformation based on differential geometry," *The International Journal of Robotics Research*, vol. 23, no. 3, pp. 293–311, 2004.

[13] J. Schulman, A. Lee, J. Ho, and P. Abbeel, "Tracking deformable objects with point clouds," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2013, pp. 1130–1137.

[14] S. Javdani, S. Tandon, J. Tang, J. F. O'Brien, and P. Abbeel, "Modeling and perception of deformable one-dimensional objects," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2011, pp. 1607–1614.

[15] M. Yan, Y. Zhu, N. Jin, and J. Bohg, "Self-supervised learning of state estimation for manipulating deformable linear objects," *IEEE Robotics and Automation Letters*, 2020.

[16] T. Bretl and Z. McCarthy, "Quasi-static manipulation of a kirchhoff elastic rod based on a geometric analysis of equilibrium configurations," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 48–68, 2014.

[17] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning (CoRL)*, 2018, pp. 734–743.

[18] A. Colomé, A. Planells, and C. Torras, "A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 5649–5654.

[19] W. Chi, J. Liu, M. E. Abdelaziz, G. Dagnino, C. Riga, C. Bicknell, and G.-Z. Yang, "Trajectory optimization of robot-assisted endovascular catheterization with reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 3875–3881.

[20] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel, "Learning to Manipulate Deformable Objects without Demonstrations," in *Robotics: Science and Systems (RSS)*, Corvalis, Oregon, USA, Jul. 2020.

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, PMLR, 2018, pp. 1861–1870.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[23] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2012, pp. 5026–5033.

[24] E. Rohmer, S. P. N. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 1321–1326.

[25] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik, and S. Cotin, "SOFA: A Multi-Model Framework for Interactive Physical Simulation," in *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, ser. Studies in Mechanobiology, Tissue Engineering and Biomaterials, Y. Payan, Ed., vol. 11, Springer, Jun. 2012, pp. 283–321.

[26] M. Servin and C. Lacoursière, "Rigid body cable for virtual environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 783–796, 2008.

[27] M. Servin, C. Lacoursiere, and N. Melin, "Interactive simulation of elastic deformable materials," in *SIGRAD 2006. The Annual SIGRAD Conference; Special Theme: Computer Games*, Linköping University Electronic Press, 2006.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] D. Carroll, E. Hankins, E. Kose, and I. Sterling, "A survey of the differential geometry of discrete curves," *The Mathematical Intelligencer*, vol. 36, no. 4, pp. 28–35, 2014.

[30] A. Stooke and P. Abbeel, "Rlpyt: A research code base for deep reinforcement learning in pytorch," *arXiv preprint arXiv:1909.01500*, 2019.

PAPER B

ReForm: A Robot Learning Sandbox for Deformable Linear Object Manipulation

Rita Laezza*, Robert Gieselmann*, Florian T. Pokorny, Yiannis Karayiannidis

Published in *International Conference on Robotics and Automation (ICRA)*,
 DOI: 10.1109/ICRA48506.2021.9561766

* equal contribution

The layout has been revised.

Abstract

Recent advances in machine learning have triggered an enormous interest in using learning-based approaches for robot control and object manipulation. While the majority of existing algorithms are evaluated under the assumption that the involved bodies are rigid, a large number of practical applications contain deformable objects. In this work we focus on Deformable Linear Objects (DLOs) which can be used to model cables, tubes or wires. They are present in many applications such as manufacturing, agriculture and medicine. New methods in robotic manipulation research are often demonstrated in custom environments impeding reproducibility and comparisons of algorithms. We introduce ReForm, a simulation sandbox and a tool for benchmarking manipulation of DLOs. We offer six distinct environments representing important characteristics of deformable objects such as elasticity, plasticity or self-collisions and occlusions. A modular framework is used, enabling design parameters such as the end-effector degrees of freedom, reward function and type of observation. ReForm is a novel robot learning sandbox with which we intend to facilitate testing and reproducibility in manipulation research for DLOs.

1 Introduction

Countless manufacturing and every-day tasks require handling of non-rigid objects. Thus, it is important that they are properly studied, in all their variability. Yet, the dynamics of deformable objects are complex and inherently difficult to model and simulate [1]. This makes robotic manipulation of such objects using *learning-free* control a challenge. For this reason machine learning, in particular Reinforcement Learning (RL), has become increasingly popular for solving robotic manipulation tasks [2], [3]. Despite their success, RL methods are notoriously unstable and results hard to reproduce [4]. Furthermore, the complexity of the robotics system and the learning algorithm together, make it difficult to evaluate each component independently. There

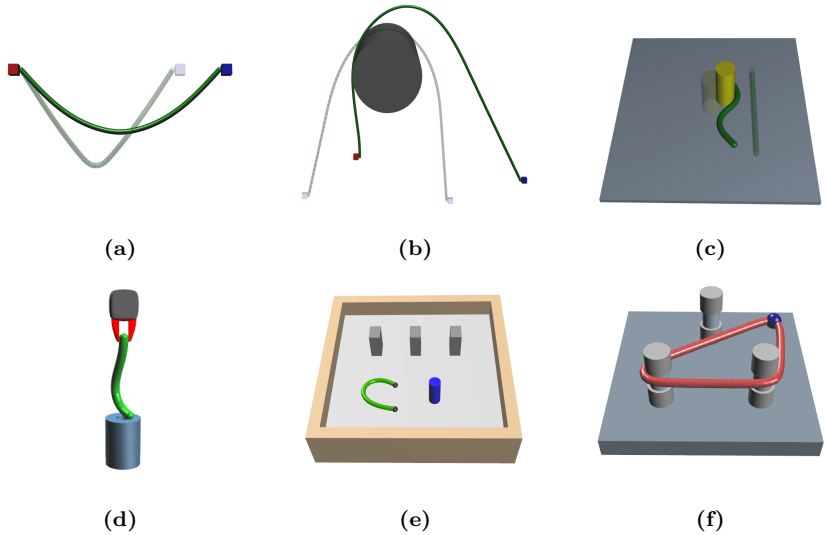


Figure 1: Overview of environments included in ReForm: (a) BendWire (b) BendWireObstacle (c) PushRope (d) PegInHole (e) CableClosing (f) RubberBand. The first three are explicit shape control tasks, while the last three are implicit shape control tasks.

have been several efforts to create robotics benchmarks in order to facilitate comparisons between methods [5]–[13].

Nevertheless, current simulation benchmarks such as RLBench [7] or Meta-World [9] focus almost exclusively on the manipulation of rigid objects. Due to the large number of industrial applications, it is necessary to also address the challenges of deformable objects. Simulation environments are particularly important for learning algorithms, since they require large amounts of data which is costly to obtain from real systems [5]. Further, real-world experiments involving deformable parts are challenging due to effects such as irreversiblity of deformations and self-occlusion.

There are few works that sufficiently address deformable objects. Based on this observation, we developed ReForm, a novel simulation sandbox and benchmarking tool for deformable object manipulation. In particular, ReForm focuses on Deformable Linear Objects (DLOs) with an emphasis on the variation of mechanical properties, such as low compression strength (e.g. rope), elastic, plastic and elastoplastic behaviors. The motivation to focus on

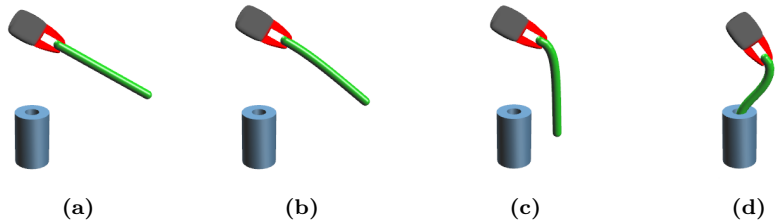


Figure 2: Illustration of a peg-in-hole task for pegs with different stiffness values: (a) rigid, (b) flexible and (c) soft. Image (d) shows an example of the object being inserted.

DLO's stems from the numerous manipulation tasks which are found across industries, such as manufacturing, surgery and agriculture [14]. Concurrent to our work, [15] recently presented SoftGym, a tool to benchmark soft object manipulation in simulation. Despite its focus on deformability, they employ a particle-based simulator which does not accurately model important material properties such as stiffness or plasticity.

An important aspect of ReForm is the freedom given with respect to simulation and problem settings including the type of observation, actuation, reward or material. For instance, one could modify the stiffness of an object in a peg-in-hole task (see Figure 2). This enables users to quickly set up new experiments with custom parameters. ReForm consists of six core DLO manipulation tasks which are integrated using the popular OpenAI Gym [10] framework. Moreover, we provide a modular interface to allow the creation of entirely new manipulation tasks. In all environments, Cartesian manipulators are employed that provide a continuous control setting. In this regard, the active Degrees of Freedom (DOFs) can be modified by the user. To demonstrate the influence of the type of observation, controlled DOFs and rewards, we evaluate standard model-free reinforcement learning for a subset of the manipulation tasks.

2 Related Work

Compared to rigid objects, shape in addition to pose estimation is required to fully capture the state of a deformable object. While in some applications deformation is treated as a disturbance, in many others achieving a particular

shape or deformation is the main objective [1]. The most commonly studied problems are textile flattening or folding, knot tying and shape control of ropes or cables [14], [16]–[19].

Matas et al, [17] studied an end-to-end reinforcement learning solution for cloth manipulation. Their method learns to predict torque signals directly from images in simulation. Using sim-to-real techniques they demonstrated that the learned policy can be transferred to the real world. Note that their method uses pre-training based on expert demonstrations. The work by [20] studies dual-arm manipulation of flexible cables. [21] applies deep reinforcement learning for a peg-in-hole task where the insertion is made of foam. In [22], the authors train an agent to insert a soft cable into a hole. While they demonstrated great results in simulation and on a real robot, their method also relies on expert demonstrations.

The variety of implementations of algorithms and experiments makes comparisons between methods and results difficult. The development of benchmarks is motivated by this need for reproducibility and comparability [4], [5]. Despite their success, deep learning methods are inherently difficult to compare due to stochasticity in the optimization process, dependence on initial parameters and random seeds. Moreover, deep reinforcement learning is known to be data inefficient and often takes millions of interactions until convergence. It also requires laborious hyperparameter tuning which hampers the ability to discern true algorithmic improvements from the amount of hand-tuning performed. [4] covered these and more issues of RL, focusing on policy gradient methods for continuous control tasks. In this regard, simulation benchmarks are particularly interesting as they provide constant conditions and allow to iterate quicker over different methods.

RLBench [7] is a recent simulation benchmark which includes 100 manipulation tasks. Unfortunately, only one of those, namely the rope-straightening task, involves a deformable object. Concept2Robot [8], which is a framework for learning manipulation concepts from human visual demonstrations and language instructions, includes the task of *folding something*. However, the simulation appears to be overly simplified[1]. Again, only one of 74 tasks addresses the challenges of deformable objects. The SURREAL [5] framework provides a smaller robotics benchmark, with six classical manipulation tasks,

[1]Task 14 of Figure 3 in [8], shows two rigid bodies attached through a hinge constraint. Code is currently not available.

such as peg-in-hole, but none including non-rigid objects. This suite was also used in the RoboTurk [6] crowd-sourcing platform for imitation learning. Meta-World [9] is yet another benchmark which focuses on multi-task learning in 50 scenarios but also excludes deformation from its simulations. SAPIEN [13] is a household simulation benchmark which allows manipulation of articulated objects, e.g doors. iGibson [12] is a similar benchmark that allows interactions with the environment. Still, it mainly focuses on indoor navigation. AssistiveGym [11] provides six benchmark tasks for human-robot interaction, one of which involves dressing a human with a deformable textile.

SoftGym [15] is a recently proposed open-source benchmark for manipulating deformable objects in simulation. It contains control tasks for cloth-like objects, ropes and a fluid. At the core, the simulator models deformable bodies using a particle-based system. In contrast, our framework focuses on DLOs and uses specialized object classes to model realistic material properties such as stiffness, elasticity and plasticity. Elastica [23] is an open-source simulation environment for soft, slender rods. It was designed to simulate soft robotic actuators that can bend, twist, shear and stretch. Unlike Elastica, our system addresses control of DLOs from a general object manipulation perspective. Thus, we provide more flexibility with respect to tasks and material properties.

Henderson et al. [4] argues that the choice of environment plays an important role when validating a new RL algorithm, because typically no single algorithm outperforms the others across all tasks. At present, deformable objects are underrepresented in robot learning benchmarks. For that reason, it is unclear how well state-of-the-art RL methods cope with the challenges inherent to the manipulation of non-rigid bodies.

3 ReForm

With the introduction of ReForm[2], we provide tools to experiment with DLOs in simulation and train agents for the manipulation of cables, ropes and wires in six different tasks. Our intention is to provide a robot learning sandbox to benchmark new methods and foster research on deformable object manipulation in simulation. We categorize different tasks as either *explicit shape control* problems, where the goal is to deform the object into a specific geo-

[2]https://sites.google.com/view/reformdlo/home

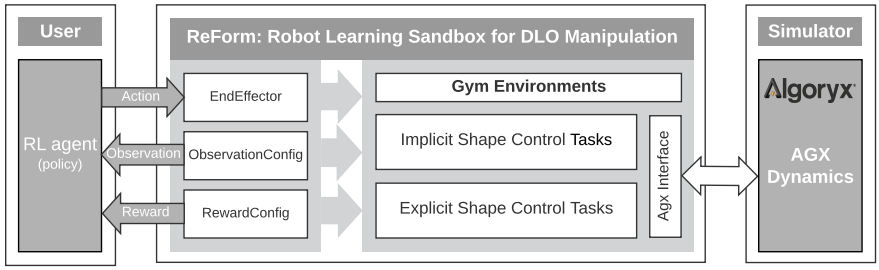


Figure 3: System overview of ReForm sandbox for robot learning. It provides a modular design to make benchmarking of new deformable object manipulation strategies easy. Note that the RL agent can be replaced by an arbitrary control policy.

metric configuration, or *implicit shape control* problems. For the latter, the exact shape of the object is not the primary objective, instead a set of high-level conditions must be fulfilled to solve the task. A few examples include hot-wire cutting [24], tube mounting [25], knot tying and threading [14], etc. Both implicit and explicit shape control problems are addressed in this work.

Similar to previous robotics benchmark software, we provide a modular implementation, as shown in Figure 3. This is done by providing an OpenAI Gym [10] interface, which is a well established toolbox for RL research. ReForm is also designed to allow configuration freedom, making it easy to modify and extend. Furthermore, the benchmark provides over ten observation types, including ground-truth and image data. Agent actions are limited to task space control, defined as linear and angular velocities of the end-effector. These can also be defined in a modular fashion, as a set of motor constraints. For environments where the gripper is holding the DLO, it is possible to control grip compliance.

In the following, we present the main components of ReForm, starting with the multiphysics simulator, followed by the description of available observation types, the control interface and finally the reward interface.

Simulation Environment Modelling and simulating DLOs or deformable objects, in general, is inherently difficult due to the complex mechanics. AGX Dynamics[3] provides unified lumped element models with implicit integration

[3]https://www.algoryx.se/agx-dynamics/

and direct, sparse factorization. Specifically, it offers a `Cable` class for which properties, such as Young modulus and Poisson's ratio, can be defined along stretch, twist and bend directions [26], [27]. Further, elastic `Cable` objects can be assigned plasticity properties by defining a yield point. Figure 4a shows the impact of this mechanical property on a shape control problem. Additionally, it offers specialized material classes which capture elastoplastic deformation in real-time. For these reasons, it provides an advantage over other physics simulators e.g. Mujoco [28], Bullet and SOFA [29]. While MuJoCo enables simulations of long object chains, it uses an explicit solver and cannot provide high stiffness. SOFA, on the other hand, offers only an iterative solver and was initially developed for interactive computational medical simulation.

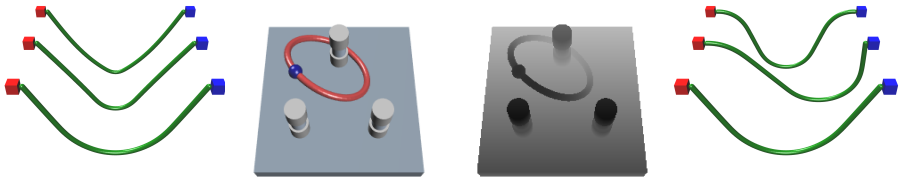
Photorealism is another important aspect to be considered, specially for end-to-end approaches which aim to learn directly from images. Currently, ReForm uses OpenSceneGraph for rendering, but AGX Dynamics also supports more realistic game engines, e.g. Unity and Unreal. It is left as future work to make use of these capabilities.

State Representation Robust state estimation is an open problem in deformable object manipulation research [30], [31]. In ReForm, both visual observations such as RGB/depth (see Figure 4) and force/torque measurements are supported. Using the `ObservationConfig` class (see Listing 6.1) it is even possible to create composite input types.

For evaluating the success of implemented methods, it is also possible to obtain ground truth position and rotation of the segments that constitute the DLO. Finally, for shape control problems, in which the location of the DLO is not necessarily important, intrinsic metrics such as discrete curvature and torsion [32] can be useful state representations.

```
observation_config = ObservationConfig(
    goals=[ObservationConfig.ObservationType.DLO_POSITIONS]
    )
observation_config.set_img_rgb()
```

Listing 6.1: Example of an observation object. In this case, the observations consist of RGB image, while the goal is defined based on Cartesian coordinates of discretized DLO.



(a) **(b)** **(c)** **(d)**

Figure 4: (a) Illustration of the effect of of plasticity on the shape of a DLO. All three shapes were generated with identical velocity trajectories, the only difference is the value of the yield point of the DLOs. This property relates to the transition from the elastic to the plastic domain. Examples of visual observations available in ReForm: (b) RGB image (c) Depth image. (d) Illustration of grip compliance changes on the shape of DLOs with identical mechanical properties, and identical trajectories.

Control Strategy In order to be platform-independent, we focus on task space control settings only. An `EndEffector` class is provided to define the controlled DOFs, along with velocity and acceleration limits (see Listing 6.2). Besides velocity control commands, there is also support for grip compliance, which allows for more complex DLO configurations, by varying the resistance to rotation along one axis, as seen in Figure 4d. Our system allows to include velocity and acceleration limits which is important because RL policies often produce jitter. The work in [33] highlights the importance of trajectory speed for dynamic manipulation strategies applied to non-rigid objects. Adding these constraints acts as a filter which prevents fast velocity changes and excessive forces. Note that this has a similar effect to using temporally correlated exploration noise, such as in [34], [35]. As a further benefit, this makes our control interface task agnostic, since agent actions always lie between $[-1, 1]$, but are automatically rescaled to a range appropriate for the task.

Reward Definition Though ReForm can be used to evaluate methods other than RL, we follow the formalism from OpenAI Gym [10], and thus include a reward computation step. While standard implementations of Gym environments have a fixed reward, effectively making it part of the environment, our library provides an abstract `RewardConfig` class, which enables the reward definition to be part of the solution strategy. We find this to be particularly

important since for many deformable object manipulation tasks, the definition of the reward function is not trivial [36]. While sparse rewards are generally applicable, a learning signal is only provided once the goal has been reached. It was shown by [3] that for some multi-goal scenarios this type of reward makes learning nearly impossible. Engineering dense rewards by hand is often a laborious task that might require significant domain expertise. In Section 5, we evaluate the impact of the reward definition for a subset of our environments.

```
gripper_right = EndEffector(
    name='gripper_right',
    controllable=True,
    observable=True,
    max_velocity=0.014,  # m/s
    max_acceleration=0.010  # m/(s*s)
    )
gripper_right.add_constraint(
    name='prismatic_joint_right',
    end_effector_dof=EndEffectorConstraint.Dof.X_TRANSLATION,
    compute_forces_enabled=True,
    velocity_control=True,
    compliance_control=False
    )
gripper_right.add_constraint(
    name='hinge_joint_right',
    end_effector_dof=EndEffectorConstraint.Dof.Y_COMPLIANCE,
    compute_forces_enabled=False,
    velocity_control=False,
    compliance_control=False
    )
```

Listing 6.2: Example of an end-effector object. Both velocity and acceleration limits are set at initialization (SI units). Note that since the simulation consists of a small DLO, it is important to keep velocities low and prevent large accelerations. If the end-effector is set to be `observable`, this enables force-torque measurements that can be used for the observations. Besides `TRANSLATION` constraints, there are `COMPLIANCE` constraints to control the compliance of the grip.

4 Manipulation Tasks

In this section, we describe the tasks that are currently implemented in ReForm. An overview of all environments is given in Figure 1. In general, we distinguish between explicit and implicit shape control. The former describes problems in which the goal is represented by a particular shape configuration of the deformable object. For some problems however it might be more convenient to define the primary task by means of a high-level description. Examples include assembly problems in which the final shape of the object is secondary. These kinds of tasks are captured by implicit shape control. In the following, we briefly describe the features and challenges of each task in ReForm:

BendWire: a thin wire is attached to two grippers using a hinge constraint. The goal is to deform the wire into a desired shape, hence it can be seen as explicit shape control. The material of the wire is stiff and exhibits elastoplastic properties. The fact that plastic deformations are usually hard or even impossible to reverse presents a key challenge. Even small deformations can significantly change the set of reachable wire states. DLOs of this kind include most metal wires, which are found throughout manufacturing as well as in medical applications, such as in dental braces. A similar problem was tackled by [20].

BendWireObstacle: this environment is similar to BendWire but includes a cylindrical obstacle in the workspace. While not necessary, the obstacle can be leveraged to facilitate the deformation task. With this setting, we open the possibility for extrinsic manipulation. The work by Zhu et al. [37] covered such a scenario for cable routing. This task is also an explicit shape control problem.

PushRope: a soft rope is located on a planar surface. A controllable pusher is used to bring the rope into a certain goal configuration. The rope exhibits little stiffness and deforms immediately after contact. Further, the friction between the rope and surface adds another interesting feature to the manipulation task. PushRope is a common explicit shape control benchmark which has been studied in [7], [36].

PegInHole: a soft peg is on one end rigidly attached to a gripper. The task is to insert the peg into a hollow cylindrical object. Compared to the previous environments, the goal is not represented by just a single configuration of the object. Instead it is a set of configurations which satisfy the condition

of being inserted. Clearly, this is an implicit shape control problem. Due to its simplicity and significance for assembly tasks, the peg-in-hole scenario has become a widely-studied problem in robotics research. We offer a simulated version that uses a deformable peg with a rigid hole. Note that [21] presented the converse problem in which a rigid peg is inserted into a deformable hole.

CableClosing: a cable is controlled on both ends by planar actuators. The task is successfully completed once the cable fully encloses a goal obstacle. There are several static obstacles in the scene. In order to reach the goal, the object has to be circumnavigated around the obstacles. Again, this describes an implicit shape control problem. This task is quite unique, although it can be compared with the work by [14], where they used objects in the environment to tie different types of knots.

RubberBand: a purely elastic circular rubber band is connected to a gripper by a ball joint. The task is to find a policy which wraps the rubber band around three poles, making it an implicit shape control task. This environment incorporates complicated contact mechanics between the deformable object and the poles. Further, it describes a wrapping task which bridges the gap to industrial applications, such as [38]. Though a rubber band is technically not a linear object, it can be seen as a DLO connected with itself.

5 Benchmarking Experiments

In order to establish a first baseline, we performed an evaluation of all six environments using model-free reinforcement learning. For each, we trained and evaluated DDPG agents [35] as implemented in [39]. We used two hidden layers for both policy and value networks, with $[300, 400]$ neurons each. Due to the different nature of explicit and implicit shape control tasks, we evaluate them separately. For each task, the DDPG algorithm was applied using $1{\times}10^6$ steps of environment interaction.

Explicit Shape Control For these environments, the goal is to control the end-effector(s) in order to deform a DLO into a desired shape. Each task has different challenges including plasticity, interaction with a rigid body and low compression strength. In order to evaluate the impact of design choices such as observation type, action space and reward definition, we run experiments varying these parameters. Figure 6 (top row) shows the effect of three differ-

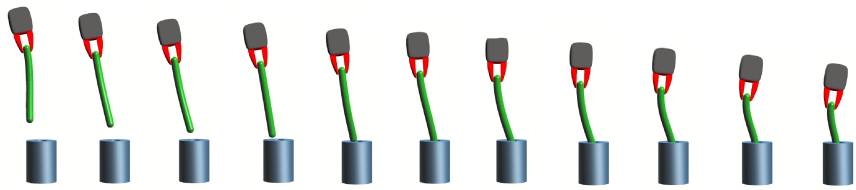


Figure 5: Example of successful trajectory for the PegInHole task learned by the DDPG agent.

ent observation types: ground truth positions, rotations and curvature. No observation type seems to work best across all tasks. It is curious that for the wire environments, there is a clear winner but they do not coincide, with curvature working better for BendWire and DLO segment positions for BendWireObstacle. Results for the PushRope task were quite close and suffered from high variance, making it difficult to draw conclusions.

Figure 6 (top row) further shows the impact on augmenting the action space. For the wire tasks, this was done by adding control of grip compliance, which can lead to very different shapes (see Figure 4d). For the PushRope task, a third DOF was added, effectively allowing the pusher to move over the rope, thus making exploration much harder. As expected, increasing the action space led to slower learning for this task. However, for the wire tasks it is not easy to draw a clear conclusion. For the BendWire environment it even seems to help performance. This may be because the goal shape consists of a sharp angle (see Figure 1a), making the extra DOF useful for creating sharper deformations.

For both the observation and action experiments, the same dense reward was used, computed as negative the L2 distance between the current curvature and the desired curvature. Further, a small positive reward is given when the agent is close to the target shape. However, to analyse the impact of our reward definition, we also implemented a sparse reward which provides a negative signal until the goal is reached, after which a positive signal is given. For each environment, the final curvature error for both sparse and dense rewards were compared. Results are nearly identical across all tasks, with 0.13 $\pm$0.07 (dense) and 0.13$\pm$0.20 (sparse) for BendWire; 0.23 $\pm$0.08 (dense) and 0.24$\pm$0.10 (sparse) for BendWireObstacle; and 0.14 $\pm$0.03 (dense) and 0.15$\pm$0.01 (sparse) for PushRope.

Implicit Shape Control For all of the implicit shape control tasks, the state observation consists of the position of the DLO segments and the grippers. We further evaluated a second setting which also takes into account the velocity of the involved bodies. Note that the definition of a continuous reward function is not obvious for these tasks due the presence of obstacles in the workspace. Instead, we provided intermediate feedback for reaching task-specific subgoals. In the PegInHole task, the agent receives a positive reward of +1 for each cable segment being inserted. In the CableClosing environment, the agent receives a positive reward of +1 for partially enclosing the pole and an additional reward of +1 for fully enclosing it. In the RubberBand task, the agent receives a positive reward of +1 for each pole being enclosed by the rubber band and an additional reward of +5 at task completion. Reversing the progress is penalized accordingly using negative rewards.

The corresponding training returns are shown in Figure 6 (bottom row). It can be seen that the policies converge in the PegInHole and CableClosing environments. Figure 5 shows a trajectory generated by the learned policy for the PegInHole insertion task. Yet, we could not achieve statisfying results in the RubberBand environment. The DDPG standard implementation applies Gaussian noise to the predicted actions in order to explore the state-action space. We believe that this strategy is insufficient to reach high-reward regions in the RubberBand case and suggest the use of more sophisticated exploration methods. Furthermore, the wobbly dynamics present an additional challenge of this control task. Note that we did not observe significant improvements when including velocities into the observations. The increased dimensionality of the input might have overshadowed the benefits of using this additional type of information. After all, the issue of dealing with high-dimensional state-spaces is inherent to the manipulation of deformable objects.

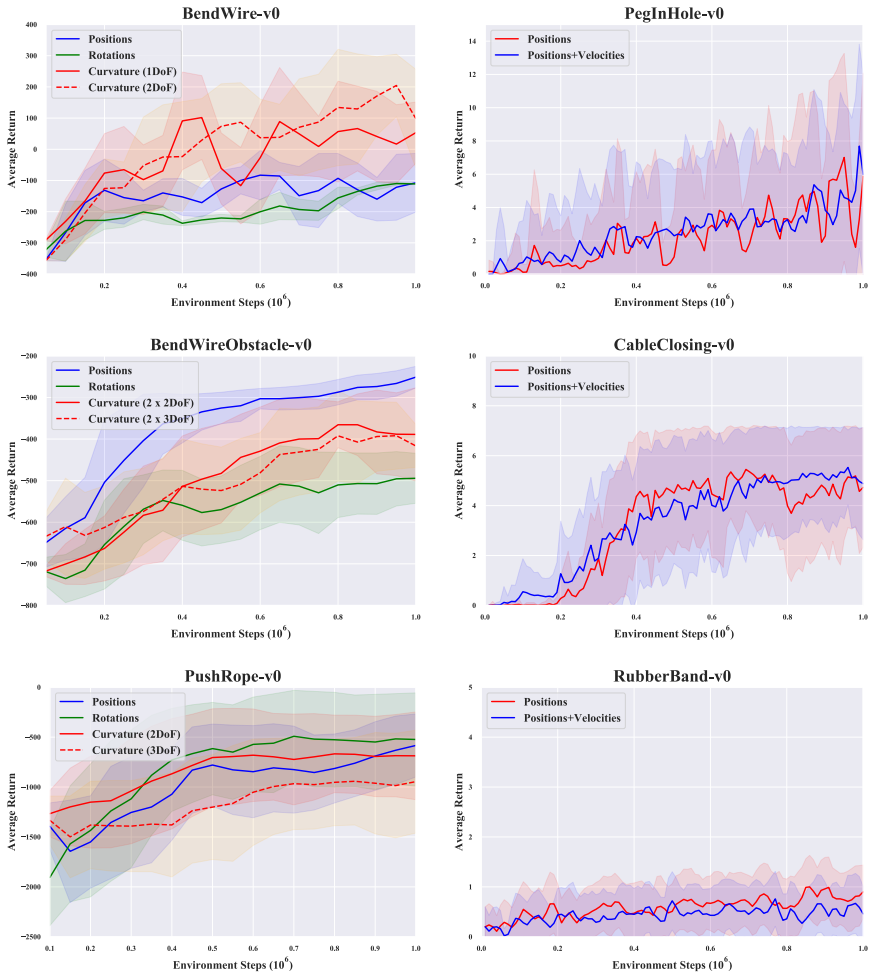


Figure 6: Left column shows average return results for the three explicit shape control environments: BendWire, BendWireObstacle and PushRope. Results are averaged over 5 runs, shaded regions represent mean standard deviation during training. Results indicate the effect of different observation types and larger action spaces on learning. Right column shows average return results for the three implicit shape control environments: PegInHole, CableClosing and RubberBand. Results are presented for observations with just the DLO positions and with both positions and velocities. A dense reward definition was used; only one trial is shown.

6 Conclusion

In this work we introduced ReForm, a new sandbox for robot learning including six DLO manipulation environments. The development of our framework is motivated by the need for more benchmarks which capture the complexities of deformable objects. We evaluated the performance of DDPG agents for all environments using different types of observations, action spaces and reward functions. The results present a first baseline for future investigations.

During the Robotics Debates[4] workshop at ICRA 2020, the issue of whether "Robotics research is over-reliant on benchmark datasets and simulation" was debated. Though there has been no final consensus about the need for simulation benchmarks, several arguments in favor of it are i. the ability to replicate results; ii. the idea that "simulators democratize robotics", since researchers with limited resources can still test their methods; iii. in related fields such as computer vision and natural language understanding, benchmark datasets have long been an important part of these communities.

In future work, we seek to extend ReForm by adding new manipulation tasks. Moreover, photorealistic rendering and procedural generation abilities will enable our framework for domain-adaptation and sim-to-real research.

References

[1] J. Sanchez, J.-A. Corrales, B.-C. Bouzgarrou, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 688–716, 2018.

[2] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[3] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *31st International Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 5055–5065.

[4] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] https://roboticsdebates.org/

[5] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, "Surreal: Open-source reinforcement learning framework and robot manipulation benchmark," in *Conference on Robot Learning (CoRL)*, 2018, pp. 767–782.

[6] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, S. Savarese, and L. Fei-Fei, "Roboturk: A crowdsourcing platform for robotic skill learning through imitation," in *Conference on Robot Learning (CoRL)*, 2018.

[7] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3019–3026, 2020.

[8] L. Shao, T. Migimatsu, Q. Zhang, K. Yang, and J. Bohg, "Concept2robot: Learning manipulation concepts from instructions and human demonstrations," *Robotics: Science and Systems (RSS)*, 2020.

[9] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2020, pp. 1094–1100.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[11] Z. Erickson, V. Gangaram, A. Kapusta, C. K. Liu, and C. C. Kemp, "Assistive gym: A physics simulation framework for assistive robotics," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020.

[12] F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. E. Tchapmi, A. Toshev, R. Martın-Martın, and S. Savarese, "Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 713–720, 2020.

[13] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, *et al.*, "Sapien: A simulated part-based interactive environment," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 097–11 107.

[14] M. Saha and P. Isto, "Motion planning for robotic manipulation of deformable linear objects," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2006, pp. 2478–2484.

[15] X. Lin, Y. Wang, J. Olkin, and D. Held, "Softgym: Benchmarking deep reinforcement learning for deformable object manipulation," in *Conference on Robot Learning (CoRL)*, 2020.

[16] D. Berenson, "Manipulation of deformable objects without modeling and simulating deformation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2013, pp. 4525–4532.

[17] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning (CoRL)*, 2018, pp. 734–743.

[18] M. Ruan, D. Mc Conachie, and D. Berenson, "Accounting for directional rigidity and constraints in control for manipulation of deformable objects without physical simulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 512–519.

[19] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel, "Learning to Manipulate Deformable Objects without Demonstrations," in *Robotics: Science and Systems (RSS)*, Corvalis, Oregon, USA, Jul. 2020.

[20] J. Zhu, B. Navarro, P. Fraisse, A. Crosnier, and A. Cherubini, "Dual-arm robotic manipulation of flexible cables," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 479–484.

[21] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, and A. M. Agogino, "Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 2062–2069.

[22] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[23] N. Naughton, J. Sun, A. Tekinalp, T. Parthasarathy, G. Chowdhary, and M. Gazzola, "Elastica: A compliant mechanics environment for soft robotic control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3389–3396, 2021.

[24] S. Duenser, R. Poranne, B. Thomaszewski, and S. Coros, "Robocut: Hot-wire cutting with robot-controlled flexible rods," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 98–1, 2020.

[25] M. Rambow, T. Schauß, M. Buss, and S. Hirche, "Autonomous manipulation of deformable objects based on teleoperated demonstrations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2012, pp. 2809–2814.

[26] M. Servin and C. Lacoursière, "Rigid body cable for virtual environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 783–796, 2008.

[27] H. Lang, J. Linn, and M. Arnold, "Multi-body dynamics simulation of geometrically exact Cosserat rods," English, *Multibody System Dynamics*, vol. 25, no. 3, pp. 285–312, 2011, ISSN: 1384-5640.

[28] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2012, pp. 5026–5033.

[29] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik, *et al.*, "Sofa: A multi-model framework for interactive physical simulation," in *Soft tissue biomechanical modeling for computer assisted surgery*, Springer, 2012, pp. 283–321.

[30] M. Yan, Y. Zhu, N. Jin, and J. Bohg, "Self-supervised learning of state estimation for manipulating deformable linear objects," *IEEE Robotics and Automation Letters*, 2020.

[31] T. Tang and M. Tomizuka, "Track deformable objects from point clouds with structure preserved registration," *The International Journal of Robotics Research*, p. 0 278 364 919 841 431, 2018.

[32] D. Carroll, E. Hankins, E. Kose, and I. Sterling, "A survey of the differential geometry of discrete curves," *The Mathematical Intelligencer*, vol. 36, no. 4, pp. 28–35, 2014.

[33] R. Jangir, G. Alenyà, and C. Torras, "Dynamic cloth manipulation with deep reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 4630–4636.

[34] P. Wawrzynski, "Control policy with autocorrelated noise in reinforcement learning for robotics," *International Journal of Machine Learning and Computing*, vol. 5, no. 2, p. 91, 2015.

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

[36] X. Lin, H. S. Baweja, and D. Held, "Reinforcement learning without ground-truth state," *arXiv preprint arXiv:1905.07866*, 2019.

[37] J. Zhu, B. Navarro, R. Passama, P. Fraisse, A. Crosnier, and A. Cherubini, "Robotic manipulation planning for shaping deformable linear objects withenvironmental contacts," *IEEE Robotics and Automation Letters*, vol. 5, no. 1, pp. 16–23, 2019.

[38] M. Murase, K. Yamazaki, and T. Matsubara, "Kullback leibler control approach to rubber band manipulation," in *2017 IEEE/SICE International Symposium on System Integration (SII)*, IEEE, 2017, pp. 680–685.

[39] A. Stooke and P. Abbeel, "Rlpyt: A research code base for deep reinforcement learning in pytorch," *arXiv preprint arXiv:1909.01500*, 2019.