Title of Dissertation: **LEARNING-BASED MOTION PLANNING FOR HIGH-DOF ROBOT SYSTEMS**

Biao Jia
Doctor of Philosophy, 2023

Dissertation Directed by: **Professor Dinesh Manocha**
Department of Computer Science

A high-degree-of-freedom (DoF) robot system refers to a type of robotic system that possesses many independently controllable mechanical degrees of freedom. This includes high-DoF robots or objects being manipulated, such as flexible robotic arms and flexible objects. Degrees of freedom in robotics represent the different ways a robot can move or manipulate its parts. High-DoF robot systems have a significant number of these independent motions, allowing them to exhibit complex and versatile movements and behaviors. These systems are employed in various applications, including manufacturing and healthcare, where precise and flexible control is essential. The main difficulty associated with high-DoF robot systems is the complexity arising from their numerous degrees of freedom. Calculating the optimal trajectories or control inputs for high-DoF systems can be computationally intensive. The sheer number of variables and the need for real-time responsiveness pose significant challenges in terms of computation and control. In some cases, high-DoF robot systems interact with deformable objects such as fabrics and foam. Modeling and controlling these objects add additional layers of complexity due to their dynamic and unpredictable behavior.

To address these challenges, we delve into several key areas: Object Deformation Modeling, Controller Parameterization, System Identification, Control Policy Learning, and Sim-to-Real Transfer. We begin by using cloth manipulation as an example to illustrate how to model high-DoF objects and design mapping relationships. By leveraging computer vision and visual feedback-based controllers, we enhance the ability to model and control objects with substantial shape variations, which is particularly valuable in applications involving deformable materials. Next, we shift our focus to Controller Parameterization, aiming to define control parameters for high-DoF objects. We employ a random forest-based controller along with imitation learning, resulting in more robust and efficient controllers, which are essential for high-DoF robot systems. This method can be used for human-robot collaboration involving flexible objects and enables imitation learning to converge in as few as 4-5 iterations. Furthermore, we explore how to reduce the dimensionality of both high-degree-of-freedom (high-DoF) robot systems and objects simultaneously. Our system allows for the more effective use of computationally intensive methods like reinforcement learning (RL) or trajectory optimization. Therefore, we design a system identification method to reduce the need for repeated rendering or experiments, significantly improving the efficiency of RL. This enables some algorithms with exponential computational complexity to be solved in linear time. In this part of the work, we adopt a real setup where humans and robots collaborate in real-time to manipulate flexible objects.

In the second part of our research, we focus on the task of natural media painting. We utilize reinforcement learning techniques. Painting itself can be considered a high-DoF robot system, as it entails a multitude of context-dependent actions to complete the task. Our objective is to replicate a reference image using brush strokes, with the goal encoded through observations. We will focus on how to address the sparse reward distribution with a large continuous action

space. Additionally, we investigate the practicality of transferring learned policies from simulated environments to real-world scenarios, with a specific focus on tasks like painting. This research bridges the gap between simulation and practical application, ensuring that the knowledge gained from our work can be effectively utilized in real-world settings. Ultimately, we will demonstrate the use of RL-learned painting strategies in both virtual and real robot environments.

LEARNING-BASED MOTION PLANNING FOR HIGH-DOF ROBOT SYSTEMS

by

Biao Jia

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:
Professor Dinesh Manocha, Chair/Advisor
Professor Mumu Xu, Dean's Representative
Professor Yiannis Aloimonos
Professor Huaishu Peng
Professor Furong Huang

Acknowledgments

Firstly, I would like to express my deepest gratitude to my advisor, Prof. Dinesh Manocha, for his extensive support of my research. His meticulousness has greatly shaped my academic discipline. Every published work of mine has benefited from his careful revisions. Working and studying in the GAMMA Lab has given me insights into various fields like robotics, computer vision, graphics, and natural language processing, which have significantly broadened my horizon.

I am grateful to my mentors at Adobe Research, especially Dr. Chen Fang and Dr. Jonathan Brandt. Dr. Fang opened the door to reinforcement learning for me, allowing me to understand what research in the industry looks like. Dr. Brandt has been an exceptional role model, demonstrating what a distinguished researcher should be like, always passionate about research problems and continuing to code.

Special thanks to my colleagues at GAMMA Lab, particularly Dr. Zherong Pan and Dr. Jae-Sung Park. They have been excellent collaborators. Even though our research directions might vary, they have provided me with a lot of research inspiration and guidance. They are both outstanding programmers and researchers, from whom I have learned a great deal. Their presence has made my time at Chapel Hill unforgettable.

Thanks to Prof. Jia Pan from Hong Kong University, who led me into the field of deformable object manipulation. This has inspired me to apply robotic motion planning in real

industrial and daily life scenarios, rather than just testing in simulated environments.

I am thankful to Jianhua Zhong, my teacher who introduced me to PASCAL programming, bringing me into the world of computer programming. My gratitude also goes to Prof. Haizhou Ai, who guided my graduation project and opened the door to scientific research for me.

Lastly, I want to express my deepest gratitude to my family, who have been the guiding light on my journey. I am especially thankful to my father, Zhidong Jia, and my mother, Ruixiang Geng. Without them, this thesis would not have been possible. Their teachings, both in words and deeds, have been the best education for me. I also wish to express my longing for my grandparents in heaven, Shiyin Jia, Guihua Jia, and Runmou Geng. It's regrettable that I couldn't show them the developments in artificial intelligence technology today. I hope that my work can make them proud.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

With the advancement of industrial technology, robots play significant roles in structured production lines [5]. However, in our daily lives, environments are often far from structured, especially in non-industrial settings. The shift towards integrating robots into unstructured environments has opened exciting opportunities for research and innovation. This shift has been driven by the need for robots to adapt to dynamic and unpredictable situations, such as elder-care [6] and interactive assembly [7, 8], where some objects themselves exhibit non-structured characteristics and are challenging to model, such as flexible or deformable objects.

Deformable objects are ubiquitous in our daily lives, including our own biological organs and skin tissues, as well as common household items like clothing, fabrics, and sponges. In these sce- narios, the number of degrees of freedom (DoFs) can be very large, and the repeated evaluation of forward dynamics becomes a significant bottleneck.

In other scenarios, we have developed increasingly complex robots. These flexible robotic systems are designed to provide safer and more convenient solutions to a wide range of challenges in both daily life and industrial production. Soft robots equipped with deformable joints introduce high-dimensional configuration spaces. For instance, they excel in applications such as medical procedures [9], offering human assistance, and handling delicate object manipulation [10].

Deformable or soft robots have a wide range of applications across various domains. They

are used in medical procedures, such as minimally invasive surgeries, where their compliance allows for precise and safe interactions with delicate tissues and organs. These robots also excel in search and rescue missions, navigating through complex and cluttered environments, including collapsed buildings, to locate and assist survivors. Additionally, their gentle and adaptive nature means they are employed to assist individuals with mobility impairments, the elderly, and those in need of rehabilitation exercises. Soft robots are proficient at handling and manipulating objects with varying shapes and sizes, making them valuable in logistics and warehouse automation, where items may not be rigid and structured.

Unlike rigid bodies, these objects and robots cannot be fully described by a finite-length vector representing their position and pose. Instead, they require more complex representations and models to capture their deformable nature and adaptability to various configurations and forces. The study of robotics in the context of high-DoF objects presents unique challenges and opportunities for advancements in areas like soft robotics, manipulation, and human-robot interaction.

While robot motion planning has been extensively studied for decades, primarily focusing on rigid objects and robots, high-DoF robot systems present unique challenges due to their extremely high-dimensional configuration spaces. Addressing these challenges necessitates developing manipulation algorithms capable of handling such complexity while maintaining precision. This is crucial for applications ranging from cloth folding to robot-assisted dressing, household chores, robot-assisted surgery, and even some artistic creations.

1.1 Research Questions

As mentioned above, high-DoF robot systems mark a significant departure from their traditional counterparts. The defining characteristic of high-DoF systems is their abundance of independently controllable mechanical degrees of freedom, offering unparalleled complexity and versatility. Unlike conventional robot control systems with limited degrees of freedom, high-DoF systems possess a multitude of degrees of freedom.

This increased complexity and versatility come with a unique set of challenges, especially in the domain of motion planning. Traditional motion planning techniques, effective in scenarios with fewer degrees of freedom, may struggle to scale to high-DoF systems due to the sheer volume of variables involved. As a result, we face computational and logistical challenges when planning and executing motions efficiently in these systems.

Furthermore, high-DoF robot systems often interact with deformable objects like fabrics, foams, or biological tissues, introducing an additional layer of complexity. Deformable materials exhibit dynamic and unpredictable behaviors, making their modeling and control a challenging endeavor. Traditional rigid-body motion planning approaches may fall short when dealing with deformable objects, necessitating the development of specialized techniques.

In addition to these challenges, high-level tasks require us to tackle the intricacies of policy learning, transforming controller parameterization into more complex policy development. Our research is dedicated to addressing these fundamental questions and finding innovative solutions to the multifaceted challenges posed by high-DoF robot systems.

As a result, we focus on several fundamental questions to address these challenges:

1. **Object Deformation Modeling:** How can we accurately model objects while considering their substantial deformations? This process involves managing perceptual data, data processing, feature extraction, dimensionality reduction, and associated control variables.

2. **Controller Parameterization:** How can we design controllers that effectively map an object's state to precise robotic actions for specific tasks? Often, this requires laborious manual parameter tuning to generalize across diverse tasks. How can we streamline controller parameterization to handle the complexities of high-DoF robot systems?

3. **Control Policy Learning:** How can we efficiently learn a control policy tailored to high-DoF robot systems? Many traditional methods rely on defining simple actions, but for these systems, task definition itself is complex. How can we tackle the challenge of defining and collecting data for such intricate tasks?

1.2 State-of-the-Art High-DoF Robotics

In recent years, there has been substantial progress in the field of high-degree-of-freedom (DoF) robotic systems. Traditional methods, such as those using knot theory or energy theory for linear deformable objects, have paved the way for more advanced techniques tailored to the capabilities of high-DoF robots.

Early work by Saha and Moll [11, 12] concentrated on linear deformable objects like ropes. However, these methods demanded precise knowledge of the object's geometric and deformation parameters, rendering them impractical for real-world applications, especially when dealing with high-DoF robotic systems.

To address these challenges, recent research has focused on sensor-based guidance and real-

time control strategies. For instance, Matsuno et al. [13] employed image-based approaches to estimate knot configurations, while Miller et al. [14] used vision to determine cloth configurations and utilized gravity for folding tasks [15]. These approaches represent a shift towards more practical and real-time solutions.

In the context of high-DoF robots, trajectory optimization and model-based controllers have been explored for deformable object manipulation (DOM) applications [16–18]. However, their accuracy often comes at the cost of real-time performance. Despite efforts to develop real-time trajectory optimization approaches for robots with lower DoF, extending these methods to high-DoF robotic systems remains a significant challenge due to the computational complexity associated with simulating deformable objects.

Currently, achieving real-time performance in high-DoF robotic systems is primarily realized through learning-based controllers [19–21]. These controllers employ supervised learning to train models that can operate in real-time. While effective, they may face challenges in handling unseen data and maintaining robustness, as highlighted in Ross et al. [22]. To address these issues, researchers have begun to explore the integration of imitation learning techniques.

Reinforcement learning (RL) has emerged as a promising approach for controlling high-DoF robotic systems. RL has achieved notable success in various domains, including playing Atari games [23], mastering the game of Go [24], and implementing robot control [25]. The focus of these efforts has been on enhancing the efficiency of learning algorithms, particularly in terms of time and data requirements.

In the context of high-DoF robotic systems, deep reinforcement learning techniques such as Deep Q-Learning [23] and proximal policy optimization (PPO) [3] have shown promise for handling continuous action spaces. Additionally, hindsight experience replay [26] has enabled

off-policy reinforcement learning to sample from sparse and binary rewards efficiently, further improving the performance of high-DoF robotic systems.

1.3 Research Contributions

In this thesis, we address several key challenges presented by high-DoF robot systems and offer solutions to these challenges. We simplify and resolve these issues using machine learning approaches.

With the rapid development of artificial intelligence and machine learning technologies, the expressive and inferential capabilities of massive artificial neural network models have undergone a qualitative transformation. Notably, in the fields of natural language processing (NLP) and computer vision, transformer-based models have made remarkable strides. These advancements, such as stable diffusion [27] and OpenAI's GPT models [28], demonstrate the remarkable progress in AI systems' capabilities. They can now create art, generate lengthy and complex paragraphs, and achieve feats once considered beyond reach.

By leveraging machine learning techniques, our goal is to tackle complex control challenges in robotics, where traditional rule-based approaches fall short. This thesis delves into the design and implementation of machine learning algorithms capable of handling unstructured environments, high-DoF robots, and flexible objects. Through this research, we aspire to contribute to the advancement of robotics and empower robots with the adaptability and intelligence required to excel in real-world, non-structured scenarios.

1.3.1 High-DoF Object Modeling

For a high-degree-of-freedom (high-DoF) robotic system, especially when dealing with objects that possess a high degree of freedom themselves, such as cloth, foam, or strings, the most direct approach is to model or extract features from the object itself. Our research is dedicated to modeling objects with significant deformability precisely, with a specific emphasis on cloth as our primary subject of study. In pursuit of this objective, we introduce a novel feature extraction technique known as the "histogram of oriented wrinkles." This method autonomously captures variations in the shapes of deformable objects. Concurrently, we propose a feedback controller that relies on a visual feedback dictionary. This controller learns the intricate behaviors of deformable objects through human demonstrations (refer to Chapter 2).

Dealing with highly deformable materials, like textiles, presents significant challenges for robotic manipulation systems. To address these challenges, we present an innovative approach based on a visual feedback dictionary for autonomous robotic manipulation of deformable objects, steering them toward desired configurations. Our method is grounded in visual servoing, and we employ an efficient technique to extract crucial features from the RGB sensor stream, creating a histogram of deformable model features. These histograms serve as high-level representations of the deformable material's state. Subsequently, we gather manipulation data and utilize a visual feedback dictionary to map velocity in the high-dimensional feature space to the robotic end-effector's velocity for manipulation. Our approach undergoes rigorous evaluation through a series of complex manipulation tasks, including human-robot collaboration, encompassing diverse cloth materials, each with varying characteristics.

This comprehensive approach addresses the intricate challenges posed by deformable ob-

jects in robotic manipulation, significantly advancing our understanding of their behavior and offering practical applications.

1.3.2 High-DoF Controller Parameterization

To further address controller parameterization for high-DoF robot systems, we integrate feature extraction and controller design using a random forest trained through a two-stage learning process. This approach combines feature extraction with controller parameterization, reducing the need for manual parameter determination. The construction of the random forest and controller optimization are integrated into an imitation learning framework, enhancing the robustness of human-robot co-manipulation tasks (refer to Chapter 3).

In this chapter, we introduce an approach to controller parameterization designed for the manipulation of high-DoF deformable objects, specifically focusing on examples like cloth. Our method harnesses a controller based on random forests, which effectively translates the observed visual features of the deformable object into optimal control actions for the robotic manipulator. The underlying topological structure of this random forest is automatically determined from the training dataset, which comprises visual features and corresponding control signals. This dataset is dynamically constructed through an online imitation learning algorithm. We have extensively assessed the efficacy of our approach across various cloth manipulation tasks, including flattening, folding, and twisting. Across all these tasks, we have consistently observed convergent behavior in the random-forest-based controller. When it reaches convergence, our controller demonstrates exceptional robustness in handling observation noise, surpassing alternative techniques such as convolutional neural networks and nearest neighbor searches.

1.3.3 High-DoF Robot System Identification

Subsequently, we identified a pivotal challenge: if we could reduce the dimensionality of both high-DoF robot systems and objects simultaneously, our system could employ computationally intensive methods like reinforcement learning (RL) more effectively. Thus, we propose a system identification method here to tackle the modeling difficulties encountered in high-DoF robot systems. This approach proves effective not only for addressing object deformations but also for mitigating issues such as slow rendering in simulation systems and protracted training times in reinforcement learning. We've observed that DoFs often display redundancy, which allows for the compression of their configuration space (as elaborated in Chapter 4).

Within this chapter, we introduce an efficient algorithm for motion planning and control of robot systems characterized by a high number of DoFs. These systems encompass high-DoF soft robots and articulated robots interacting with deformable environments. Our innovation lies in a novel technique designed to expedite the evaluation of the forward dynamics function. This is accomplished by storing the outcomes of computationally expensive computations in a hierarchical adaptive grid. Furthermore, we leverage the underactuated properties of these robot systems, constructing the grid in a lower-dimensional space. Our approach approximates the forward dynamics function while ensuring error bounds, rendering it suitable for optimization-based motion planning and feedback control using reinforcement learning. We showcase its performance on two high-DoF robot systems: a line-actuated elastic robot arm and an underwater swimming robot in a water environment. Compared to previous techniques reliant on exact dynamics evaluation, we observe performance improvements on the order of one to two magnitudes.

1.3.4 High-DoF System Policy Learning with Reinforcement Learning

In addition to the discussion on imitation learning presented in Chapter 4, we explore more efficient learning strategies for complex control policies in high-degree-of-freedom (high-DoF) robot systems. Specifically, we delve into reinforcement learning approaches, which require state space compression and the integration of conditions, self-imitation, and other techniques to enhance the learning of control policies (as detailed in Chapter 5).

Within this chapter, we introduce a novel natural media painting algorithm based on reinforcement learning. Painting itself can be considered a high-DoF robot system, as it entails a multitude of context-dependent actions to complete the task. Therefore, data collection for such a task is notably challenging. Hence, employing an RL-based approach is highly suitable. Our objective is to replicate a reference image using brush strokes, with the goal encoded through observations. We acknowledge that the reward distribution in the action space is sparse, and training an RL algorithm from scratch can be formidable.

To address these challenges, we present an approach that combines self-supervised learning and reinforcement learning to effectively transform negative samples into positive ones and alter the reward distribution. Through this method, we demonstrate the advantages of our painting agent in faithfully reproducing reference images using brush strokes.

1.3.5 Transferring Learned Policies to Real Environments

Once we have an agent trained through reinforcement learning in a virtual environment, another significant challenge we tackle is the transfer of learned policies from simulation to real-world environments. We conducted experiments involving painting tasks to delve deeper into this

transition, as elaborated in Chapter 7.

Within this chapter, we introduce an innovative approach that serves as a bridge between digital and robotic sketching, harnessing the power of behavior cloning and reinforcement learning techniques. This research introduces a method designed to seamlessly connect simulated and real-world robotic sketching by integrating behavior cloning and reinforcement learning techniques. Our approach trains painting policies that operate effectively in both virtual environments and real-world robotic sketching systems.

We have implemented a robotic sketching system featuring an UltraArm robot equipped with a RealSense D415 camera, closely emulating the MyPaint virtual environment. This system possesses the ability to perceive its surroundings and adapt its painting policies to natural painting media. Our results underscore the effectiveness of our agent in acquiring policies for high-dimensional continuous action spaces, enabling the smooth transfer of brush manipulation techniques from simulation to practical robotic sketching. Furthermore, we showcase the capability of our robotic sketching system to generate intricate images and strokes using various configurations.

1.4 Overview

In the following chapters, we'll explore specific research problems, review relevant literature, propose new solutions, share important results, and summarize our findings. As we conclude our work, we'll provide a comprehensive overview, highlighting our main accomplishments, addressing research limitations, and suggesting directions for future studies. Here are the key contributions of each chapter:

- **Chapter 2: High-DoF Object Modeling for Robotic Manipulation**
  - Introducing a novel histogram feature representation of highly deformable materials (HOW-features) computed directly from streaming RGB data using Gabor filters.
  - Developing a sparse representation framework employing a visual feedback dictionary, directly correlating histogram features to control instructions.
  - Demonstrating the combination of deformable feature representation, a visual feedback dictionary, and sparse linear representations, allowing us to perform complex manipulation tasks, including human-robot collaboration, without substantial training data.
- **Chapter 3: High-DoF Controller Parameterization**
  - Presenting a novel controller parametrization designed for cloth manipulation applications.
  - Defining the optimal control action on the leaf nodes of a random forest.
  - Integrating both random forest construction and controller optimization with the imitation learning algorithm, enabling them to evolve with training data.
  - Evaluating our method using a 3-task cloth manipulation application, demonstrating its robustness to noise in human motion and observations, as well as its adaptability to evolving training data.
- **Chapter 4: High-DoF Robot System Identification**
  - Introducing a hierarchical, grid-based data structure for system identification in high-DoF soft robots.

  - Identifying a low-dimension to high-dimension mapping function f and storing it in the grid to expedite computation.
  - Effectively reducing the number of grid corner points to be evaluated, resulting in a significant reduction in total running time.

- **Chapter 5: High-DoF System Policy Learning with Reinforcement Learning**
  - Proposing an approach for collecting supervised data for painting tasks through self-supervised learning.
  - Introducing an adapted deep reinforcement learning network trained using both human expert data and self-supervised data, with a predominant reliance on self-supervised data.
  - Implementing an efficient rendering system for generating stroke-based paintings of desired resolutions by our trained painting agent.

- **Chapter 6: High-DoF Policy Transfer from Simulation to Reality**
  - Developing an approach for collecting supervised data for painting tasks through self-supervised learning.
  - Introducing an adapted deep reinforcement learning network trained using human expert data and self-supervised data, with a predominant reliance on self-supervised data.
  - Implementing an efficient rendering system for generating stroke-based paintings of desired resolutions by our trained painting agent.

Chapter 2: High-DoF Object Modelling for Robotic Manipulation

2.1 Introduction

The problem of manipulating highly deformable materials such as clothes and fabrics frequently arises in different applications. These include laundry folding [14], robot-assisted dressing or household chores [29, 30], ironing [17], coat checking [31], sewing [32], and transporting large materials like cloth, leather, and composite materials [33]. Robot manipulation has been extensively studied for decades and there is extensive work on the manipulation of rigid and deformable objects. Compared to the manipulation of a rigid object, the state of which can be completely described by a six-dimensional configuration space, the manipulation of a deformable object is more challenging due to its very high configuration space dimensionality. The resulting manipulation algorithm needs to handle this dimensional complexity and maintain the tension to perform the task.

One practical approach to dealing with general deformable object manipulation is based on visual servoing, [34, 35]. At a broad level, these servoing techniques use perception data captured using cameras and formulate a control policy mapping to compute the velocities of the robotic end-effectors in real-time. However, a key issue in these methods is to automatically extract key low-dimensional features of the deformable material that can be used to compute a mapping. The simplest methods use manual or other techniques to extract features corresponding to line

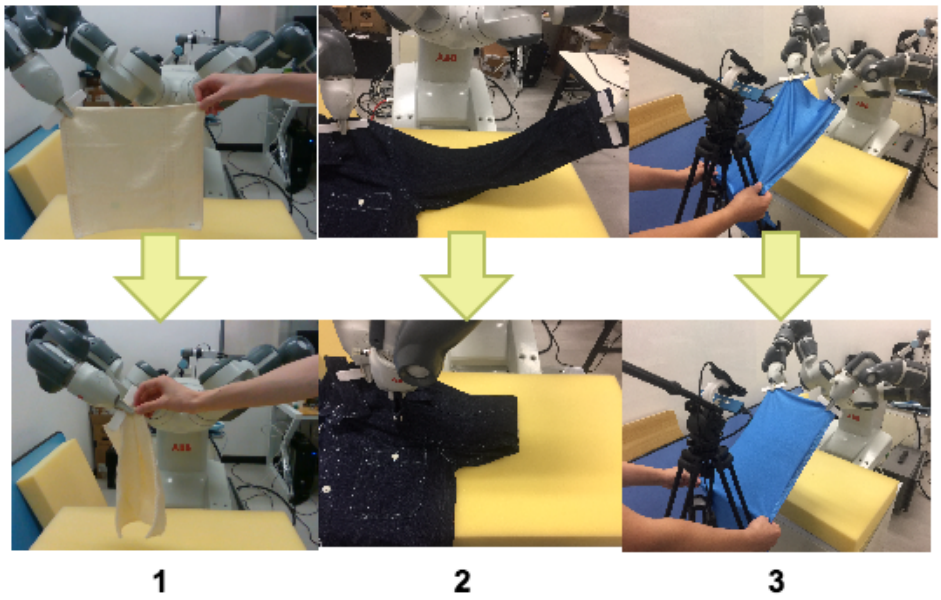


Figure 2.1: **Manipulation Benchmarks:** We highlight the real-time performance of our algorithm on three tasks: (1) human-robot jointly folding a cloth with one hand each; (2) robot folding a cloth with two hands; (3) human-robot stretching a cloth with four combined hands. The top row shows the initial state for each task and the bottom row is the final state. Our approach can handle the perturbations due to human movements.

segments or curvature from a set of points on the surface of the deformable material. In addition, we need appropriate mapping algorithms based on appropriate low-dimensional features. Current methods may not work well while performing complex tasks or when the model undergoes large deformations. Furthermore, in many human-robot systems, the deformable material may undergo unknown perturbations and it is important to design robust manipulation strategies [30, 36].

Main Results: In this paper, we present a novel feature representation, a histogram of oriented wrinkles (HOW), to describe the shape variation of a highly deformable object like clothing. These features are computed by applying Gabor filters and extracting the high-frequency and low-frequency components. We precompute a visual feedback dictionary using an offline training phase that stores a mapping between these visual features and the velocity of the end-effector. At runtime, we automatically compute the goal configurations based on the manipulation task and

use sparse linear representation to compute the velocity of the controller from the dictionary (Section III). Furthermore, we extend our approach so that it can be used in human-robot collaborative settings. Compared to prior manipulation algorithms, the novel components of our work include:

- A novel histogram feature representation of highly deformable materials (HOW-features) that are computed directly from the streaming RGB data using Gabor filters (Section 2.4).
- A sparse representation framework using a visual feedback dictionary, which directly correlates the histogram features to the control instructions (Section 2.5).
- The combination of deformable feature representation, a visual feedback dictionary, and sparse linear representations that enable us to perform complex manipulation tasks, including human-robot collaboration, without significant training data (Section 2.5.3).

We have integrated our approach with an ABB YuMi dual-arm robot and a camera for image capture and used it to manipulate different cloth materials for different tasks. We highlight the real-time performance for tasks related to stretching, folding, and placement (Section 2.6).

2.2 Related Work

Many techniques have been proposed for the automated manipulation of flexible materials. Some of them have been designed for specific tasks, such as peg-in-hole and laying down tasks with small elastic parts [37] or wrapping a cloth around a rigid surface [38]. There is extensive work on folding laundry using pre-planned materials. In this section, we give a brief review of prior work on deformable object representation, manipulation, and visual servoing.

2.2.1 Deformable Object Representation

The recognition and detection of deformable object characteristics is essential for manipulation tasks. There is extensive work in computer vision and related areas on tracking features of deformable models. Some of the early work is based on active deformable models [39]. Ramisa *et al.* [40] identify the grasping positions on a cloth with many wrinkles using a bag-of-features-based detector. Li *et al.* [41] encode the category and pose of a deformable object by collecting a large set of training data in the form of depth images from different view points using offline simulation.

2.2.2 Deformable Object Manipulation

Robotic manipulation of general deformable objects relies on a combination of different sensor measurements. The RGB images or RGB-Depth data are widely used for deformable object manipulation [14, 17, 35]. Fiducial markers can also be printed on the deformable object to improve the manipulation performance [42]. In addition to visual perception, information from other sensors can also be helpful, like the use of contact force sensing to maintain the tension [33]. In many cases, simulation techniques are used for manipulation planning. Clegg *et al.* [43] use reinforcement learning to train a controller for haptics-based manipulation. Bai *et al.* [44] use physically-based optimization to learn a deformable object manipulation policy for a dexterous gripper. McConachie *et al.* [45] formulate model selection for deformable object manipulation and introduces a utility metric to measure the performance of a specific model. These simulation-based approaches need accurate deformation material properties, which can be difficult to achieve. Data-driven techniques have been used to design the control policies for

deformable object manipulation. Yang *et al.* [21] propose an end-to-end framework to learn a control policy using deep learning techniques for folding clothes. Cusumano-Towner *et al.* [46] learn a Hidden Markov Model (HMM) using a sequence of manipulation actions and observations.

2.2.3 Visual Servoing for Deformable Objects

Visual servoing techniques [47, 48] aim at controlling a dynamic system using visual features extracted from images. They have been widely used in robotic tasks like manipulation and navigation. Recent work includes the use of histogram features for rigid objects [49]. Sullivan *et al.* [39] use a visual servoing technique to solve the deformable object manipulation problem using active models. Navarro-Alarcon *et al.* [34, 35] use an adaptive and model-free linear controller to servo-control soft objects, where the object's deformation is modeled using a spring model [50]. Langsfeld *et al.* [51] perform online learning of part deformation models for robot cleaning of compliant objects. Our goal is to extend these visual servoing methods to perform complex tasks on highly deformable materials.

2.3 Overview

In this section, we introduce the notation used in the paper. We also present a formulation of the deformable object manipulation problem. Next, we give a brief background on visual servoing and the feedback dictionary. Finally, we give an overview of our deformable manipulation algorithm that uses a visual feedback dictionary.

Symbol	Meaning
m	state of the deformable object
r	robot's end-effector configuration
v	robot's end-effector velocity, $v = \dot{r}$
$I(w, h)$	image from the camera, of size (w_I, h_I)
$s(I)$	HOW-feature vector extracted from image I
$d_i(I)$	the i-th deformation kernel filter
$I^{(t)}, r^{(t)}$	time index t of images and robot configurations
$\{\Delta s^{(i)}\}$,$\{\Delta r^{(i)}\}$	features and labels of the visual feedback dictionary
$\rho(\cdot, \cdot)$	error function between two items
λ	positive feedback gain
L	interaction matrix linking velocities of the feature space to the end-effector configuration space
F	interaction function linking velocities of the feature space to the end-effector configuration space
$p_j(i)$	j_{th} histogram of value i
$N_{dof,I,r,d}$	number of degrees of freedom of the manipulator images $\{I^{(t)}\}$, samples $\{r^{(t)}\}$,filters $\{d^{(t)}\}$
C_{fr}	constant of frame rate
r^*, s^*, m^*	desired target configuration, feature, state
$\hat{r}, \hat{s}, \hat{m}$	approximated current configuration, feature, state

Table 2.1: Symbols

2.3.1 Problem Formulation

The goal of manipulating a highly deformable material is to drive a soft object towards a given target state (m^*) from its current state (m). The state of a deformation object (m) can be complex due to its high dimensional configuration. In our formulation, we do not represent this state explicitly and treat it as a hidden variable. Instead, we keep track of the deformable object's current state in the feature space (s) and its desired state (s^*), based on HOW-features.

The end-effector's configuration r is represented using the Cartesian coordinates and the orientations of end-effectors or the degree of each joint of the robot. When r corresponds to the Cartesian coordinates of the end-effectors, an extra step is performed by the inverse kinematics motion planner [52] to map the velocity v to the actual controlling parameters.

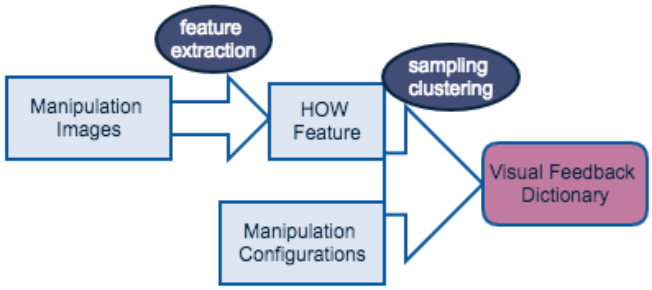


Figure 2.2: **Computing the Visual Feedback Dictionary:** The input to this offline process is the recorded manipulation data with images and the corresponding configuration of the robot's end-effector. The output is a visual feedback dictionary, which links the velocity of the features and the controller.

The visual servo-control is performed by computing an appropriate velocity (v) of the end-effectors. Given the visual feedback about the deformable object, the control velocity (v) reduces the error in the feature space ($s - s^*$). After continuously applying feedback control, the robot will manipulate the deformable object toward its goal configuration. In this way, the feedback controller can be formulated as computing a mapping between the velocity in the feature space of the deformable object and the velocity in the end-effector's configuration space (r):

$$r^* - r = -\lambda F(s - s^*) \tag{2.1}$$

where F is an interaction function that is used to map the two velocities in different spaces and λ is the feedback gain. This formulation works well only if some standard assumptions related to the characteristics of the deformable object hold. These include infinite flexibility with no energy contribution from bending, no dynamics, and being subject to gravity and downward tendency (see details in [53]). For our tasks of manipulating highly deformable materials like clothes or laundry at a low speed, such assumptions are valid.

2.3.2 Visual Servoing

In this section we give a brief overview of visual servoing [47, 49, 54, 55], which is used in our algorithm. In general-purpose visual servoing, the robot wants to move an object from its original configuration (r) towards a desired configuration (r^*). In our case, the object's configuration (r) also corresponds to the configuration of the robot's end-effector, because the rigid object is fixed relative to the end-effectors during the manipulation task. These methods use a cost function $\rho(\cdot, \cdot)$ as the error measurement in the image space and the visual servoing problem can be formulated as an optimization problem:

$$\hat{r^*} = \mathbf{argmin}\ r\rho(r, r^*) \quad (2.2)$$

where $\hat{r^*}$ is the configuration of the end-effector after the optimization step and is the closest possible configuration to the desired position (r^*). In the optimal scenario, $\hat{r^*} = r^*$.

Let s be a set of HOW-features extracted from the image. Depending on the configuration r, the visual servoing task is equivalent to minimizing the Euclidean distance in the feature space and this can be expressed as:

$$\hat{r^*} = \mathbf{argmin}\ ((s(r) - s^*)^T(s(r) - s^*)) \quad (2.3)$$

where $s^* = s(r^*)$ is the HOW-feature corresponding to the goal configuration (r^*). The visual servoing problem can be solved by iteratively updating the velocity of the end-effectors according

to the visual feedback:

$$v = -\lambda L_s^+ (s - s^*) \tag{2.4}$$

where L_s^+ is the pseudo-inverse of the interaction matrix $L_s = \frac{\partial s}{\partial r}$. It corresponds to an interaction matrix that links the variation of the velocity in the feature space $\dot{s}$ to the velocity in the end-effector's configuration space. L_s^+ can be computed offline by training data defined in [49].

2.3.3 Visual Feedback Dictionary

The visual feedback dictionary corresponds to a set of vectors with instances of visual feedback $\{\Delta s^{(i)}\}$ and the corresponding velocities of the end-effectors $\{\Delta r^{(i)}\}$, where $\{\Delta s^{(i)}\} = (s - s^*)$. Furthermore, we refer to each instance $\{\Delta s^{(i)}, \Delta r^{(i)}\}$ as the *visual feedback word*. This dictionary is computed from the recorded manipulation data. The input includes a set of images ($\{I^{(t)}\}$) and the end-effector configurations ($\{r^{(t)}\}$). Its output is computed as ($\{\{\Delta s^{(i)}\}, \{\Delta r^{(i)}\}\}$). We compute this dictionary during an offline learning phase using sampling and clustering methods (see Algorithm 2 for details), and use this dictionary at runtime to compute the velocity of the controller by computing the sparse decomposition of a feedback Δs. More details about computing the visual feedback dictionary are give in Algorithm 2.

2.3.4 Offline and Runtime Computations

Our approach computes the visual dictionary using an offline process (see Fig. 2.2). Our runtime algorithm consists of two components. Given the camera stream, we extract the HOW-features from the image ($s(I)$) and compute the corresponding velocity of the end-effector using an appropriate mapping. For the controlling procedure, we use the sparse representation method

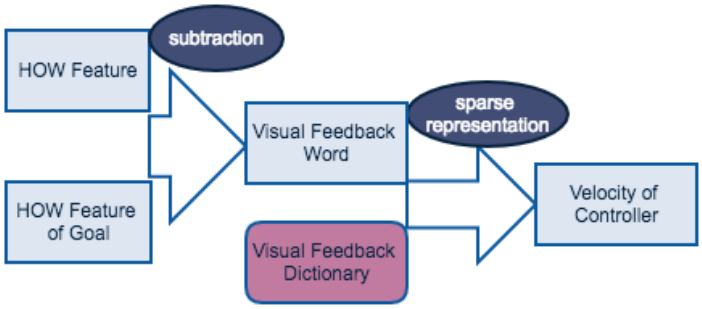


Figure 2.3: **Runtime Algorithm:** The runtime computation consists of two stages. We extract the deformable features (HOW-features) from the visual input and computes the visual feedback word by subtracting the extracted features from the features of the goal configuration. We apply the sparse representation and compute the velocity of the controller for manipulation.

to compute the interaction matrix, as opposed to directly solving the optimization problem (Equation 2.2). In practice, the sparse representation method is more efficient. The runtime phase performs the actual visual servoing with the current image at time t ($I^{(t)}$), the visual feedback dictionary ($\{\{\Delta s^{(i)}\}, \{\Delta r^{(i)}\}\}$), and the desired goal configurations given by (I^*) as the input.

2.4 Histogram of Deformation Model Feature

In this section, we present our algorithm to compute the HOW-features from the camera stream. These are low-dimensional features of the highly deformable material. The pipeline of our HOW-feature computation process is shown in Figure 2.4. Next, we explain each of these stages in detail.

2.4.1 Foreground Segmentation

To find the foreground partition of a cloth, we apply the Gaussian mixture algorithm [56] on the RGB data captured by the camera. The intermediate result of segmentation is shown in Figure 2.4(2).

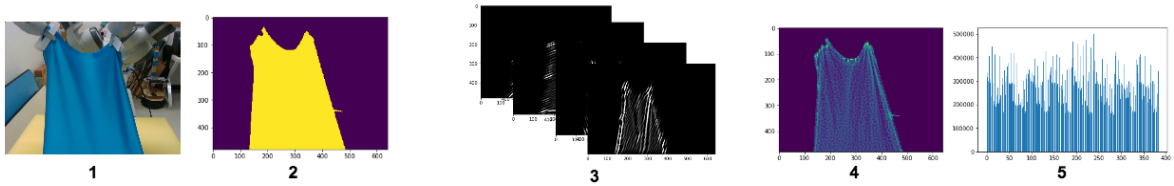


Figure 2.4: **Pipeline for HOW-feature Computation:** We use the following stages for the input image (1): (2) Foreground segmentation using Gaussian mixture; (3) Image filtering with multiple orientations and wavelengths of Gabor Kernel; (4) Discretization of the filtered images to form grids of histogram; (5) Stacking the feature matrix to a single column vector.

2.4.2 Deformation Enhancement

To model the high dimensional characteristics of the highly deformable material, we use deformation enhancement. This is based on the perceptual observation that most deformations can be modeled by shadows and shape variations. Therefore, we extract the features corresponding to shadow variations by applying a Gabor transform [57] to the RGB image. This results in the enhancement of the ridges, wrinkles, and edges (see Figure 2.4). We convolve the N deformation filters $\{d_i\}$ to the image I and represent the result as $\{d_i(I)\}$.

In the spatial domain, a 2D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave [58] and it has been used to detect wrinkles [59]. The 2D Gabor filter can be represented as follows:

$$g(x, y; \lambda, \theta, \phi, \sigma, \gamma) = \exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) \sin(2\pi \frac{x'}{\lambda} + \phi), \tag{2.5}$$

where $x' = x\cos\theta + y\sin\theta$, $y' = -x\sin\theta + y\cos\theta$, θ is the orientation of the normal to the parallel stripes of the Gabor filter, λ is the wavelength of the sinusoidal factor, ϕ is the phase offset, σ is the standard deviation of the Gaussian, and γ is the spatial aspect ratio. When we apply

the Gabor filter to our deformation model image, the choice of wavelength (λ) and orientation (θ) are the key parameters with respect to the wrinkles of deformable materials. As a result, the deformation model features consist of multiple Gabor filters ($d_{1\cdots n}(I)$) with different values of wavelengths (λ) and orientations (θ).

2.4.3 Grids of Histogram

A histogram-based feature is an approximation of the image which can reduce the data redundancy and extract a high-level representation that is robust to local variations in an image. Histogram-based features have been adopted to achieve a general framework for photometric visual servoing [49]. Although the distribution of the pixel value can be represented by a histogram, it is also significant to represent the position in the feature space of the deformation to achieve the manipulation task. Our approach is inspired by the study of grids of Histogram of Oriented Gradient [60], which is computed on a dense grid of uniformly spatial cells.

We compute the grids of histogram of deformation model feature by dividing the image into small spatial regions and accumulating local histogram of different filters (d_i) of the region. For each grid, we compute the histogram in the region and represent it as a matrix. We vary the grid size and compute matrix features for each size. Finally, we represent the entries of a matrix as a column feature vector. The complete feature extraction process is described in Algorithm 1.

The HOW-feature has several advantages. It captures the deformation structure, which is based on the characteristics of the local shape. Moreover, it uses a local representation that is invariant to local geometric and photometric transformations. This is useful when the translations or rotations are much smaller than the local spatial or orientation grid size.

Algorithm 1 Computing HOW-Features

Input: image I of size (w_I, h_I), deformation filtering or Gabor kernels $\{d_1 \cdots d_{N_d}\}$, grid size set$\{g_1, \cdots, g_{N_g}\}$.
Output: feature vector s

```
for i = 1, ..., N_d do
    for j = 1, ..., N_g do
        for (w, h) = (1, 1), ..., (w_I, h_I) do
            (x, y) = (TRUNC(w/g_j), TRUNC(h/g_j)) // compute the indices using truncation
            s_{i,j,x,y} = s_{i,j,x,y} + d_i(I(w, h)) //add the filtered pixel value to the specific bin of the grid
        end for
    end for
end for
return s =0
```

2.5 Manipulation using Visual Feedback Dictionary

In this section, we present our algorithm for computing the visual feedback dictionary. At runtime, this dictionary is used to compute the corresponding velocity (Δr) of the controller based on the visual feedback ($\Delta s(I)$).

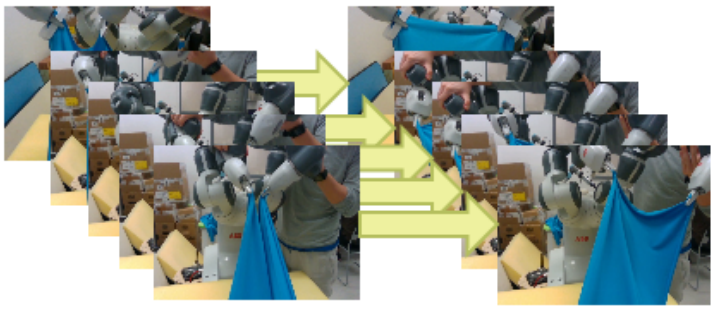


Figure 2.5: **Visual Feedback Dictionary:** The visual feedback word is defined by the difference between the visual features $\Delta s = s - s^*$ and the controller positions $\Delta r = r - r^*$. The visual feedback dictionary $\{\{\Delta s^{(i)}\}, \{\Delta r^{(i)}\}\}$ consists of visual feedback words computed. We show the initial and final states on the left and right, respectively.

2.5.1 Building Visual Feedback Dictionary

As shown in Figure 2.2, the inputs of the offline training phase are a set of images and end-effector configurations ($\{I^{(t)}\}$, $\{r^{(t)}\}$) and the output is the visual feedback dictionary ($\{\{\Delta s^{(i)}\}, \{\Delta r^{(i)}\}\}$).

For the training process, the end-effector configurations, ($\{r^{(t)}\}$), are either collected by human tele-operation or generated randomly. A single configuration ($r^{(i)}$) of the robot is a column vector of length N_{dof}, the number of degrees-of-freedom to be controlled. $r \in \mathbb{R}^{N_{dof}}$ and its value is represented in the configuration space.

In order to compute the mapping (F_H) from the visual feedback to the velocity, we need to transform the configurations $\{r^{(t)}\}$ and image stream $\{I^{(t)}\}$ into velocities $\{\Delta r^{(t)}\}$ and the visual feature feedback $\{\Delta s^{(t)}\}$. One solution is to select a fixed time step Δt and to represent the velocity in both the feature and the configuration space as:

$$\Delta r^{(t)} = \frac{r^{(t+\frac{\Delta t}{2})} - r^{(t-\frac{\Delta t}{2})}}{\Delta t / C_{fr}};\ \Delta s(I^{(t)}) = \frac{s(I^{(t+\frac{\Delta t}{2})}) - s(I^{(i-\frac{\Delta t}{2})})}{\Delta t / C_{fr}}$$

where C_{fr} is the frame rate of the captured video.

However, sampling by a fixed time step (Δt) leads to a limited number of samples ($N_I - \Delta t$) and can result in over-fitting. To overcome this issue, we break the sequential order of the time index to generate more training data from $I^{(t)}$ and $r^{(t)}$. In particular, we assume the manipulation task can be observed as a Markov process [61] and each step is independent from every other. In this case, the sampling rates are given as follows, (when the total sampling amount is n):

$$\Delta r^{(t)} = \frac{r^{(p_t)} - r^{(p_{t+n})}}{(p_t - p_{t+n})/C_{fr}};\ \Delta s(I^{(t)}) = \frac{s(I^{(p_t)}) - s(I^{(p_{t+n})})}{(p_t - p_{t+n})/C_{fr}}$$

where $p_{1,\cdots,2n}$ is a set of indices randomly generated, and $p_t \in [1 \cdots N_I]$. In order to build a more concise dictionary, we also apply K-Means Clustering [62] on the feature space, which enhance the performance and prevent the over-fitting problem.

In practice, the visual feedback dictionary can be regarded as an approximation of the interaction function F (see Equation 2.1). The overall algorithm to compute the dictionary is given in Algorithm 2.

2.5.2 Sparse Representation

At runtime, we use sparse linear representation [63] to compute the velocity of the controller from the visual feedback dictionary. These representations tend to assign zero weights to most irrelevant or redundant features and are used to find a small subset of most predictive features in the high dimensional feature space. Given a noisy observation of a feature at runtime (s) and the visual feedback dictionary constructed by features $\{\Delta s^{(i)}\}$ with labels $\{\Delta r^{(i)}\}$, we represent Δs by $\hat{\Delta} s$, which is a sparse linear combination of $\{\Delta s^{(i)}\}$, where β is the sparsity-inducing L_1 term. To deal with noisy data, we rather use the L_2 norm on the data-fitting term and formulate the resulting sparse representation as:

$$\hat{\beta} = \underset{\beta}{\mathbf{argmin}} \left(||min(\Delta s - \sum_i \beta_i \Delta s^{(i)})||_2^2 + \alpha ||\beta||_1 \right) \tag{2.6}$$

where α is a slack variable that is used to balance the trade-off between fitting the data perfectly and using a sparse solution. The sparse coefficient β^* is computed using a minimization

Algorithm 2 Building the Visual Feedback Dictionary

Input: image stream $\{I^{(t)}\}$ and positions of end-effectors $\{r^{(t)}\}$ with sampling amount n, dictionary size N_{dic}

Output: Visual Feedback Dictionary $\{\{\Delta s_d^{(i)}\}, \{\Delta r_d^{(i)}\}\}$

$\{\Delta s_d\} = \{\}, \{\Delta r_d\} = \{\}$
$p = N_I \text{RAND}(2n)$ // generate random indices for sampling
for $i = 1, \cdots, n$ **do**
    $\Delta s^{(i)} = s(I^{(p(i))}) - s(I^{(p(i+n))})$ // sampling
    $\Delta r^{(i)} = r^{(p(i))} - r^{(p(i+n))}$ // sampling
end for
$centers = \text{K-MEANS}(\{\Delta s^{(i)}\}, N_{dic})$ // compute the centers of the feature set for clustering
for $i = 1, \cdots, N_{dic}$ **do**
    $j = \underset{i}{\mathbf{argmin}} \ (centers[i] - s^{(i)})$
    $\{\Delta s_d\} = \{\Delta s_d, \Delta s^{(j)}\}$ $\{\Delta r_d\} = \{\Delta r_d, \Delta r^{(j)}\}$
end for
return $\{\Delta s_d\}, \{\Delta r_d\}$ =0

formulation:

$$\beta^* = \underset{\beta}{\mathbf{argmin}} \ (\sum_i \rho(\Delta s_i^* - \sum_j \beta_j \Delta s_i^{(j)}) + \alpha \sum_j ||\beta_j||_1) \tag{2.7}$$

After $\hat{\beta}$ is computed, the observation $\hat{\Delta s}$ and the probable label $\hat{\Delta r}$ can be reconstructed by the visual feedback dictionary :

$$\hat{\Delta s} = \sum_i \hat{\beta}_i \Delta s^{(i)} \qquad \hat{\Delta r} = \sum_i \hat{\beta}_i \Delta r^{(i)} \tag{2.8}$$

The corresponding Δr^* of the $i-$th DOF in the configuration is given as:

$$\Delta r_i^* = \sum_j \beta_j^* \Delta r_i^{(j)}, \tag{2.9}$$

where $\Delta s_i^{(j)}$ denotes the $i-$th datum of the $j-$th feature , Δs_i^* denotes the value of the response, and the norm-1 regularizer $\sum_j ||\beta_j||_1$ typically results in a sparse solution in the feature space.

2.5.3 Goal Configuration and Mapping

We compute the goal configuration and the corresponding HOW-features based on the underlying manipulation task at runtime. Based on the configuration, we compute the velocity of the end-effector. The different ways to compute the goal configuration are:

- For the task of manipulating deformable objects to a single state m^*, the goal configuration can be represented simply by the visual feature of the desired state $s^* = s(I^*)$.

- For the task of manipulating deformable objects to a hidden state h^*, which can be represented by a set of states of the object $h^* = \{m_1, \cdots, m_n\}$ as a set of visual features $\{s(I_1), \cdots, s(I_n)\}$. We modify the formulation in Equation 2.1 to compute v as:

$$v = -\lambda \min_i (F(s(I) - s(I_j))) \tag{2.10}$$

- For a complex task, which can be represented using a sequential set of states $\{m_1, \cdots, m_n\}$, we estimate the sequential cost of each state as $c(m_i)$. We use a modification that tends to compute the state with the lowest sequential cost:

$$i^* = \textbf{argmin}\ \ i(c(m_i) - \lambda F(s(I) - s(I_i))). \tag{2.11}$$

  After i^* is computed, the velocity for state m_i is determined by $s(I_{i^*})$, and m_i is removed from the set of goals for subsequent computations.

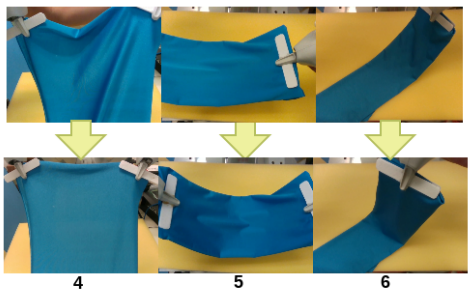


Figure 2.6: **Manipulation Benchmarks:** We highlight three benchmarks corresponding to: (4) flattening; (5) placement; (6) folding. Top Row: The initial state of each task. Bottom Row: The goal state of each task. More details are given in Fig. 11.

2.5.4 Human Robot Interaction

In many situations, the robot is working next to the human. The human is either grasping the deformable object or applying force. We classify the human-robot manipulation task using the hidden state goal h^*, where we need to estimate the human's action repeatedly. As the human intention is unknown to the robot, the resulting deformable material is assigned several goal states $\{m_1, \cdots, m_n\}$, which are determined conditionally by the action of human.

2.6 Implementation

In this section, we describe our implementation and the experimental setup, including the robot and the camera. We highlight the performance on several manipulation tasks performed by the robot only or robot-human collaboration. We also highlight the benefits of using HOW-features and the visual feedback dictionary.

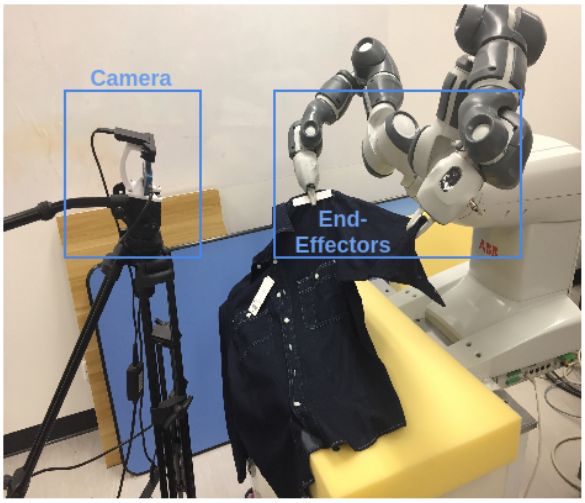


Figure 2.7: **Setup for Manipulation Tasks:** We use an 12-DOF dual-arm ABB YuMi and an RGB camera to perform complex manipulation tasks using visual servoing, with and without humans.

2.6.1 Robot Setup and Benchmarks

Our algorithm was implemented on a PC and integrated with an ABB YuMi dual-arm robot with 12-DOF to capture $\{r^{(t)}\}$ and perform manipulation tasks. We use a RealSense camera is used to capture the RGB videos at (640×480) resolution. In practice, we compute the Cartesian coordinates of the end-effectors of the ABB YuMi as the controlling configuration $r \in \mathbb{R}^6$ and use an inverse kinematics-based motion planner [52] directly. The setup is shown in Figure 2.7.

To evaluate the effectiveness of our deformable manipulation framework, we use 6 benchmarks with different clothes, which have different material characteristics and shapes. Moreover, we use different initial and goal states depending on the task, e.g. stretching or folding. The details are listed in Table 2.2. In these tasks, we use three different forms of goal configurations for the deformable object manipulations, as discussed in Section 2.5.3. For benchmarks 4-6, the task corresponds to manipulating the cloth without human participation and we specify the goal configurations. For benchmarks 1-3, the task is to manipulate the cloth with human participa-

Benchmark#	Object	Initial State	Task/Goal
1	towel	unfold in the air	fold with human
2	shirt	shape (set by human)	fixed shape
3	unstretchable cloth	position (set by human)	fixed shape
4	stretchable cloth	random	flattening
5	stretchable cloth	random	placement
6	stretchable cloth	unfolded shape on desk	folded shape

Table 2.2: **Benchmark Tasks:** We highlight various complex manipulation tasks performed using our algorithm. Three of them involve human-robot collaboration and we demonstrate that our method can handle external forces and perturbations applied to the cloth. We use cloth benchmarks of different material characteristics. The initial state is a random configuration or an unfolded cloth on a table, and we specify the final configuration for the task. The benchmark numbers correspond to the numbers shown in Fig. 2.1 and Fig. 2.6

tion. The human is assisted with the task, but the task also introduces external perturbations. Our approach makes no assumptions about the human motion, and only uses the visual feedback to guess the human's behavior. In benchmark 1, the robot must anticipate the human's pace and forces for grasping to fold the towel in the air. In benchmark 2, the robot needs to process a complex task with several goal configurations when performing a folding task. In benchmark 3, the robot is asked to follow the human's actions to maintain the shape of the sheet. All 6 benchmarks are defined with goal states/features of the cloth, regardless of whether if there is a human moving the cloth or not. Because different states the cloth can be precisely represented and corresponding controlling parameters can be computed, the robot can perform complicated tasks as well.

2.6.2 Benefits of HOW-feature

There are many standard techniques to compute low-dimensional features of deformable models from RGB data known in computer vision and image processing. These include standard HOG and color histograms. We evaluated the performance of HOW-features along with the oth-

ers and also explore the combination of these features. The test involves measuring the success rate of the manipulator in moving towards the goal configuration based on the computed velocity, as given by Equation 2.4. We obtain best results in our benchmarks using HOG+HOW features. The HOG features capture the edges in the image and the HOW captures the wrinkles and deformation, so their combination works well. For benchmarks 1 and 2, the shapes of the objects changes significantly and HOW can easily capture the deformation by enhancing the edges. For benchmarks 3, 4, and 5, HOW can capture the deformation by the shadowed area of wrinkles. For benchmark 6, the total shadowed area continuously changes through the process, in which the color histogram describes the feature slightly better.

2.6.3 Benefits of Sparse Representation

The main parameter related to the visual feedback dictionary that governs the performance is its size. At runtime, it is also related to the choice of the slack variable in the sparse representation. As the size of the visual feedback dictionary grows, the velocity error tends to reduce. However, after reaching a certain size, the dictionary contributes less to the control policy mapping. That implies that there is redundancy in the visual feedback dictionary.

The performance of sparse representation computation at runtime is governed by the slack variable α in Equations 2.6 and 2.7. This parameter provides a tradeoff between data fitting and sparse solution and governs the velocity error between the desired velocity v^* and the actual velocity, $||v - v^*||_2$. In practice, α affects the convergence speed. If α is small, the sparse computation has little or no impact and the solution tends to a common linear regression. If α is large, then we suffer from over-fitting.

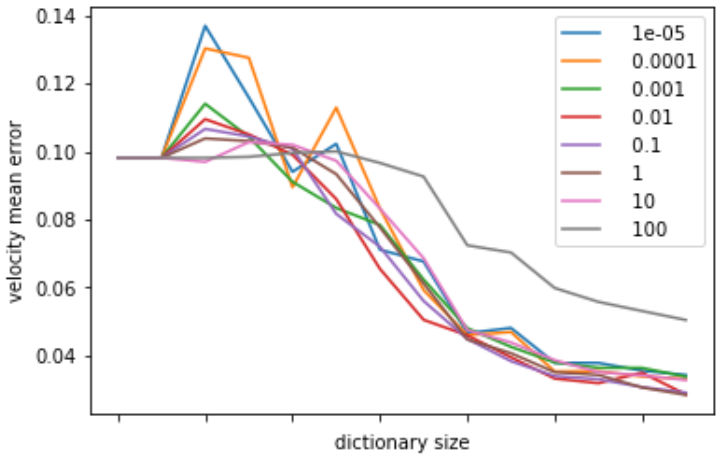


Figure 2.8: **Parameter Selection for Visual Feedback Dictionary and Sparse Representation:** We vary the dictionary size on the X-axis and compute the velocity error for different values of α chosen for sparse representation for benchmark 4.

Feature	Benchmark 4	Benchmark 5	Benchmark 6
HOG	71.44%	67.31%	82.62%
Color Histograms	62.87%	53.67%	**97.04%**
HOW	92.21%	71.97%	85.57%
HOW+HOG	**94.53%**	**84.08%**	95.08%

Table 2.3: **Comparison between Deformable Features:** We evaluated the success rate of the manipulator based on different features in terms of reaching the goal configuration based on the velocity computed using those features. For each experiment, the number of goals equals the number of frames. There are 393, 204 and 330 frames in benchmarks 4, 5, and 6, respectively. Overall, we obtain the best performance by using HOG + HOW features.

2.7 Conclusion, Limitations and Future Work

We present an approach to manipulate deformable materials using visual servoing and a precomputed feedback dictionary. We present a new algorithm to compute HOW-features, which capture the shape variation and local features of the deformable material using limited computational resources. The visual feedback dictionary is precomputed using sampling and clustering techniques and used with sparse representation to compute the velocity of the controller to perform the task. We highlight the performance on a 12-DOF ABB dual arm and perform complex

tasks related to stretching, folding, and placement. Furthermore, our approach can also be used for human-robot collaborative tasks.

Our approach has some limitations. The effectiveness of the manipulation algorithm is governed by the training data of the specific task, and the goal state is defined by the demonstration. Because HOW-features are computed from 2D images, the accuracy of the computations can also vary based on the illumination and relative colors of the cloth. There are many avenues for future work. Besides overcoming these limitations, we would like to make our approach robust to the training data and the variation of the environment. Furthermore, we could use a more effective method for collecting the training data and generate a unified visual feedback dictionary for different tasks.

Chapter 3: High-DoF Controller Parameterization

3.1 Introduction

High-DOF deformable object manipulation, such as cloth manipulation, is an important and challenging problem in robotics and related areas. It has many applications, including assisted human dressing [64], cloth folding [65], sewing [66], etc. Compared with rigid bodies or three-dimensional volumetric deformable objects [67], cloth can undergo large deformations and form wrinkles or folds, which greatly increases the complexity of cloth manipulation tasks. The possibility of such large deformations is the major challenge in designing a cloth manipulation controller. In a real-life cloth manipulation task, a typical robot only observes a single RGB(D) image of the cloth. As a result, we need robust methods that can perform such complex manipulation tasks based on a single view observation. This involves inferring the 3D configuration of the cloth from the image-based representation and compute the appropriate control action. For example, if a robot manipulates a piece of cloth by holding two corners of the cloth mesh, then the controller should infer the desired end-effector positions of the robot.

Several machine learning models have been proposed to parameterize such controllers, some of which have been used for cloth manipulations. Because of the recent development of deep (reinforcement) learning, one prominent method [68] is to represent feature extraction and controller parametrization as two neural networks, which are trained either jointly or separately.

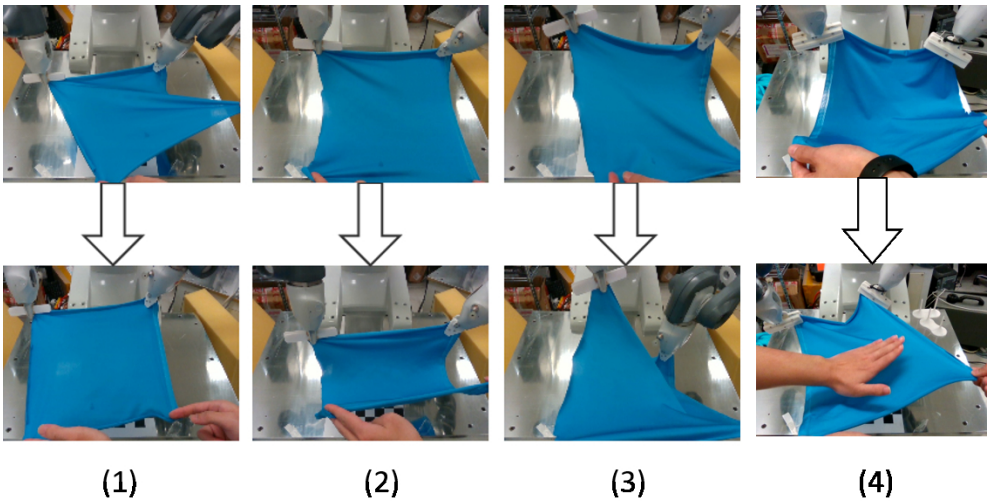


Figure 3.1: **Manipulation Benchmarks:** We highlight the realtime performance of our algorithm on three basic robot-human collaboration tasks. (1): keep the cloth straight; (2): keep the cloth bent; (3): keep the cloth twisted. (4): add noise to the human actions and the visual RGB-D outputs and evaluated the robustness of our approach. (5): evaluate the performance on complex tasks that simultaneously perform straightening, bending, and twist operations to highlight the benefits of our approach.

Other works, such as [69], use one unified neural network architecture, but the structures of these neural networks are determined via trial and error. Recently, [2] represented the controller as a set of observations/control-signal pairs constructed manually. However, due to observation noise at runtime, it is not clear whether this constructed set can cover the experienced cases.

Main Result: In this paper, we present a new method for cloth manipulation. Our method represents the controller as a random-forest. The random-forest takes the observation of the cloth configuration, an RGB(-D) image, as input. It then classifies the input by bringing it to a leaf-node of each decision tree. The optimal control signals are stored on the leaf-node and used as controller outputs. The random-forest is trained iteratively using imitation learning by collecting a dataset online. In each iteration, more data are collected and the random-forest is retrained to be more robust to observation noises.

Compared with other parametric models such as neural networks, random-forest is non-

parametric and the number of leaf-nodes can be dynamically adjusted. As a result, arbitrarily complex cloth configurations can be represented as more training data are provided. Compared with other non-parametric methods such as nearest neighbor, random-forest exhibits better robustness in terms of avoiding over-fitting. We show that as more iterations of imitation learning are performed, the number of leaf-nodes in a random-forest will converge.

We compare the performance of different controller models on three cloth manipulation tasks involving large deformations: cloth flattening, cloth folding, and cloth twisting. The results show that our model always outperforms nearest neighbor [2] and neural networks in terms of matching optimal control signals and robustness to noise. In addition, the number of leaf-nodes converges as imitation learning progresses.

The rest of the paper is organized as follows. Section 3.2 reviews related works. In Section 4.3, we introduce the notation and formulate the problem. In Section 3.4, we provide details for training the random-forest-based controller. Finally, we highlight the performance on challenging benchmarks in Section 3.5 and compare the performance with prior methods.

3.2 Related Work

In this section, we give a brief summary of prior works on large deformation and manipulation, dimension reduction, and controller optimization.

Large Deformation and Manipulation: Different techniques have been proposed for motion planning for deformable objects. Most of these works (e.g., [67, 70, 71]) focus on volumetric objects such as a deforming ball or linear deformable objects such as steerable needles. By comparison, cloth-like thin-shell objects tend to exhibit more complex deformations, forming wrinkles

and folds. Current solutions for thin-shelled manipulation problems are limited to specific tasks, including folding [65, 72, 73], ironing [74], sewing [66], and dressing [64]. On the other hand, deformable body tracking solves a simpler problem, namely inferring the 3D configuration of a deformable object from sensing inputs. There is literature on deformable body tracking, which infers the 3D configuration from sensor data [75–77]. However, these methods usually require a template mesh as a priori and are mainly limited to handling small deformations.

Dimension Reduction: Previous DOM methods use various feature extraction and dimensionality reduction techniques, including SIFT-features [74], HOW-features [2], and depth-based features [78–80]. Recently, deep neural networks have also been used as general-purpose feature extractors. They have also been used to manipulate low-DOF articulated bodies [81] and in DOM applications [21, 82]. For simplicity, our random-forest uses HOW-features as inputs. Another feature recently proposed in [83] represents cloth using a small set of feature points. However, these feature points can only characterize small-scale deformations because there can be a lot of occlusions under large deformations.

Controller Optimization: In robotics, reinforcement learning [84], imitation learning [85], and direct trajectory optimization [86] have been used to compute optimal control actions. Trajectory optimization, or a model-based controller, has been used in [65, 74, 87] for DOM applications. Although the resulting algorithms tend to be accurate, these methods cannot be used for realtime applications. For low-DOF robots such as articulated bodies [88], researchers have developed realtime trajectory optimization approaches, but it is difficult to extend them to deformable models due to the high simulation complexity of such models. Currently, realtime performance can only be achieved through learning-based controllers [2, 21, 78, 79], which use supervised learning to train realtime controllers. However, as pointed out in [1], these methods are not robust

Symbol	Meaning
$\mathcal{C}$	3D configuration space of the cloth
$\mathbf{c}$	a configuration of the cloth
$\mathcal{O}(\mathbf{c})$	an observation of cloth
$\mathbf{c}^*$	target configuration of the cloth
$\mathbf{x}$	robot end-effectors' grasping points
$\mathbf{x}^*$	optimal grasping points returned by the expert
P	transfer function encoding cloth dynamics
$\mathbf{dist}$	distance measure between two observations
π	DOM-control policy
α	random-forest topology
β	controller parameters
γ	confidence of leaf-node
θ	parameter sparsity
K	the number of decision trees
l_k	a leaf-node of k-th decision tree
$l_k(\mathcal{O}(\mathbf{c}))$	the leaf-node that $\mathcal{O}(\mathbf{c})$ belongs to
$\mathcal{L}$	labeling function for optimal actions
$\mathcal{F}$	feature transformation for observation

Table 3.1: Symbol table.

in handling unseen data. Therefore, we further improve the robustness by using imitation learning. Apart from imitation learning used in this work, realtime cloth manipulation controllers can also be optimized using reinforcement learning methods as done in [89–91]. Recently, [92–94] proposed using non-rigid registration to transfer human demonstrations of cloth manipulations to real robots and [95] required an adaptive cloth simulator to predict the future state of a cloth. However, these methods require the knowledge of full 3D cloth geometries, which are not available in our applications.

3.3 Problem Formulation

In this section, we introduce our notations and formulate the problem. Our goal is to compute a realtime feedback controller to deform a cloth into an unknown target configuration.

We denote the 3D configuration space of the cloth as $\mathcal{C}$. Typically, a configuration $\mathbf{c} \in \mathcal{C}$ can be discretely represented as a 3D mesh of cloth and the dimension of $\mathcal{C}$ can be in the thousands. However, we assume that only a partial observation $\mathcal{O}(\mathbf{c})$ is known, which is an RGB-D image from a single, fixed point of view in our case. The goal of the controller is to transform $\mathbf{c}$ into a target configuration $\mathbf{c}^*$. We assume that, over the entire process of control, the robot grasps the cloth at a fixed set of N points whose coordinates are $\mathbf{x}$, where $|\mathbf{x}| = 3N$ and the control action is constituted by the desired positions of these grasping points, denoted as $\mathbf{x}^*$. Therefore, the controller corresponds to a function:

$$\mathbf{x}^* = \pi(\mathcal{O}(\mathbf{c})|\beta), \tag{3.1}$$

where β are its learnable parameters. Given x^*, the corresponding joint angles of the robot can then be determined via conventional inverse kinematics. Given the control action, the configuration of the cloth and the grasping points can be given by the following distribution:

$$p(\mathbf{c}_{i+1}, \mathbf{x}_{i+1}|\mathbf{c}_i, \pi(\mathcal{O}(\mathbf{c}_i))). \tag{3.2}$$

This distribution can be a cloth simulator [96] in a simulated environment or it can be obtained from a real-life robot. Note that, although the action is the desired grasping points ($\mathbf{x}^*$), $\mathbf{x}^*$ and $\mathbf{x}_{i+1}$ are generally not the same because the controller's output can violate a robot's kinematic or dynamic constraints.

3.3.1 Controller Optimization Problem

Our main goal is to optimize the learnable parameters β to optimize the performance of the controller, π. This controller optimization problem can take different forms depending on the available information about $\mathbf{c}^*$. If $\mathcal{O}(\mathbf{c}^*)$ is known, then we can define a reward function: $R(\mathbf{c}) = -\mathbf{dist}(\mathcal{O}(\mathbf{c}), \mathcal{O}(\mathbf{c}^*))$, where $\mathbf{dist}$ can be any distance measure between RGB-D images. In this setting, we want to solve the following reinforcement learning problem:

$$\underset{\alpha,\beta}{\mathbf{argmax}} \qquad \mathbf{E}_{\tau\sim\pi}\left[\sum_{i}^{\infty}\gamma^i R(\mathbf{c}_i)\right] \tag{3.3}$$

where $\tau = (\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_\infty)$ is a trajectory sampled according to π, γ is the discount factor, and the subscript figures denote the timesteps. Another widely used setting assumes that $\mathcal{O}(c^*)$ is unknown, but that an expert is available to provide an optimal control action $\pi^*(\mathcal{O}(\mathbf{c}))$. The expert is a ground truth controller following the definition of [85]. In this case, we want to solve the following imitation learning problem:

$$\underset{\alpha,\beta}{\mathbf{argmax}} \qquad \mathbf{E}_{\tau\sim\pi}\left[\sum_{i}^{\infty}\gamma^i \mathbf{dist}(\pi^*(\mathcal{O}(\mathbf{c}_i)), \pi(\mathcal{O}(\mathbf{c}_i)))\right] \tag{3.4}$$

This expert can be easily acquired in a typical human-robot collaboration task. Our method is based on the imitation learning formulation.

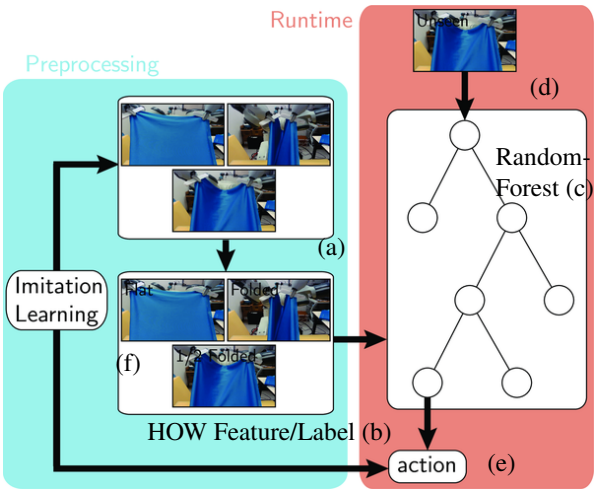


Figure 3.2: **Approach Pipeline:** The pipeline of learning a random-forest-based DOM-controller that maps the visual feature (RGB-D image) to the control action. Given a sampled dataset (a), we first label each data point (shown as red text in (b)) to get a labeled dataset, (b). We then construct a random-forest to classify the images, (c). After training, the random-forest is used as a controller. Given an unseen visual observation (d), the observation is brought through the random-forest to a set of leaf-nodes. The optimal control actions are defined on these leaf-nodes, (e). The entire process of labeling, classification, and controller optimization can be integrated into an IL algorithm, (f).

3.4 Learning Random-Forest-Based Controller

To find the controller parameters, we use an imitation learning algorithm [1], which can be decomposed into two sub-steps: online dataset sampling and controller optimization. The first step samples a dataset $\mathcal{D} = \{\langle \mathcal{O}(\mathbf{c}), \mathbf{x}^* \rangle\}$, where each sample is a combination of cloth observation and optimal action. The second step optimizes the random-forest-based controller with respect to β, given $\mathcal{D}$.

3.4.1 Feature Extraction

Before constructing the random-forest from $\mathcal{D}$, we apply a feature transform to $\mathcal{D}$. Our raw observation of the cloth, $\mathcal{O}(\mathbf{c})$, is an RGB-D image. it has been noted, (e.g., by [97]) that

applying a simple feature transform can improve the accuracy of a classifier such as random-forest. In addition, our input is a 320×240 RGB-D image of the cloth mesh, which corresponds to 76800 entries each having three colors and one depth channel, which is high-dimensional. Therefore, a feature transform effectively reduces the dimensions of the input observation and makes the classifier more robust when the size of the dataset is small.

In our approach, we use HOW-features [2] as the low-dimensional representation. HOW-features is a variant of HOG-features. HOW-features first applies Gabor filters to each patch of the image and then concatenates these patches, resulting in a 768-dimensional feature space. Since each image patch is spatially localized, HOW-features requires each image to be aligned as a pre-processing step. Because our input is an RGB-D image, we can perform a foreground extraction using the depth-channel and then align the image to the center of the screen using the same procedure as in [97]. We summarize this algorithm in Algorithm 3 and denote this feature transform as a function $\mathcal{F}$. The dataset after the feature transform is defined as $\bar{\mathcal{D}} = \{\langle \mathcal{F} \circ \mathcal{O}(\mathbf{c}), \mathbf{x}^* \rangle\}$.

Algorithm 3 Feature extraction operation $\mathcal{F}$.

Input: RGB-D image $\mathcal{O}(\mathbf{c})$
Output: Extracted HOW-feature $\mathcal{F} \circ \mathcal{O}(\mathbf{c})$
0: Foreground extraction using depth channel.
0: Resize/align image to the center of screen using [97].
0: Compute HOW-feature [2]. =0

3.4.2 Random-Forest Construction

Our key contribution is to use a random-forest as the underlying learnable controller in an imitation learning framework. A random-forest is an ensemble of K decision trees, where the k-th tree classifies $\mathcal{F} \circ \mathcal{O}(\mathbf{c})$ by bringing it to a leaf-node $l_k(\mathcal{F} \circ \mathcal{O}(\mathbf{c}))$, where $1 \leq l_k(\mathcal{F} \circ \mathcal{O}(\mathbf{c})) \leq L_k$

and L_k is the number of leaf-nodes in the k-th decision tree. The random-forest makes its decision by classifying $\mathcal{F} \circ \mathcal{O}(\mathbf{c})$ using every decision tree and then computing the average over all the decisions of the trees in the forest. To use an already constructed random-forest as a controller, we define an optimal control action $x^*_{l,k}$ so that the final action is determined by averaging:

$$x^* = \pi(\mathcal{O}(\mathbf{c})|\beta) = \frac{1}{K}\sum_{k=1}^{K} x^*_{l_k(\mathcal{F}\circ\mathcal{O}(\mathbf{c})),k}. \tag{3.5}$$

To construct the random-forest, we use a strategy similar to that in [98]. We construct K binary decision trees in a top-down manner, each using a random subset of $\mathcal{D}$. Specifically, for each node of a tree, a set of random partitions is computed and the one with the maximal Shannon information gain [99] is adopted. Each tree is grown until a maximum depth is reached or the best Shannon information gain is lower than a threshold. The optimal control action of a leaf-node is defined as the average of the control actions of the data sample belonging to that leaf-node.

3.4.3 Imitation Learning

We use an imitation learning algorithm [1] that includes two steps into an outer loop. During each outer iteration, we query an expert, which in our case is a ground-truth hard-coded control algorithm. Specifically, we generate a set of cloth simulation trajectories using a cloth simulator (Equation 3.2). During each timestep of these trajectories, we query the expert to get an optimal control action $\pi^*(\mathcal{O}(\mathbf{c}))$. This optimal control action is combined with the action proposed by our random-forest $\pi(\mathcal{O}(\mathbf{c}))$. The combined action is fed to the simulator to get the next observation. As a result, more data is added into $\mathcal{D}$ and a new random-forest, β, is constructed from a new $\mathcal{D}$. This algorithm is outlined in Algorithm 4.

Algorithm 4 Training DOM-controller using imitation learning algorithm.

Input: Initial guess of β, optimal policy π^*
Output: Optimized β

0: ▷ imitation learning outer loop
0: **while** imitation learning has not converged **do**
0: ▷ Generate training data based on current $\pi(\mathcal{O}(\mathbf{c})|\beta)$
0: Sample $\mathcal{D}$ by querying π^* as in [1]
0: ▷ Extract HOW feature for each data sample
0: Define $\bar{\mathcal{D}} = \emptyset$
0: **for** each $\mathcal{O}(\mathbf{c})$ **do**
0: Extract HOW feature $\mathcal{F} \circ \mathcal{O}(\mathbf{c})$ as in [2]
0: Define $\bar{\mathcal{D}} = \bar{\mathcal{D}} \bigcup \{\langle \mathcal{F} \circ \mathcal{O}(\mathbf{c}), \pi^*(\mathcal{O}(\mathbf{c}))\rangle\}$
0: **end for**
0: ▷ Construct random-forest, i.e., β
0: **for** $1 \leq k \leq K$ **do**
0: Sample random subset of $\bar{\mathcal{D}}$
0: Construct k-th binary decision tree using [98]
0: **end for**
0: **end while**=0

3.4.4 Analysis

In typical DOM applications, data are collected using numerical simulations. Unfortunately, the high dimensionality of $\mathbf{c}$ induces a high computational cost for simulations (i.e. evaluating P in Equation 3.2) and generating a large dataset can be quite difficult. Therefore, we design our method so that it can be used with a small number of data samples. Our method's performance relies on the random-forest's stopping criterion (i.e. the threshold of gain in Shannon entropy). We choose to use a large Shannon entropy threshold so that the random-forest construction stops early, leaving us with a relatively small number of leaf-nodes. We expect that, with a large enough number of imitation learning iterations, the number of nodes in each decision tree of the random-forest will converge. Indeed, such convergence can be guaranteed by the following Lemma.

Lemma: *When the number of imitation learning iterations $N \to \infty$, the distribution*

incurred by the random-forest-based controller will converge to a stationary distribution and the expected classification error of the random-forest will converge to zero.

Proof: By assuming that Algorithm 4 generates a controller π^n at the n-th iteration, Lemma 4.1 of [1] showed that π^n incurs a distribution that converges when $n \to \infty$. Obviously, the number of data samples used to train the random-forest also increases to ∞ with $n \to \infty$. The expected error of the random-forest's classification on a stationary distribution converges to zero according to Theorem 5 of [100]. In Section 3.5, we show that, empirically, the number of leaf-nodes in the random-forest also converges to a fixed value.

3.5 Results

We now describe our implementation and the experimental setup on both simulated environments and real robot hardware. We highlight the performance on several manipulation tasks performed by human-robot collaboration. We also highlight the benefits of using a random-forest-based controller by comparing our method with prior approaches. More implementation details are given in [101].

3.5.1 Robot Setup

We evaluate our method on a simulated environment. For the simulated environment, the robot's kinematics are simulated using Gazebo [102] and the cloth dynamics are simulated using ArcSim [96], a highly accurate cloth simulator. We use OpenGL to capture RGB-D in this simulated environment. Our goal is to manipulate a 35cm$\times$30cm rectangular piece of cloth with four corners initially located at: $v^0 = (0, 0, 0), v^1 = (0.3, 0, 0), v^2 = (0, 0.35, 0), v^3 =$

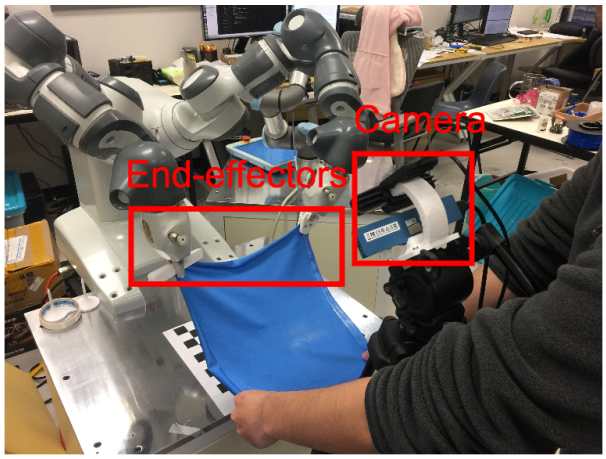


Figure 3.3: **Setup for Manipulation Tasks:** A dual-armed robot and a human are holding four corners of the cloth. We use a 12-DOF dual-armed ABB YuMi and a RealSense RGB-D camera to perform complex manipulation tasks. Our goal is to manipulate a 35cm$\times$30cm rectangular-shaped piece of cloth.

$(0.3, 0.35, 0)$(m). Our manipulator holds the first two corners, v^0, v^1, of the cloth and the environmental uncertainty is modeled by having a human hold the last two corners, v^2, v^3, of the cloth so that we have $\mathbf{x} \triangleq (v^0, v^1)^T$ and each control action is 6-dimensional. The human could move v^2, v^3 to an arbitrary location under the following constraints:

$$\|v^2 - v^3\| \leq 0.3\text{m} \tag{3.6}$$

$$\|(v^2, v^3)_{i+1}^T - (v^2, v^3)_i^T\|_\infty < 0.1(\text{m/s}), \tag{3.7}$$

where the first constraint avoids tearing the cloth apart and the second constraint ensures that the speed of the human hand is slow.

3.5.2 Synthetic Benchmarks

To evaluate the robustness of our method, we design the 3 manipulation tasks listed below:

- Cloth should remain straight in the direction orthogonal to human hands. This is illustrated

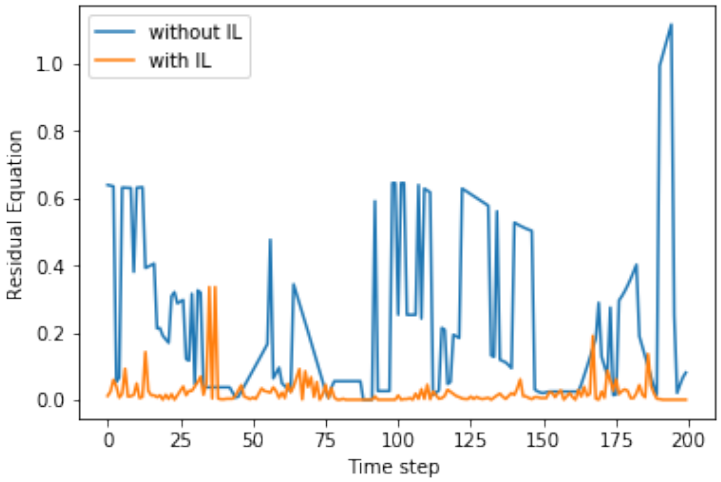


Figure 3.4: **Robustness of the imitation learning algorithm:** In a realtime human-robot interaction, we plot the mean action error (Equation 3.8). The blue curve shows the performance of a controller trained using only one imitation learning iteration (this choice corresponds to supervised learning [2]) and the orange curve shows the performance of a controller trained with 20 iterations. We compare the residuals (Equation 3.8) between the two methods. Increasing the number of iterations in imitation learning significantly reduces the mean action error.

in Figure 3.6 (a). Given v^2, v^3, the robot's end-effector should move to:

$$v^0 = v^2 + 0.35\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|} \; v^1 = v^3 + 0.35\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|}.$$

- Cloth should remain bent in the direction orthogonal to human hands. This is illustrated in Figure 3.6 (b). Given v^2, v^3, the robot's end-effector should move to:

$$v^0 = v^2 + 0.175\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|} \; v^1 = v^3 + 0.175\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|}.$$

- Cloth should remain twisted along the direction orthogonal to human hands. This is illus-

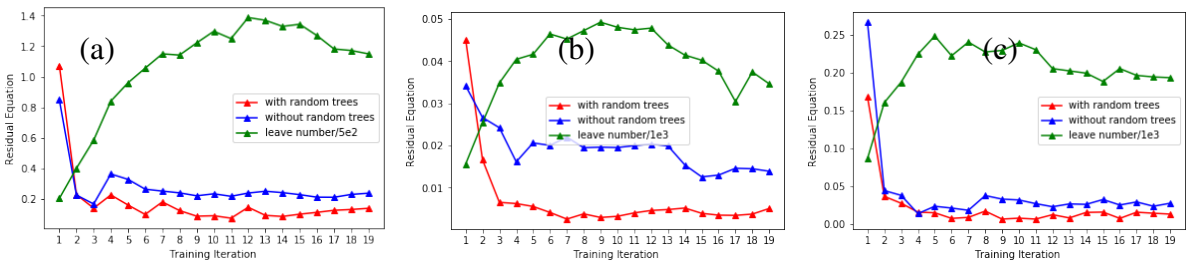


Figure 3.5: **Controller with and without random-forest:** (Red): Residual (Equation 3.8) plotted against the imitation learning iterations (Line 4 of Algorithm 4). (Green): Number of leaf-nodes plotted against the imitation learning iterations. (Blue): Residual (Equation 3.8) plotted against the imitation learning iterations, which precludes random-forest construction. (a): Flatten the cloth; (b): Bend the cloth; (c): Twist the cloth.

trated in Figure 3.6 (c). Given v^2, v^3, the robot's end-effector should move to:

$$v^0 = \frac{v^2 + v^3}{2} + 0.31\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|} + 0.15\frac{(v^3 - v^2) \times (z \times (v^3 - v^2))}{\|(v^3 - v^2) \times (z \times (v^3 - v^2))\|}$$
$$v^1 = \frac{v^2 + v^3}{2} + 0.31\frac{z \times (v^3 - v^2)}{\|z \times (v^3 - v^2)\|} - 0.15\frac{(v^3 - v^2) \times (z \times (v^3 - v^2))}{\|(v^3 - v^2) \times (z \times (v^3 - v^2))\|}.$$

The above formula for determining v^0, v^1 is used to simulate an expert. Note that these equations for the expert requires the knowledge of the four corner positions of the piece of cloth, and such information may not be available in a real robot system that only observes the cloth using a single RGB(D) image. Therefore, we train our random-forest in a simulated environment. These three equations assume that the expert knows the location of the human hands, but that robot does not have this information and it must infer this latent information from a single-view RGB-D image of the current cloth configuration. We also test the performance on complex benchmarks that combine flattening, folding, and twisting, or have considerable occlusion from a single camera.

Although we have only evaluated our method on a simulated environment, we can also deploy our controller on real robot hardware. For the real robotic environment, we use a RealSense

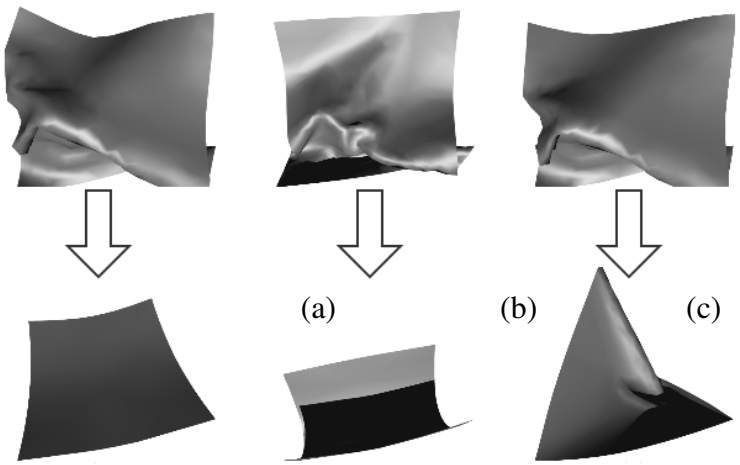


Figure 3.6: **Synthetic Benchmarks:** We highlight the realtime performance of our algorithm on three tasks for the robot simulator (a): keep the cloth straight; (b): keep the cloth bent; (c): keep the cloth twisted.

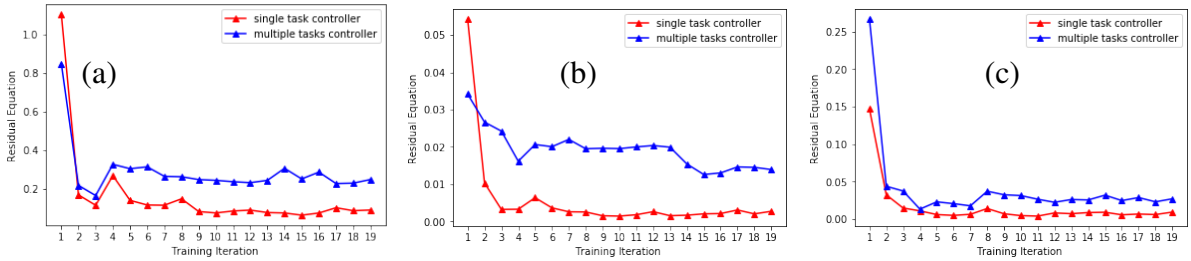


Figure 3.7: **Multi-Task Controller vs. Single-Task Controller:** Residual (Equation 3.8) using a joint 3-task controller (blue) and a single-task controller (red). (a) Flatten the cloth; (b) Bend the cloth; (c) Twist the cloth. Both controllers converge after a few iterations of the imitation learning algorithm. The single-task controller performs slightly better than the multi-task controller with a relative action error of 0.4954%, but the difference is not significant.

depth camera to capture 640×480 RGB-D images and a 12-DOF ABB YuMi dual-armed manipulator to perform the actions, as illustrated in Figure 3.6. More details about the simulation benchmarks and transferring from simulation to real robots can be found in the project webpage.

3.5.3 Multi-task Controller

Unlike single-task controller, a multi-task random-forest-based controller stores multiple actions in a leaf-node. Each observed image is classified by each decision tree in a manner that is

Name	Value
Fraction term used in imitation learning algorithm [1]	0.8
Training data collected in each imitation learning iteration	500
Resolution of RGB-D image	640×480
Dimension of HOW-feature used in [2]	768
Random-forest's stopping criterion when impurity decrease less than [98]	1×10^{-4}

Table 3.2: Meta-parameters used for training.

similar to that of a single-task controller. The leaf node chooses an action according to the id of the task. In this benchmark, we train a 3-task controller for the 3 synthetic tasks in Section 3.5.2. And we transfer the controller to the real robot as benchmark (5) mentioned in Figure 3.1. We combines straightening, bending and twisting to show that our approach can perform complex tasks, as shown in the video. Moreover, we also show tasks which involve occlusion from a single camera viewpoint by adding noise to inputs.

We compare the performances of a single-task controller and a multi-task controller, both of which are based on random-forests. Again, during each evaluation in the simulated environment, the human hands move to 10 random target positions v^{2*}, v^{3*}. As shown in Figure 3.7 (red), we profile the residual (Equation 3.8). Our controller performs consistently well with a relative action error of 0.4954%. We then train a joint 3-task controller. This is performed by defining a single random-forest and defining 3 optimal actions on each leaf-node. The performance of the 3-task controller is compared with that of the single-task controller in Figure 3.7. The multi-task controller performs slightly worse in each task, but the difference is quite small.

3.5.4 Complexity and Algorithm Properties

As illustrated in Algorithm 4, the complexity of our overall approach mainly depends on three parts: dataset sampling, feature extraction, and random-forest construction. When constructing a single decision tree based on the sampled dataset $\bar{D}$, the complexity has an upper bound of $O(|\bar{D}|^2)$. For the construction of a random-forest with K decision trees, the complexity is $O(K|\bar{D}|^2)$.

To evaluate the performance of each component in our method, we run several variants of Algorithm 4. All the meta-parameters used for training are illustrated in Table 3.2. In our first set of experiments, we train a single-task random-forest-based controller for each task and profile the mean action error:

$$err = \sum_{\langle \mathcal{O}(c), \mathbf{x}^* \rangle} \frac{1}{|\mathbf{x}^*||\bar{\mathcal{D}}|} \|\mathbf{x}^* - \frac{1}{K} \mathbf{x}^*_{l_k(\mathcal{F} \circ \mathcal{O}(c)),k}\|^2, \tag{3.8}$$

with respect to the number of imitation learning iterations (Line 4 of Algorithm 4). As illustrated in Figure 3.5 (red), the action error reduces quickly within the first few iterations and later converges. We also plot the number of leaf-nodes in our random-forest in Figure 3.5 (green). As more iterations are performed, the number of leaf-nodes in our random-forest also converges.

3.5.5 Comparison With Other Solutions

A key feature of our method is that it allows the robot to react to random human movements while the effect of these movements is indirectly reflected via a piece of cloth. This setting is similar to [103]. However, [103] assumes the 3D geometric mesh of cloth c is known without

any sensing error, which is not practical.

Our method falls into a broader category of visual-servoing methods, but most previous work in this area (such as [104]) has focused on navigation tasks and there is relatively little work on deformable body manipulation. [105] based their servoing engine on histogram features, which is similar to our use of HOW-features. However, they use direct optimization to minimize the cost function ($\mathbf{dist}(\mathcal{O}(\mathbf{c}), \mathcal{O}(\mathbf{c}^*))$), which is not possible in our case because our cost function is non-smooth in general.

Finally, our method is closely related to methods in [78, 79], which also use random-forest and store actions on the forest. However, our method is different from prior methods in two ways. First, our controller is continuous in its parameters, which means it can be trained using an imitation learning algorithm. Moreover, we use both feature extraction and controller parametrization in the imitation learning algorithm [1] so that both the feature extractor and the controller benefit from evolving training data.

To show the benefits of random-forest, we compare three different models of controllers: random-forest, linear regression, and neural network [1]. During each evaluation in the simulated environment, the human hands move to 10 random target positions v^{2*}, v^{3*}. In Table 3.3, we plot of the residual (Equation 3.8) of the tree methods against the number of imitation learning iterations. On the convergence of Algorithm 4, the random-forest-based controller outperforms the two other opponents, exhibiting a lower residual.

To implement the neural-network-based controller, we use Tensorflow, which is a neural network toolkit. The structure of the neural network is fully connected and consists of a hidden layer of 128 neurons. To implement the linear-regression-based controller, we use the apply the implementation from scikit-learn [106], which is a standard machine learning toolkit. We use the

Training Set Proportion	20%	40%	60%	80%	100%
Random-Forest	0.0154	0.0078	0.0046	0.0040	0.0038
Neural Network	0.0551	0.0469	0.0458	0.0459	0.0451
Linear Regression	$1.66e18$	$4.58e18$	$8.77e17$	$9.23e17$	$8.82e-5$

Table 3.3: **Comparison with Different Controllers:** Residual (Equation 3.8) of random-forest-based controller, neural-network-based controller [1], and linear regression controller, computed with different proportions of the training set. We use a dataset collected by an expert. The dataset contains 5702 points and we randomly select 20% of the data as the test dataset. The random-forest-based controller exhibits a lower residual. Linear regression increases residual on unseen data. A neural-network-based controller does not fit well when the size of the training set is limited.

standard parameters from the linear regression module.

3.5.6 Benefits of Random-Forest

There are many standard techniques for computing low-dimensional controlling parameters from high-dimensional perceptual data such as RGB images and depth maps. These include standard regression models and neural-network-based models. We evaluate the performance of our algorithm along with the others. The test involves measuring the residual of the manipulator as it moves towards the goal configuration based on the computed control parameters, as given by Equation 3.8.

We obtain best results in our benchmarks using a random-forest-based controller. Using the random-forest-based controller and the imitation framework requires fewer parameters to configure a task. Further, the computed control parameters are limited to the labels of the random-forest, which makes the controller robust to the unseen data. In practice, the random-forest-based imitation learning requires fewer computation resources which can enable the controller to be used in real-time applications. The performance is governed by the total number of iterations of the imitation learning. As the number of iterations of imitation learning grows, the residual

Equation 3.8 reduces. After reaching a certain iteration, the imitation learning contributes less to the performance enhancement. In other words, when the imitation learning framework converges, the overall performance of the controller is guaranteed.

3.6 Conclusion

We present a novel controller parametrization for cloth manipulation applications. In our parametrization, the optimal control action is defined on the leaf-nodes of a random-forest. Further, both the random-forest construction and controller optimization are integrated with the imitation learning algorithm and evolve with training data. We evaluate our method using a 3-task cloth manipulation application. The result shows that our method can seamlessly handle feature extraction and controller parametrization problems. In addition, our method is robust to random noises in human motion and observations. Moreover, our controller parametrization can robustly adapt to evolving training data and quickly reduce the mean action error for real-time human robot interaction. During our evaluations, the controller performs consistently well in terms of accomplishing the cloth manipulation tasks, including the ones with very large cloth deformations. In terms of comparing with the traditional regression-based controller, our approach can model complex relationships between high dimensional input data and configurations of the controller. Comparing with a neural-network-based controller, our approach can converge fast with limited input data, which makes it easier to adapt to unseen data.

Chapter 4: High-DoF Robot System Identification

4.1 Introduction

High-DOF robot systems are increasingly used for different applications. These systems include soft robots with deformable joints [107, 108], which have a high-dimensional configuration space, and articulated robots interacting with highly deformable objects like cloths [109, 110] or deformable environments like fluids [111, 112]. In these cases, the number of degrees-of-freedom (DOF $\mathbb{C}$, $N = |\mathbb{C}|$) can be more than 1000. As we try to satisfy dynamics constraints, the repeated evaluation of forward dynamics of these robots becomes a major bottleneck. For example, an elastically soft robot can be modeled using the finite-element method (FEM) [113], which discretizes the robot into thousands of points. However, each forward dynamics evaluation reduces to factorizing a large, sparse matrix, the complexity of which is $\boldsymbol{o}(N^{1.5})$ [114]. An articulated robot swimming in water can be modeled using the boundary element method (BEM) [111] by discretizing the fluid potential using thousands of patches on the robot's surface. In this case, each evaluation of the forward dynamics function involves inverting a large, dense matrix, the complexity of which is $\boldsymbol{O}(N^2\mathbf{log}(N))$ [115].

The high computational cost of forward dynamics becomes a major bottleneck for dynamics-constrained motion planning and feedback control algorithms. To compute a feasible motion plan or optimize a feedback controller, these algorithms typically evaluate the forward dynamics func-

tion hundreds of times per iteration. For example, a sampling-based planner [116] evaluates the feasibility of a sample using a forward dynamics simulator. An optimization-based planner [117] requires the Jacobian of the forward dynamics function to improve the motion plan during each iteration. Finally, a reinforcement learning algorithm [118] must perform a large number of forward dynamics evaluations to compute the policy gradient and improve a feedback controller.

Several methods have been proposed to reduce the number of forward dynamics evaluations. For sampling-based planners, the number of samples can be reduced by learning a prior sampling distribution centered on highly successful regions [119]. For optimization-based planners, the number of gradient evaluations can be reduced by using high-order convergent optimizers [120]. Moreover, many sampling-efficient algorithms [121] have been proposed to optimize feedback controllers. However, the number of forward dynamics evaluations is still on the level of thousands [120] or even millions [121].

Another method for improving the sampling efficiency is system identification [122, 123] that approximates the exact forward dynamics model with a surrogate model. A good surrogate model should accurately approximate the exact model while being computationally efficient [124]. These methods are mostly learning-based and require a training dataset. However, it is unclear whether the learned surrogate dynamics model is accurate enough for a given planning task. Indeed, [125] noticed that the learned dataset could not cover the subset of a configuration space required to accomplish the planning or control task.

Main Results: In this paper, we present a method of system identification for high-DOF robot systems. Our key observation is that, although the configuration space is high-dimensional, these robot systems are highly underactuated, with only a few controlled DOFs. The number of controlled DOFs typically corresponds to the number of actuators in the system and applications

tend to use a small number of actuators for lower cost [126, 127]. As a result, the state of the remaining DOFs can be formulated as a function of the few controlled DOFs, leading to a function $\mathbf{f} : \mathbb{C}_c \to \mathbb{C}$, where $\mathbb{C}_c$ is the space of the controlled DOFs. Since $\mathbb{C}_c$ is low-dimensional, sampling in $\mathbb{C}_c$ does not suffer from a curse-of-dimensionality. Therefore, our method accelerates the evaluations of $\mathbf{f}$ by precomputing and storing $\mathbf{f}$ on the vertices of a hierarchical grid. The hierarchical grid is a high-dimensional extension of the octree in 3D, where each parent node has $2^{|\mathbb{C}_c|}$ children. This hierarchical data structure has two desirable features. First, the error due to our approximate forward dynamics function can be bounded. Second, we construct the grid in an on-demand manner, where new sample points are inserted only when a motion planner requires more samples. As a result, the sampled dataset covers exactly the part of the configuration space required by the given motion planning task and the construction of the hierarchical grid is efficient.

We have evaluated the performance of our method on two benchmarks: a 1575-dimensional line-actuated soft robot arm and a 1415-dimensional underwater swimming robot. Our use of a hierarchical grid reduces the number of forward dynamics evaluations by one to two orders of magnitude and a plan can be computed within 2 hours on a desktop machine. We show that the error of our system identification method can be bounded and the algorithm converges to the exact solution of the dynamics constrained motion planning problem as the error bound tends to zero.

4.2 Related Work

In this section, we give a brief overview of prior work on high-DoF robot systems, motion planning and control with dynamics constraints, and system identification.

High-DoF Robot Systems are used in various applications such as soft robots [128]. A popular method for numerically modeling these soft robots is the finite-element method (FEM) [113, 129, 130]. Another example is a low-DoF articulated robot swimming in high-DoF fluid environments [111], where the boundary element method (BEM) [115] is used to model robot-fluid interactions. A third example is a robot arm manipulating a piece of cloth [2, 109, 110], where the state of the cloth is also discretized using FEM in [2]. Both FEM and BEM induce a forward dynamics function, the evaluation of which involves matrix factorization and inversion, where the matrix is of size $\boldsymbol{O}(N \times N)$. As a result, the complexity of each evaluation is $\boldsymbol{o}(N^{1.5})$ using FEM [114] and $\boldsymbol{O}(N^2\mathbf{log}(N))$ using BEM [115]. Prior work [131, 132] compromises accuracy for speed by using iterative linearization and fast matrix solvers. Instead, our method uses accurate FEM or BEM solvers but stores the solver results in a hierarchical grid for speedup.

Dynamics-Constrained Motion Planning algorithms can be optimization-based or sampling-based methods. Optimization-based methods find locally optimal motion plans [117, 120, 133–136] by iteratively minimizing an objective function under the dynamics constraints, where each iteration involves evaluating the forward dynamics function and its differentials. Sampling-based methods [116, 137] seek globally optimal motion plans, where the feasibility of each sampled motion plan is checked by calling the forward dynamics function. Differential dynamic programming [138] relies on forward dynamics evaluations to provide state and control differentials. Finally, reinforcement learning algorithms [118] requires a large number of forward dynamics

evaluations to compute the policy gradient. Our method can be combined with all these methods.

System Identification has been widely used to approximate the forward dynamics function. Most system identification methods are data-driven and approximate the system dynamics using non-parametric models such as the Gaussian mixture model [139], Gaussian process [122, 140], neural networks [141], and nearest-neighbor computations [142]. Our method based on the hierarchical grid is also non-parametric. In most prior learning methods, training data are collected before using the identified system for motion planning. Recently, system identification has been combined with reinforcement learning [143, 144] for more efficient data-sampling of low-DoF dynamics systems. However, these methods do not guarantee the accuracy of the resulting approximation. In contrast, our method provides guaranteed accuracy.

4.3 Problem Formulation

In this section, we introduce the formulation of high-DoF robot systems and forward dynamics evaluations. Next, we formulate the problem of dynamics-constrained motion planning for high-DoF robots.

4.3.1 High-DoF Robot System Dynamics

A high-DoF robot can be formulated as a dynamics system, the configuration space of which is denoted as $\mathbb{C}$. Each $\mathbf{x} \in \mathbb{C}$ uniquely determines the kinematic state of the robot and the high-DoF environment with which it is interacting. To compute the dynamics state of the robot, we need $\mathbf{x}$ and its time derivative $\dot{\mathbf{x}}$. Given the dynamics state of the robot, its behavior is

governed by the forward dynamics function:

$$\mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i) = (\mathbf{x}_{i+1}, \dot{\mathbf{x}}_{i+1}),$$

where the subscript denotes the timestep index, $\mathbf{x}_i$ is the kinematic state at time instance $i\Delta t$, and Δt is the timestep size. Finally, we denote $\mathbf{u}_i \in \mathbb{C}_c$ as the control input to the dynamics system (e.g., the joint torques for an articulated robot). In this work, we assume that the robot system is highly underactuated so that $|\mathbf{u}| \ll |\mathbf{x}|$. This assumption holds because the number of actuators in a robot is kept small to reduce manufacturing cost. For example, [108] proposed a soft robot octopus where each limb is controlled by only two air pumps. The forward dynamics function $\mathbf{g}$ is a result of discretizing the Euler-Lagrangian equation governing the dynamics of the robot. In this work, we consider two robot systems: an elastically soft robot arm and an articulated robot swimming in water.

4.3.2 Elastically Soft Robot

According to [113, 131, 132], the elastically soft robot is governed by the following partial differential equation (PDE):

$$\mathbf{M}\frac{\partial^2 \mathbf{x}}{\partial \mathbf{t}^2} = \mathbf{p}(\mathbf{x}) + \mathbf{c}(\mathbf{x}, \mathbf{u}), \tag{4.1}$$

where $\mathbf{p}(\mathbf{x})$ corresponds to the internal and external forces, $\mathbf{M}$ is the mass matrix, and $\mathbf{c}(\mathbf{x}, \mathbf{u})$ is the control force. This system is discretized by representing the soft robot as a tetrahedral mesh with $\mathbf{x}$ representing the vertex positions, as illustrated in Figure 4.1. Then the governing PDE

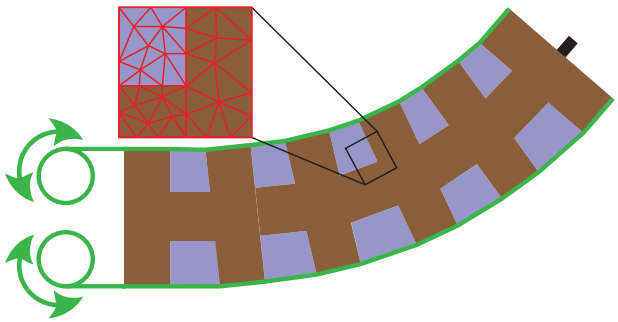


Figure 4.1: A 2D soft robot arm modeled using two materials (a stiffer material shown in brown and a softer material shown in blue), making it easy to deform. It is discretized by a tetrahedral mesh with thousands of vertices (red). However, the robot is controlled by two lines (green) attached to the left and right edges of the robot, so that $|\mathbf{u}| = 2$. The control command is the pulling force on each line (green circles).

(Equation 4.1) is discretized using an implicit-Euler time integrator as follows:

$$\mathbf{M}\frac{\mathbf{x}_{i+1} - 2\mathbf{x}_i + \mathbf{x}_{i-1}}{\Delta t^2} = \mathbf{p}(\mathbf{x}_{i+1}) + \mathbf{c}(\mathbf{x}_{i+1}, \mathbf{u}_i). \tag{4.2}$$

This function $\mathbf{g}$ is costly to evaluate because solving for $\mathbf{x}_{i+1}$ involves factorizing a large sparse matrix resulting from FEM discretization.

4.3.3 Underwater Swimming Robot System

Our second example, the articulated robot has a low-dimensional configuration space by itself. The configuration $\mathbf{x}$ consists of joint parameters. This robot is interacting with a fluid, so the combined fluid/robot configuration space is high-dimensional. According to [111, 112], the fluid's state can be simplified as a potential flow represented by the potential ϕ. This ϕ is discretized by sampling on each of the P vertices of the robot's surface mesh, as shown in Figure 4.2. The kinematic state of the coupled system is $(\mathbf{x}, \phi) \in \mathbb{C}$ and $N = |\mathbf{x}| + P$. However, ϕ can be computed from $\mathbf{x}$ and $\dot{\mathbf{x}}$ using the BEM method, denoted as $\phi(\mathbf{x}, \dot{\mathbf{x}})$. The governing dynamics equation in this case is:

$$\mathbf{M}(\mathbf{x})\frac{\partial^2 \mathbf{x}}{\partial \mathbf{t}^2} = \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}}) + \mathbf{J}(\mathbf{x})\mathbf{u} + \left[\frac{d}{dt}\frac{\partial}{\partial \dot{\mathbf{x}}} - \frac{\partial}{\partial \mathbf{x}}\right] \int \frac{1}{2}\phi(\mathbf{x}, \dot{\mathbf{x}})\frac{\partial \phi(\mathbf{x}, \dot{\mathbf{x}})}{\partial \mathbf{n}}, \tag{4.3}$$

where $\mathbf{M}$ is the generalized mass matrix, $\mathbf{C}$ is the centrifugal and Coriolis force, and $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix. Finally, the last term in Equation 4.3 is included to account for the fluid pressure forces, where the integral is over the surface of the robot and $\mathbf{n}$ is the outward surface normal. Time discretization of Equation 4.3 is performed using an explicit-Euler integrator, as follows:

$$\mathbf{M}(\mathbf{x}_i)\frac{\mathbf{x}_{i+1} - 2\mathbf{x}_i + \mathbf{x}_{i-1}}{\Delta t^2} = \mathbf{C}(\mathbf{x}_i, \dot{\mathbf{x}}_i) + \mathbf{J}(\mathbf{x}_i)\mathbf{u}_i + \left[\frac{d}{dt}\frac{\partial}{\partial \dot{\mathbf{x}}_i} - \frac{\partial}{\partial \mathbf{x}_i}\right] \int \frac{1}{2}\phi(\mathbf{x}_i, \dot{\mathbf{x}}_i)\frac{\partial \phi(\mathbf{x}_i, \dot{\mathbf{x}}_i)}{\partial \mathbf{n}}. \tag{4.4}$$

This function $\mathbf{g}$ is costly to evaluate because computing $\phi(\mathbf{x}_i, \dot{\mathbf{x}}_i)$ involves inverting the large, dense matrix that results from the BEM discretization.

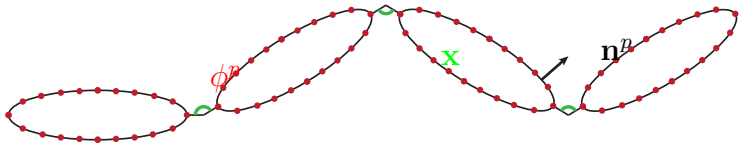


Figure 4.2: An articulated swimming robot consists of 4 rigid ellipses connected by hinge joints. The configuration space of the robot is low-dimensional, consisting of joint parameters (green). The fluid state is high-dimensional and represented by a potential function ϕ discretized on the vertices of the robot's surface mesh (the pth component of ϕ^p in red). The kinetic energy is computed as a surface integral (the pth surface normal $\mathbf{n}^p$ in the black arrow).

4.3.4 Dynamics-Constrained Motion Planning and Control

We mainly focus on the specific problem of dynamics-constrained motion planning and feedback control. In the case of motion planning, we are given a reward function $\mathcal{R}(\mathbf{x}_i, \mathbf{u}_i)$ and our goal is to find a series of control commands $\mathbf{u}_1, \cdots, \mathbf{u}_{K-1}$ that maximizes the cumulative reward over a trajectory: $\mathbf{x}_1, \cdots, \mathbf{x}_K$, where K is the planning horizon. This maximization is performed under dynamics constraints, i.e. $\mathbf{g}$ must hold for every timestep:

$$\underset{\mathbf{u}_1,\cdots,\mathbf{u}_{K-1}}{\mathbf{argmax}} \sum_{i=1}^{K} \mathcal{R}(\mathbf{x}_i, \mathbf{u}_i) \quad \mathbf{s.t.}\ \mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i) = (\mathbf{x}_{i+1}, \dot{\mathbf{x}}_{i+1}). \tag{4.5}$$

In the case of feedback control, our goal is still to compute the control commands, but the commands are generated by a feedback controller $\pi(\mathbf{x}_i, \mathbf{w}) = \mathbf{u}_i$, where $\mathbf{w}$ is the optimizable parameters of π:

$$\underset{\mathbf{w}}{\mathbf{argmax}} \sum_{i=1}^{K} \mathcal{R}(\mathbf{x}_i, \mathbf{u}_i)\ \mathbf{s.t.}\ \mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \pi(\mathbf{x}_i, \mathbf{w})) = (\mathbf{x}_{i+1}, \dot{\mathbf{x}}_{i+1}). \tag{4.6}$$

In both formulations, $\mathbf{g}$ must be evaluated tens of thousands of times to find the motion plan or controller parameters. In the next section, we propose a method to accelerate the evaluation of $\mathbf{g}$.

4.4 Hierarchical System Identification

Our method is based on the observation that high-DoF robot systems are highly underactuated. As a result, we can identify a novel function $\mathbf{f}$ that maps from the low-dimensional control input $\mathbf{u}$ to the high-dimensional kinematic state $\mathbf{x}$. When the evaluation of $\mathbf{f}$ is involved in the evaluation of $\mathbf{g}$, it causes a bottleneck. We approximate $\mathbf{f}$, instead of $\mathbf{g}$, using our hierarchical system identification method. We first show how to identify this function for different robot systems and then describe our approach to constructing the hierarchical grid.

4.4.1 Function $\mathbf{f}_s$ for an Elastically Soft Robot

We identify function $\mathbf{f}_s$ for an elastically soft robot (subscript s for short). We first consider a quasistatic procedure in which all the dynamics behaviors are discarded and only the kinematic behaviors are considered. In this case, Equation 4.2 becomes:

$$0 = \mathbf{p}(\mathbf{x}_{i+1}) + \mathbf{c}(\mathbf{x}_{i+1}, \mathbf{u}_i). \tag{4.7}$$

Equation 4.7 defines our function $\mathbf{f}_s(\mathbf{u}_i) \triangleq \mathbf{x}_{i+1}$ implicitly, whose domain has dimension $|\mathbf{u}|$ and range has dimension N. We can also compute $\mathbf{f}_s$ explicitly using Newton's method as shown in [132]. This computation is costly due to the inversion of a large, sparse matrix $\partial \mathbf{p}(\mathbf{x}_{i+1})/\partial \mathbf{x}_{i+1}$.

Given $\mathbf{f}_s$ that only models kinematics, we can also compute the dynamics function. To do this, we reinterpret $\mathbf{f}_s$ as a shape embedding function such that for each $\mathbf{x}$ there exists a latent parameter $\boldsymbol{\alpha}$ and $\mathbf{f}_s(\boldsymbol{\alpha}) = \mathbf{x}$. Note that although we used the forward kinematic function to define $\mathbf{f}_s$, $\mathbf{f}_s$ does not have a physical meaning when used as a shape embedding function and the input

$\boldsymbol{\alpha}$ is a dimensionless latent variable. This relationship can be plugged into Equation 4.1 to derive a projected dynamics system in the space of $\boldsymbol{\alpha}$ as:

$$\frac{\partial \mathbf{f}_s(\boldsymbol{\alpha}_{i+1})}{\partial \boldsymbol{\alpha}_{i+1}}^T \mathbf{M} \frac{\mathbf{f}_s(\boldsymbol{\alpha}_{i+1}) - 2\mathbf{f}_s(\boldsymbol{\alpha}_i) + \mathbf{f}_s(\boldsymbol{\alpha}_{i-1})}{\Delta t^2} = \frac{\partial \mathbf{f}_s(\boldsymbol{\alpha}_{i+1})}{\partial \boldsymbol{\alpha}_{i+1}}^T \left[\mathbf{p}(\mathbf{f}_s(\boldsymbol{\alpha}_{i+1})) + \mathbf{c}(\mathbf{f}_s(\boldsymbol{\alpha}_{i+1}), \mathbf{u}_i)\right] \tag{4.8}$$

where the left multiplication by $\partial \mathbf{f}_s(\boldsymbol{\alpha}_{i+1}) / \partial \boldsymbol{\alpha}_{i+1}{}^T$ is due to Galerkin projection (see [145] for more details). This technique is similar to reduced order method [146] but we use a special shape space defined by the forward kinematic function. Using Equation 4.8, we can compute $\boldsymbol{\alpha}_{i+1}$ from $\boldsymbol{\alpha}_i, \boldsymbol{\alpha}_{i-1}$ via Newton's method and then recover $\mathbf{x}_{i+1}$ using $\mathbf{x}_{i+1} = \mathbf{f}_s(\boldsymbol{\alpha}_{i+1})$. Computing $\boldsymbol{\alpha}_{i+1}$ is very efficient because Equation 4.7 represents a low-dimensional dynamics system. In summary, the computational bottleneck of $\mathbf{g}$ lies in the computation of $\mathbf{f}_s$, which is a mapping from the low-dimensional variables $\boldsymbol{\alpha}$ to the high-dimensional variable $\mathbf{x}$.

4.4.2 Function $\mathbf{f}_u$ for an Underwater Swimming Robot

We present our $\mathbf{f}_u$ for the underwater swimming robot in this section (subscript u for short). The kinematic state $\mathbf{x}$ is low-dimensional and the fluid potential $\phi(\mathbf{x}, \dot{\mathbf{x}})$ is high dimensional. We interpret this case as an underactuation because the state of the high-dimensional fluid changes due to the low-dimensional state of the articulated robot. The fluid potential is computed by the boundary condition that fluids and an articulated robot should have the same normal velocities at

every boundary point:

$$\left[\frac{\partial}{\partial \mathbf{n}^p}\right] \phi = \mathbf{n}^{pT} \mathbf{J}(\mathbf{x}) \dot{\mathbf{x}}, \tag{4.9}$$

where $\left[\frac{\partial}{\partial \mathbf{n}_i}\right]$ is a linear operator that is used to compute ϕ's directional derivative along the normal direction $\mathbf{n}^p$ at the pth surface sample (see Figure 4.2), which corresponds to the fluid's normal velocity. The right-hand side corresponds to the robot's normal velocity. Finally, we compute ϕ as:

$$\phi = \left[\frac{\partial}{\partial \mathbf{n}}\right]^{-1} \mathbf{n}^T \mathbf{J}(\mathbf{x}) \dot{\mathbf{x}},$$

where we assemble all the equations on all the P surface samples from Equation 4.9. Since there are a lot of surface sample points, $\left[\frac{\partial}{\partial \mathbf{n}}\right]$ is a large, dense $P \times P$ matrix and inverting it can be computationally cost. Therefore, we define:

$$\mathbf{f}_u(\mathbf{x}) \triangleq \left[\frac{\partial}{\partial \mathbf{n}}\right]^{-1} \mathbf{n}^T \mathbf{J}(\mathbf{x}), \tag{4.10}$$

which encodes the computationally costly part of the forward dynamics function $\mathbf{g}$. $\mathbf{f}_u$ has a domain of dimension $|\mathbf{x}|$ and a range of dimension $P \times |\mathbf{x}|$. Finally, $\mathbf{f}_u$ is a kinematics function like $\mathbf{f}_s$ because $\dot{\mathbf{x}}$ is excluded from $\mathbf{f}_u$. This choice reduces the dimension of the domain of $\mathbf{f}_u$.

4.4.3 Constructing the Hierarchical Grid

The evaluation of the forward dynamics function $\mathbf{g}$ requires the time-consuming evaluation of function $\mathbf{f}$ ($\mathbf{f}_s$ or $\mathbf{f}_u$). Moreover, certain motion planning algorithms require $\partial\mathbf{f}/\partial\mathbf{x}$ to solve Equation 4.5 or Equation 4.6. In this section, we develop an approach to approximate function $\mathbf{f}$ efficiently.

We accelerate $\mathbf{f}$ using a hierarchical grid-based structure, as shown in Figure 4.3 (a). Since the domain of $\mathbf{f}$ is low-dimensional, this formulation does not suffer from a-curse-of-dimensionality. To evaluate $\mathbf{f}(\mathbf{x})$ using a $|\mathbf{x}|$-dimensional grid with a grid size of Δx. We first identify the grid cell that contains $\mathbf{x}$. This grid cell has an interior:

$$\{\mathbf{y}|\forall i, \lfloor \mathbf{x}_i/\Delta x \rfloor = \lfloor \mathbf{y}_i/\Delta x \rfloor, \lceil \mathbf{x}_i/\Delta x \rceil = \lceil \mathbf{y}_i/\Delta x \rceil\}.$$

Each grid cell has $2^{|\mathbf{x}|}$ corner points $\mathbf{x}_c$ that satisfies $\lfloor \mathbf{x}_c/\Delta x \rfloor = \lceil \mathbf{x}_c/\Delta x \rceil$. For every corner point $\mathbf{x}_c$, we precompute $\mathbf{f}(\mathbf{x}_c)$ and $\partial\mathbf{f}/\partial\mathbf{x}_c$. Next, we can approximate $\mathbf{f}(\mathbf{x}), \partial\mathbf{f}/\partial\mathbf{x}$ at an arbitrary point inside the grid cell using a multivariate cubic spline interpolation [147]. Using a gird-based structure, we can improve the approximation accuracy by refining the grid and halving the grid size to $\Delta x/2$. After repeated refinements, a hierarchy of grids is constructed.

We first show how to build the grid at a fixed resolution. Evaluating $\mathbf{f}$ on every grid point is infeasible, but we do not know which grid points will be required before solving Equation 4.5. We therefore choose to build the grid on demand. When the motion planner requires the evaluation of $\mathbf{g}$ and $\partial\mathbf{g}/\partial\mathbf{x}, \dot{\mathbf{x}}$, the evaluation of $\mathbf{f}, \partial\mathbf{f}/\partial\mathbf{x}$ is also required. Next, we check each of the $2^{|\mathbf{x}|}$ corner points, $\mathbf{x}_c$. When $\mathbf{f}(\mathbf{x}_c)$ and $\partial\mathbf{f}/\partial\mathbf{x}_c$ have not been computed, we invoke the costly

procedure of computing $\mathbf{f}$ exactly (Equation 4.7 and Equation 4.10) and then store the results in our database. After all the corner points have been evaluated, we perform multivariate spline interpolation.

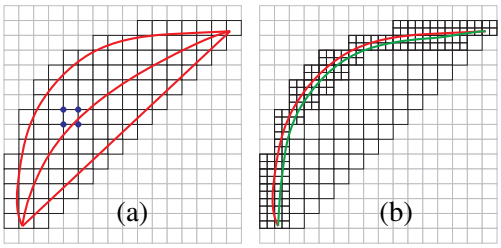


Figure 4.3: (a): We check and precompute $\mathbf{f}$ on $2^2 = 4$ corner points (blue). The initial guess of a motion plan is the straight red line and the converged plan is the curved line. (b): During the next execution, we refine the grid using the last motion plan (red) as the initial guess. The next execution updates the red curve to the green curve. The two curves are close and the number of corner points on the fine grid is limited.

Our on-demand scheme only constructs the grid at a fixed resolution or grid size. Our method allows the user to define a threshold η and continually refines the grid for $R = \lceil \mathbf{log}(\Delta x/\eta) \rceil$ times until $\Delta x/2^R < \eta$. Therefore, for each evaluation of $\mathbf{f}$ and $\partial \mathbf{f}/\partial \mathbf{x}$, we need to compute the appropriate resolution. Almost all motion planning [120] and control [118] algorithms start from an initial motion plan or controller parameters and update iteratively until convergence. We want to use coarser grids when the algorithm is far from convergence and finer grids when it is close to converging. However, measuring the convergence of an algorithm is difficult and we do not have a unified solution for different motion planning algorithms. As a result, we choose to interleave motion planning or control algorithms with grid refinement. Specifically, we execute the motion planning or control algorithms R times. During the rth execution of the algorithm, we use the result of the $(r-1)$th execution as the initial guess and use a grid resolution of $\Delta x/2^r$, as shown in Algorithm 5. Note that the only difference between the rth execution and $(r-1)$th execution is that the accuracy of the approximation for $\mathbf{f}$ is improved. Therefore, the rth execution will only disturb the solution slightly. This property will confine the solution space covered by the rth

execution and limit the number of new evaluations on the fine grid, as shown in Figure 4.3 (b). Finally, we show that under mild assumptions, the solution for Equation 4.5 and Equation 4.6 found using an approximate $\mathbf{f}$ will converge to that of the original problem with the exact $\mathbf{f}$ as the number of refinements $\mathbf{R} \to \infty$:

Lemma 4.4.1. *Assuming the functions $\mathcal{R}, \mathbf{g}$ are sufficiently smooth, the solution space of $\mathbf{x}$ is bounded, and the forward kinematic function is non-singular, then there exists a small enough Δt such that solutions $\mathbf{u}$ of Algorithm 5 will converge to a local minimum of Equation 4.5 or Equation 4.6 as $R \to \infty$, as long as the local minimum is strict (the Hessian of $\mathcal{R}$ has full rank).*

The proof of Lemma 4.4.1 is straightforward and we provide it in our appendix for completeness. in our extended report downloadable from [148].

Algorithm 5 Motion planner with system identification

if Solve motion planning problem **then**
  Input: Initial guess $\mathbb{P}^0 \leftarrow \mathbf{u}_1, \ldots, \mathbf{u}_{K-1}$
else
  Input: Initial guess $\mathbb{P}^0 \leftarrow \mathbf{w}^0$
end if
Input: Threshold of accuracy, η
▷ Run multiple times of motion planning or control
for $r = 0, 1, \cdots, R = \lceil \log(\Delta x/\eta) \rceil$ **do**
  Set grid resolution to $\Delta x/2^r$ {Refine the grid}
  ▷ Use previous solution as initial guess
  if Solve motion planning problem **then**
    Solve Equation 4.5 from initial guess $\mathbb{P}^r$
    $\mathbb{P}^{r+1} \leftarrow \mathbf{u}_1^*, \ldots, \mathbf{u}_{K-1}^*$
  else
    Solve Equation 4.6 from initial guess $\mathbb{P}^r$
    $\mathbb{P}^{r+1} \leftarrow \mathbf{w}^*$
  end if
end for
Return $\mathbb{P}^R$ =0

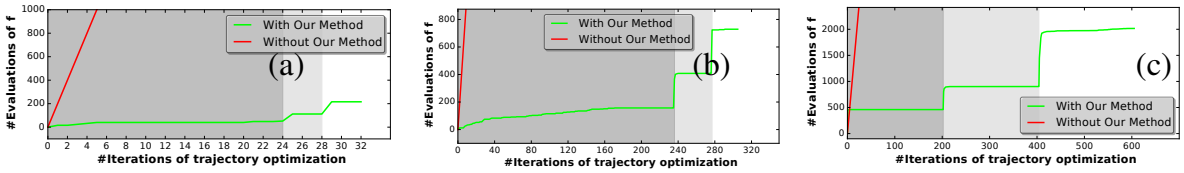


Figure 4.4: Number of evaluations of $\mathbf{f}$ plotted against the number of planning iterations with (red) and without (green) our method. (a): Optimization-based motion planning for the deformation soft arm. (b): Optimization-based motion planning for the underwater robot swimmer. (c): Reinforcement learning for the underwater robot swimmer.

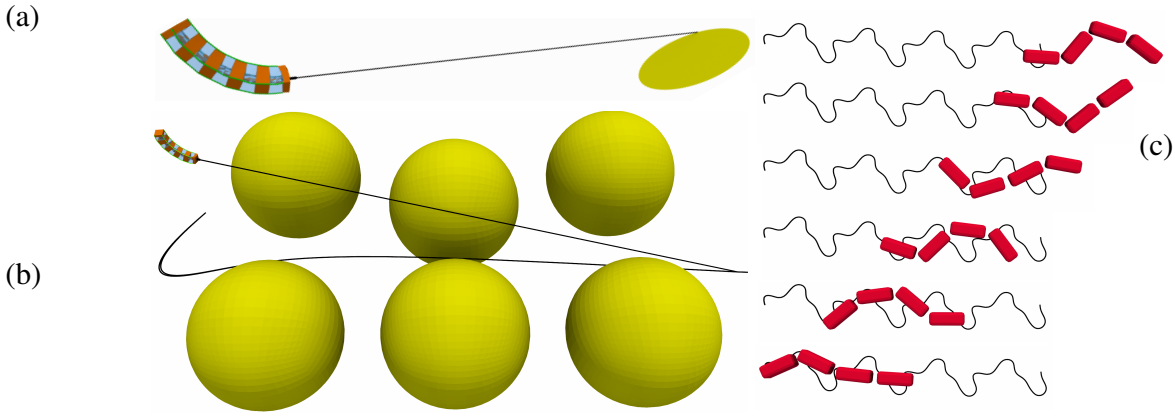


Figure 4.5: (a): A frame of a 3D soft robot arm attached with a laser cutter carving out a circle (yellow) on a metal surface. The arm is controlled by four lines attached to the four corners (green). (b): 3D soft robot arm steering the laser beam to avoid obstacles (yellow). (c): Several frames of a 3D underwater swimming robot moving forward. The robot is controlled by the 3-dimensional joint torques. The black line is the locus of the center-of-mass.

4.5 Implementation and Performance

We have evaluated our method on the 3D versions of the two robot systems described in Section 4.3. The computational cost of each substep of our algorithm is summarized in Table 4.1.

The 3D soft robot arm is controlled by four lines attached to four corners of the arm so that the control signal is 4-dimensional, $|\mathbf{u}| = 4$, and each evaluation of $\mathbf{f}_s$ requires $2^4 = 16$ grid corner point evaluations. To simulate its dynamics behavior, the soft arm is discretized using a tetrahedral mesh with 525 vertices so that $\mathbb{C}$ has $N = 3 \times 525 = 1575$ dimensions. To set up the hierarchical grid, we use an initial grid size of $\Delta x = 0.5$ and $\eta = 0.2$, so we will execute

the planning algorithm for $R = 3$ times. In this example, we simulate a laser cutter attached to the top of the soft arm and the goal of our motion planning is to have the laser cut out a circle on the metal surface, as shown in Figure 4.5 (a). We use an optimization-based motion planner [120] that solves Equation 4.5. The computed motion plan is a trajectory discretized into $K = 200$ timesteps and the trajectory is initialized to zero control forces at every timestep. In this case, if we evaluate $\mathbf{f}_s(\mathbf{x})$ exactly each time, then 200 evaluations of $\mathbf{f}_s$ are needed in each iteration of the optimization. To measure the rate of acceleration achieved by our method, we plot the number of exact $\mathbf{f}_s$ evaluations on grid corner points against the number of iterations of trajectory optimization with and without hierarchical system identification in Figure 4.4 (a). Our method requires 22 times fewer evaluations and the total computational time is 20 times faster. The total number of evaluations of function $\mathbf{f}_s$ for the elastically soft arm is 216 with system identification and is 4800 without system identification. We can also add various reward functions to accomplish different planning tasks, such as obstacle avoidance shown in Figure 4.5 (b).

Example	N	$\lvert\mathbb{C}_c\rvert$	$\mathbf{f}$ (s)	$\mathbf{g}$ (s)	$\tilde{\mathbf{g}}$ (s)	+HSI (s)	-HSI (s)	Speedup	#Corner	Err
Deformation Arm Trajectory Optimization	1575	4	1.5	1.51	0.01	5.5	305	20	216	$7e-6$
Swimming Robot Trajectory Optimization	1415	3	0.9	0.902	0.02	3.1	183	190	732	$2e-5$
Swimming Robot Reinforcement Learning	1415	3	0.9	0.902	0.02	42	16424	1590	1973	$5e-5$

Table 4.1: Summary of computational cost. From left to right: name of example, DoF of the robot system, dimension of $|\mathbb{C}_c|$, cost of evaluating $\mathbf{f}$, cost of evaluating $\mathbf{g}$, cost of evaluating $\mathbf{g}$ using system identification ($\tilde{\mathbf{g}}$), cost of each iteration of the planning algorithm with system identification, cost of each iteration without system identification (estimated), overall speedup, number of grid corner points evaluated, relative approximation error computed from: $\|\mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i) - \tilde{\mathbf{g}}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i)\| / \|\mathbf{g}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i)\|$.

For the 3D underwater robot swimmer, the robot has 3 hinge joints, so $\mathbf{x}$ is 3-dimensional

and $2^3 = 8$ grid corner points are needed to evaluate $\mathbf{f}_u$. The fluid potential ϕ is discretized on the robot surface with 1412 vertices, so $\mathbb{C}$ of the robot system has $N = 3 + 1412 = 1415$ dimensions. To set up the hierarchical grid, we use an initial grid size of $\Delta x = 0.3$ and $\eta = 0.1$, so we will execute the planning algorithm for $R = 3$ times. Our goal is to have the robot move forward like a fish, as shown in Figure 4.5 (c). We use two algorithms to plan the motions for this robot. The first algorithm is an optimization-based planner [120], which solves Equation 4.5 from an initial motion plan with zero control forces. The resulting plot of the exact number of $\mathbf{f}_u$ evaluations on grid corner points is shown in Figure 4.4 (b). Our method requires 205 times fewer evaluations and the estimated total computational time is 190 times faster. We have also tested our method with reinforcement learning [149], which solves Equation 4.6 and optimizes a feedback swimming controller parameterized by a neural network. The neural network is fully connected with one hidden layer and SmoothReLU activation function, which is initialized using random weights. This algorithm is also iterative and, in each iteration, [149] calls the function $\mathbf{g}$ 16384 times. The resulting plot of the number of exact function $\mathbf{f}_u$ evaluations during reinforcement learning with and without hierarchical system identification is given in Figure 4.4 (c). Our method requires 1638 times fewer evaluations and the total computational time is 1590 times faster.

4.5.1 Comparisons

Several prior works solve problems similar to those in our work. To control an elastically soft robot arm, [150] evaluates $\mathbf{g}$ and its differentials using finite difference in the space of control signals, $\mathbb{C}_c$. However, this method does not take dynamics into consideration and takes minutes to compute each motion plan in 2D workspaces. Other methods [151] only consider soft robots

with a very coarse FEM discretization and do not scale to high-DOF cases. To control an underwater swimming robot, [152] achieves real-time performance in terms of evaluating the forward dynamics function, but they used a simplified fluid drag model; we use the more accurate potential flow model [111] for the fluid. Finally, the key difference between our method and previous system identification methods such as [122, 139–142] is that we do not identify the entire forward dynamics function $\mathbf{g}$. Instead, we choose to identify a novel function $\mathbf{f}$ from $\mathbf{g}$ that encodes the computationally costly part of $\mathbf{g}$ and does not suffer from a-curse-of-dimensionality.

4.6 Conclusion

We present a hierarchical, grid-based data structure for performing system identification for high-DOF soft robots. Our key observation is that these robots are highly underactuated. We identify a low-dimension to high-dimension mapping function $\mathbf{f}$ and store that function in our grid to accelerate the computation. Since the domain is low-dimensional, we can precompute $\mathbf{f}$ on a grid without suffering from a curse-of-dimensionality. The construction is performed in an on-demand manner and the entire hierarchy construction is interleaved with the motion planning or control algorithms. These techniques effectively reduce the number of grid corner points to be evaluated and thus reduce the total running time by one to two orders of magnitude.

Chapter 5: High-DoF Policy Learning using Reinforcement Learning

5.1 Introduction

Digital painting systems are increasingly used by artists and content developers for various applications. One of the main goals has been to simulate popular or widely-used painting styles. With the development of non-photorealistic rendering techniques, including stroke-based rendering and painterly rendering [153, 154], specially-designed or hand-engineered methods can increasingly simulate the painting process by applying heuristics. In practice, these algorithms can generate compelling results, but it is difficult to extend them to new or unseen styles.

Over the last decade, there has been considerable interest in using machine learning methods for digital painting. These methods include image synthesis algorithms based on convolutional neural networks, including modeling the brush [155], generating brushstroke paintings [156], reconstructing paintings in specific styles [157], constructing stroke-based drawings [158], etc. Recent developments in generative adversarial networks [159] and variational autoencoders [160] have led to the development of image generation algorithms that can be applied to painting styles [161–165].

One of the goals is to develop an automatic or intelligent painting agent that can develop its painting skills by imitating reference paintings. In this paper, we focus on building an intelligent painting agent that can reproduce a reference image in an identical or transformed style with

Figure 5.1: *Results Generated by Our Painting Agent:* We use three paintings (top row) as the reference images to test our novel self-supervised learning algorithm. Our trained agent automatically generates the digitally painted image (bottom row) of the corresponding column in about 30 seconds without the need of a paired dataset of human artists.

a sequence of painting actions. Unlike methods that directly synthesize images bypassing the painting process, we focus on a more general and challenging problem of training a painting agent from scratch using reinforcement learning methods. [155, 156, 162, 166] also use reinforcement learning to solve the problem. All the methods encode goal states, which are usually defined as reference images, to the observations. This set-up is different from classic reinforcement learning tasks because, while the problem introduces an implicit objective to the policy network of reinforcement learning, the distribution of the reward in the action space can be very sparse and it makes training a reinforcement learning algorithm from scratch very difficult. To solve the problem, [155, 156, 162, 166] pre-train the policy network with a paired dataset consisting of images and corresponding actions defined in [155]. However, it is very expensive to collect such a paired dataset of human artists and we need to explore other unsupervised learning methods.

Main Results: We present a reinforcement learning-based algorithm (LPaintB) that incorporates self-supervised learning to train a painting agent on a limited number of reference images without paired datasets. Our approach is data-driven and can be generalized by expanding the

image datasets. Specifically, we adopt proximal policy optimization (PPO) [3] by encoding the current and goal states as observations and the continuous action space defined based on configurations of the paintbrush like length, orientation and brush size. The training component of our method only requires the reference paintings in the desired artistic style and does not require paired datasets collected by human artists. We use a self-supervised learning method to increase sampling efficiency. By replacing the goal state of an unsuccessful episode with its final state, we automatically generate a paired dataset with positive rewards. After applying the dataset to retrain the model using reinforcement learning, our approach can efficiently learn the optimal policy. The novel contributions of our work include:

- An approach for collecting supervised data for painting tasks by self-supervised learning.
- An adapted deep reinforcement learning network that can be trained using human expert data and self-supervised data, though we mostly rely on self-supervised data.
- An efficient rendering system that can automatically generate stroke-based paintings of desired resolutions by our trained painting agent.

We evaluate our approach by comparing our painting agent with prior painting agents that are trained from scratch by reinforcement learning [4]. We collect 1000 images with different color and patterns as the benchmark and compute L2 Loss between generated images and reference images. Our results show that self-supervised learning can efficiently collect paired data and can accelerate the training process. The training phase takes about 1 hour and the runtime algorithm takes about 30 seconds on a GTX 1080 GPU for high-resolution images.

5.2 Related Work

In this section, we give a brief overview of prior work on non-photorealistic rendering and the use of machine learning techniques for image synthesis.

5.2.1 Non-Photorealistic Rendering

Non-photorealistic rendering methods render a reference image as a combination of strokes by determining many properties like position, density, size, and color. To mimic the oil-painting process, Hertzmann hertzmann1998painterly renders the reference image into primitive strokes using gradient-based features. To simulate mosaic decorative tile effects, Hauser hausner2001simulating segments the reference image using Centroidal Voronoi diagrams. Many algorithms have been proposed for specific artistic styles, such as stipple drawings [167], pen-and-ink sketches [168] and oil paintings [169] [170]. The drawback of non photo-realistic rendering methods is the lack of generalizability to new or unseen styles. Moreover, they may require hand-tuning and need to be extended to other styles.

5.2.2 Visual Generative Algorithms

Hertzmann et al. hertzmann2001image introduce image analogies, a generative method based on a non-parametric texture model. Many recent approaches are based on CNNs and use large datasets of input-output training image pairs to learn the mapping function [171]. Inspired by the idea of variational autoencoders [160], Johnson et al. johnson2016perceptual introduce the concept of perceptual loss to model the style transfer between paired dataset. Zhu et al. zhu2017unpaired use generative adversarial networks to learn the mappings without paired train-

ing examples. These techniques have been used to generate natural images [164, 165], artistic images [172], and videos [173, 174]. Compared to previous visual generative methods, our approach can generate results of high resolution, can be applied to different painting media and is easy to extend to different painting media and artistic styles.

5.2.3 Image Synthesis Using Machine Learning

Many techniques have been proposed for image synthesis using machine learning. Hu et al. hu2018exposure present a framework using reinforcement learning and generative adversarial network to learn photo post-processing. Xie et al. xie2012,xie2015stroke,xie2013personal present a series of algorithms that simulate strokes using reinforcement learning and inverse reinforcement learning. These approaches learn a policy from either reward functions or expert demonstrations. In contrast to our algorithm, Xie et al. xie2012,xie2015stroke,xie2013personal focus on designing reward functions to generate orientational painting strokes. Moreover, their approach requires expert demonstrations for supervision. Ha et al. DBLP:journals/corr/HaE17 collect a large-scale dataset of simple sketches of common objects with corresponding recordings of painting actions. Based on this dataset, a recurrent neural network model is trained in a supervised manner to encode and re-synthesize the action sequences. Moreover, the trained model is shown to be capable of generating new sketches. Following [158], Zhou et al. zhou2018learning use reinforcement learning and imitation learning to reduce the amount of supervision needed to train such a sketch generation model. In contrast to prior methods, [4] operate in a continuous action space with higher dimensions applying PPO [3] reinforcement learning algorithm to train the agent from scratch. It can handle dense images with high resolutions. We use the same paint-

ing environment as [4] to demonstrate the benefits of our proposed learning algorithm. Although both algorithms do not need imitation data from human experts, self-supervised learning helps the reinforcement learning to converge to a better policy. Compared with prior visual generative methods, our painting agent can automatically generate results using a limited training dataset without paired dataset.

5.2.4 Reinforcement Learning

Reinforcement learning (RL) has achieved promising results recently in many problems, such as playing Atari games [23], the game of Go [24] and robot control [25]. A major focus of this effort has been to achieve improved time and data efficiency of the learning algorithms. Deep Q-Learning has been shown to be effective for tasks with discrete action spaces [23], and proximal policy optimization (PPO) [3] is currently regarded as one of the most effective for continuous action space tasks. Hindsight experience replay [26] enables off-policy reinforcement learning to sample efficiently from rewards which are sparse and binary. [26] can be seen as a sampling approach for off-policy algorithms, while we treat self-supervised learning and reinforcement learning as two components. Compared with [26], we present a practical approach to handle continuous space in a sparse reward setting and enhance the sampling efficiency by the self-supervised learning.

5.3 Self-Supervised Painting Agent

In this section, we introduce notations, formulate the problem and present our self-supervised learning algorithm for natural media painting.

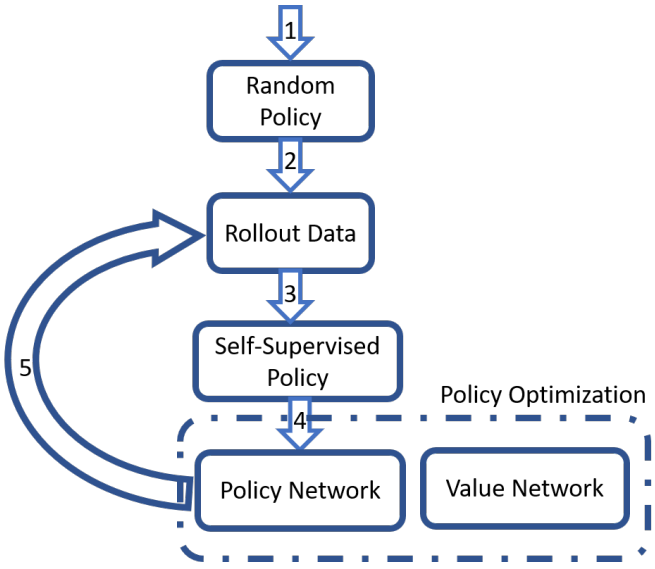


Figure 5.2: *Our Learning Algorithm:* We use self-supervised learning to generate paired dataset using a training dataset with reference images only and initialize the model for reinforcement learning. Then we feed the trained policy network to self-supervised learning to generate the paired datasets with positive rewards. (1) We initialize the policy network with random painting actions; (2) We rollout the policy by iteratively applying to the policy network to the painting environment to get paired data, followed by assigning the goal state s^* as $\hat{s^*}$ and changing the rewards of each step accordingly; (3) We retrain the policy with the supervision data to generate the self-supervised policy, and use the behavior cloning to initialize the policy network; (4) We apply policy optimization [3] and update the policy; (5) We rollout the updated policy and continue the iterative algorithm.

Symbol	Meaning
t	step index
t_q	time steps to compute accumulated rewards
s_t	current painting state of step t, canvas
s^*	target painting state, reference image
$\hat{s^*}$	reproduction of s^*
o_t	observation of step t
a_t	action of step t
r_t	reward of step t
q_t	accumulated reward of step t
γ	discount factor for computing the reward
π	painting policy, predict a by o
V_π	value function of the painting policy, predict r by o
$f(s)$	feature extraction of state s
$Render(a_t, s_t)$	render function, render action to s_t
$Obs(s^*, s_t)$	observation function, encode the current state and the target state
$Loss(s, s^*)$	loss function, measuring distance between state s and objective state s^*

Table 5.1: Notation and Symbols used in our Algorithm

5.3.1 Background

Self-supervised learning methods [175] are designed to enable learning without explicit supervision. The supervised signal for a pretext task is created automatically. It is a form of unsupervised learning where the data itself provides supervision. In its original formulation, this process is performed by withholding part of the information of the data and training the classification or regression function to predict it. The required task usually has a definition of the proxy loss so that it can be solved by self-supervised learning. There are a variety of applications for self-supervised learning in different areas such as audio-visual analysis [176], visual representation learning [177], image analysis [178], robotics [179] etc. In this paper, we use the term self-supervised learning to refer to the process of generating self-supervision data and using the data to initialize the policy network of the reinforcement learning framework.

5.3.2 Problem Formulation

Reproducing images with brush strokes can be formalized as finding a series of actions that minimizes the distance between the reference image and the current canvas in the desired feature space. Based on notations in Table 1, this can be expressed as minimizing the loss function:

$$\pi^* = \textbf{argmin}\ Loss(\hat{s^*}, s^*) \tag{5.1}$$

After we apply reinforcement learning to solve the problem by defining $Reward()$ function, we can get:

$$\pi^* = \textbf{argmax} \sum_{t}^{N} Reward(a_t, s_t) \tag{5.2}$$

5.3.3 Painting Agent

In this section, we present the technical details of our reinforcement learning-based painting agent.

5.3.3.1 Observation

Our observation function is defined as follows. First, we encode the objective state (reference image) with the painting canvas. Second, we extract both the global and the egocentric view of the state. As mentioned in [4, 162], the egocentric view can encode the current position of the agent and it provides details about the state. The global view can provide overall information about the state. $o(s_i)$ is defined as Eq.(6.1), given the patch size (h_o, w_o) and the position of the brush position(h_p, w_p).

$$o(s_i) = \left\{ s_i \left[h_p - \frac{h_o}{2} : h_p + \frac{h_o}{2}, w_p - \frac{w_o}{2} : w_p + \frac{w_o}{2} \right], s_i \right. \\ \left. s^* \left[h_p - \frac{h_o}{2} : h_p + \frac{h_o}{2}, w_p - \frac{w_o}{2} : w_p + \frac{w_o}{2} \right], s^* \right\}. \tag{5.3}$$

5.3.3.2 Action

The action is defined as a vector in continuous space with positional information and paint-brush configurations. $a_{=}\{dh, dw, width, color_R, color_G, color_B\} \in \mathbb{R}^6$. Each value is normalized to $[0, 1]$. The action is in a continuous space, which makes it possible to train the agent using policy gradient based reinforcement learning algorithms. The updated position of the paint brush after applying an action is computed by adding (dh, dw) to the coordinates of the paint brush $(p'_h, p'_w) = (p_h + dh, p_w + dw)$.

5.3.3.3 Loss Function

The loss function defines the distance between the current state and the objective state. It can guide how the agent reproduces the reference image. In practice, we test our algorithm with L_2 defined as Eq.(5.4), where s is the image of size $h \times w \times c$.

$$L_2(s, s^*) = \frac{\sum_{i=1}^{h} \sum_{j=1}^{w} \sum_{k=1}^{c} ||s_{ijk} - s^*_{ijk}||_2^2}{hwc} \tag{5.4}$$

For the self-supervised learning process, the loss function only affects reward computation. However, the reinforcement learning training process uses $\{\hat{s}^*\}$ as the reference images to train

the model and the loss function can affect the policy network.

5.3.3.4 Policy Network

To define the structure of the policy network, we consider the input as a concatenated patch of the reference image and canvas $82 \times 82 \times 3$ in egocentric view and global view, given the sample size of $41 \times 41 \times 3$. The first hidden layer convolves 64 8×8 filters with stride 4, the second convolves 64 4×4 filters with stride 2 and the third layer convolves 64 3×3 filters with stride 1. After that, it connects to a fully-connected layer with 512 neurons. All layers use ReLU activation function [180]. For the training process, we add the criteria $r > 0$ to expedite the training process.

5.3.3.5 Runtime Algorithm

After we trained a model using self-supervised learning and reinforcement learning, we can apply the model to generate reference images with different resolutions. First, we randomly sample a position from the canvas and draw a patch with size (h_o, w_o) and feed it to the policy network. Second, we iteratively predict actions $a_t = \pi(o_t)$ and render them by environment until the value network V_π returns a negative reward. Then we reset the environment by sampling another position from the canvas and keep the loop until $Loss(\hat{s^*}, s^*)$ less than $Thresh_{sim}$.

5.3.4 Behavior Cloning

Behavior cloning uses a paired dataset with observations and corresponding actions to train the policy to imitate an expert trajectory or behaviors. In our setup, the expert trajectory is en-

coded in the paired dataset $\{o^{(t)}, a^{(t)}\}$ which is related to step 4 in Figure 5.2. We use behavior cloning to initialize the policy network of reinforcement learning with the supervised policy trained by paired data. The paired dataset can be generated by a human expert or an optimal algorithm with global knowledge, which our painting agent does not have. Once the paired dataset $\{o^{(t)}, a^{(t)}\}$ is obtained, one solution is to apply supervised learning based on regression or classification to train the policy. The trained process can be represented using an optimization formulation as:

$$\pi^* = \mathbf{argmin} \sum_{t}^{N} ||\pi(o_t) - a_t|| \tag{5.5}$$

It is difficult to generate such an expert dataset for our painting application because of the large variation in the reference images and painting actions. However, we can generate a paired dataset by rolling out a policy defined as Eq.(5.6), which can be seen as iteratively applying predicted actions to the painting environment. For the painting problem, we can use the trained policy itself as the expert by introducing self-supervised learning.

5.3.5 Self-Supervised Learning

As we apply reinforcement learning to the painting problem, several new identities emerge as distinct from those of the classic controlling problems [3, 23, 181, 182]. We use the reference image as the objective and encode it in the observation of the environment defined in Eq.(6.1). As a result, the objective of the task Eq.(6.6) is not explicitly defined. Hence the rollout actions on different reference images $\{s^*\}$ can vary.

Through the reinforcement learning training process, the positive rewards in the high dimensional action space can be very sparse. In other words, only a small portion of actions sam-

pled by policy network have positive rewards. To change the reward distribution in the action space by increasing the probability of a positive reward, we propose using self-supervised learning. Our formulation uses the rollout of the policy as the paired data to train the policy network and retrains the model using reinforcement learning. Specifically, we replace the reference image s^* with the final rendering of the rollout of the policy function $\hat{s^*}$. Moreover, we use the updated observation $\{\hat{o_t}\}$ and the actions $\{a_t\}$ as the paired supervised training dataset. For the rollout process of the trained policy π, we have:

$$a_t = \pi(o_{t-1}), \tag{5.6}$$

$$s_t = Render(s_{t-1}, a_t), \tag{5.7}$$

$$r_t = \frac{Loss(s_{t-1}, s^*) - Loss(s_t, s^*)}{Loss(s_0, s^*)}, \tag{5.8}$$

$$o_t = Obs(s_t, s^*). \tag{5.9}$$

We can collect $\{o_{(t)}, a_{(t)}\}$ as the paired data. We denote the rendering of the final state as $\hat{s^*}$. The reward function is defined as the percentage improvement of the loss over the previous state.

Next, we modify o_t and r_t to a self-supervised representation as $\hat{o}_t$ and $\hat{r}_t$ as:

$$\hat{o}_t = Obs(s_t, \hat{s^*}), \tag{5.10}$$

$$\hat{r}_t = \frac{Loss(s_{t-1}, \hat{s^*}) - Loss(s_t, \hat{s^*})}{Loss(s_0, \hat{s^*})}, \tag{5.11}$$

$$\hat{q}_t = \sum_{k=t}^{t_s} \gamma^{k-t} r_k. \tag{5.12}$$

We use $\{\hat{o}_{(t)}, a_{(t)}, \hat{q}_{(t)}\}$ to train a self-supervised policy $\hat{\pi}$ and the value function $\hat{V_\pi}$. Algorithm 6

Algorithm 6 Self-Supervised Learning

Input: Set of objective states $\{s^{*(i)}\}$, its size is n_s
Output: Painting Policy π and its value function V_π

for $i = 1, \cdots, n_s$ **do**
  $t = 0$
  $s_0 = \text{INITIALIZE}()$
  // Rollout the policy and collect the paired data with positive reward
  while $r \geq 0$ **do**
    $t = t + 1$
    $a_t = \pi(Obs(s_{t-1}, s^{*(i)}))$
    $s_t = Render(s_{t-1}, a_t)$
    $r = (Loss(s_{t-1}, s^{*(i)}) - Loss(s_t, s^{*(i)})))/Loss(s_0, s^{*(i)})$
  end while
  // Build self-supervised learning dataset
  for $j = 0, \cdots, t-1$ **do**
    $\hat{o}_j = Obs(s_j, s_t)$
    $\hat{r}_j = (Loss(s_{j+1}, s_t) - Loss(s_j, s_t))/Loss(s_0, s_t)$
  end for
  // Compute cumulative rewards
  for $j = 0, \cdots, t-1$ **do**
    $\hat{q}_j = \sum_{k=j}^{t-1} \gamma^{k-j} \hat{r}_k$
  end for
  $\pi = \text{UPDATE}(\pi, \{\hat{o}_{(j)}, a_{(j)}\})$ // Initialize policy network for policy optimization
  $V_\pi = \text{UPDATE}(V_\pi, \{\hat{o}_{(j)}, \hat{q}_{(j)}, a_{(j)}\})$ // Initialize value network for policy optimization
end for
return π, V_π =0

highlights the learning process for self-supervised learning.

As we apply reinforcement learning to the painting problem, several new identities emerge as distinct from those of the classic controlling problems [3, 23, 181, 182]. We use the reference image as the objective and encode it in the observation of the environment defined in Eq.12 in [183]. As a result, the objective of the task Eq.(6.6) is not explicitly defined. Hence the rollout actions on different reference images $\{s^*\}$ can vary.

Through the reinforcement learning training process, the positive rewards in the high dimensional action space can be very sparse. In other words, only a small portion of actions sampled by policy network have positive rewards. To change the reward distribution in the action

space by increasing the probability of a positive reward, we propose using self-supervised learning. Our formulation uses the rollout of the policy as the paired data to train the policy network and retrains the model using reinforcement learning. Specifically, we replace the reference image s^* with the final rendering of the rollout of the policy function $\hat{s^*}$. Moreover, we use the updated observation $\{\hat{o_t}\}$ and the actions $\{a_t\}$ as the paired supervised training dataset. Eq.4-11 in [183] denotes the reward, action and states representation in both reinforcement learning and self-supervised learning settings. Alg.1 in [183] highlights the learning process for self-supervised learning.

5.4 Implementation

Our painting environment is similar to that in [4], which is a simplified simulated painting environment. Our system can execute painting actions with parameters describing stroke size, color and positional information and updates the canvas accordingly. We use a vectorized environment [**?**] for a parallel training process, as shown in Figure 5.3, to train our model.

5.4.1 Performance

In practice, we use a 16 core CPU and a GTX 1080 GPU to train the model with a vectorized environment of dimension 16. We use SSPE [4] as $Render(a, s)$ to accelerate the training process. The learned policy can also be transferred to other simulated painting media like MyPaint or WetBrush [**?**] to get different visual effects and styles.

Benchmarks	Benchmark1	Benchmark2
Reinforcement Learning Only	4.67	26.33
Self-supervised Learning Only	31.20	30.79
Our Combined Scheme	49.42	61.13

Table 5.2: *Comparison of Different Training Schemes:* We evaluate our method by comparing the average cumulative rewards on the test dataset.Self-supervised learning only refers to a policy that is trained with rollouts of a random policy by supervised learning, which reference image s^* is replaced as the final rendering s_t.

5.5 Results

In this section, we highlight the results and compare the performance with prior learning-based painting algorithms.

For the first experiment, we apply a critic condition to reward each step $r_t \geq 0$ for $t \geq 5$. Once the agent fails the condition, the environment will stop the rollout. We compare the cumulative reward $\sum_t r_t$ by feeding the same set of unseen images $\{s^{*(i)}\}$ to the environment. We use two benchmarks to test the generalization of the models. Benchmark1 is to reproduce an image $s^{*(i)}$ from a random image s_j. Benchmark2 is to reproduce an image $s^{*(i)}$ from a blank canvas. It can lead to a higher cumulative reward of Benchmark2 because the initial loss of Benchmark1 is less than Benchmark2. Each benchmark have 1000 $41 \times 41 \times 3$ patches. As shown in Table 2, our combined training scheme outperforms using only self-supervised learning or only reinforcement learning.

For the second experiment, we evaluate the performance on the high-resolution reference images. We compute the L_2 loss and cumulative rewards and compare our approach with [4]. We draw 1000 400×400 patches from 10 reference images to construct the benchmark. Moreover, we iteratively apply both the algorithms 1000 times to reproduce the reference images. We use the same training dataset with images to train the models. As shown in Table 3, our approach

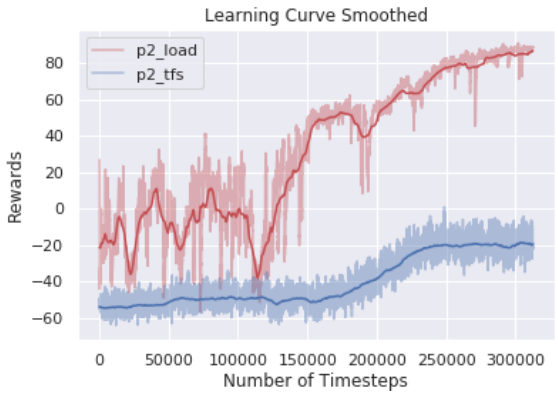


Figure 5.3: *Learning Curve Comparison* We evaluate our algorithm by plotting the learning curve of training from scratch (blue) and training with self-supervised learning (red). As shown in the figure, the method with self-supervision have better convergence and performance.

have a lower L_2, loss although both methods perform well in terms of cumulative rewards.

Approaches	Cumulative Rewards	L_2 Loss
PaintBot [4]	97.74	1920
LPaintB	98.25	1485

Table 5.3: *Comparison with Previous Work:* We evaluate our method by comparing the average cumulative reward and L_2 loss between final rendering and the reference image Eq.(5.4) on the test dataset.

5.6 Conclusion

We present a novel approach for stroke-based image reproduction using self-supervised learning and reinforcement learning. Our approach is based on a feedback loop with reinforcement learning and self-supervised learning. We modify and reuse the rollout data of the previously trained policy network and feed it into the reinforcement learning framework. We compare our method with both the model trained with only self-supervised learning and the model trained from scratch by reinforcement learning. The result shows that our combination of self-supervised and reinforcement learning can greatly improve efficiency of sampling and performance of the policy.

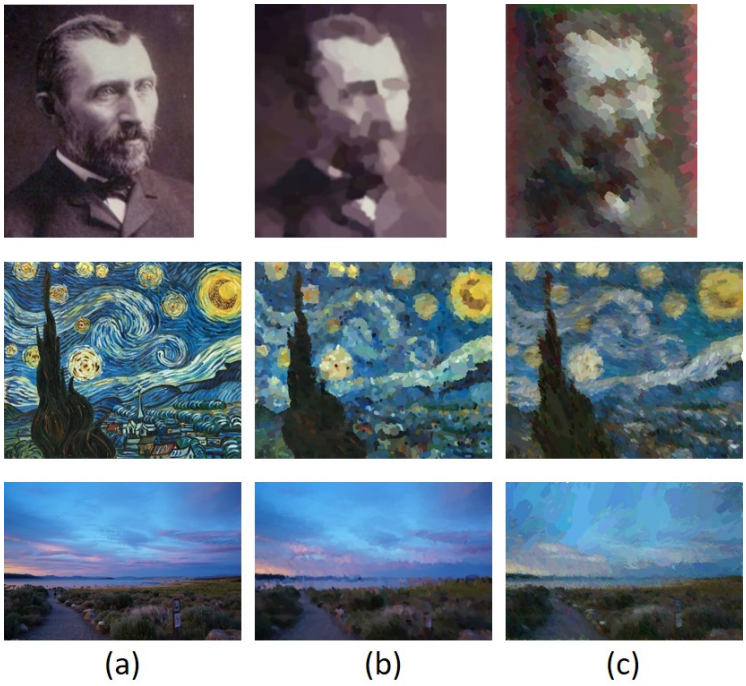


Figure 5.4: *Our results compared with [4]* We compare the final rendering result using the same scale of the reference image and the same amount of painting actions. (a) are the reference images. (b) are generated by our painting agent (c) are generated by the agent [4]. We demonstrate the benefits of self-supervised learning by reference images with different resolutions. The training dataset for both algorithms consists of 374 $41 \times 41 \times 3$ patches sampling from one painting.

Chapter 6: High-DoF Policy Transfer from Simulation to Reality

6.1 Introduction

Painting, a diverse and complex art form, spans various styles from watercolors to oil portraits. Efforts to simulate these styles have used non-photorealistic rendering techniques [153, 154] with some success. In generative image models conditioned on text, advancements have enabled diverse image synthesis [184, 185], yet transferring these techniques to real robots remains a challenge. Machine learning has been applied to painting, including brush modeling [155], stroke-based drawings [158], and artistic style emulation [161]. However, existing approaches often rely on manual engineering or lack stroke-based methods. Our work bridges these gaps in the realm of robotic sketching systems.

Distinct from these efforts, a category of robotic sketching systems has emerged, which often translate stroke-based methods directly into real robotic sketching [186]. Other studies, like those by Chen et al. [187], El et al. [188], and Vempati et al. [189], have concentrated on learning low-level manipulation policies to address challenges posed by uneven painting surfaces.

Our approach tackles a broader and more intricate challenge: training robotic painting policies using reinforcement learning techniques, behavior cloning, and stroke modeling. Our aim is to design a brand-new robotic sketching system capable of transferring the trained painting policy while incorporating sophisticated brush manipulation techniques.

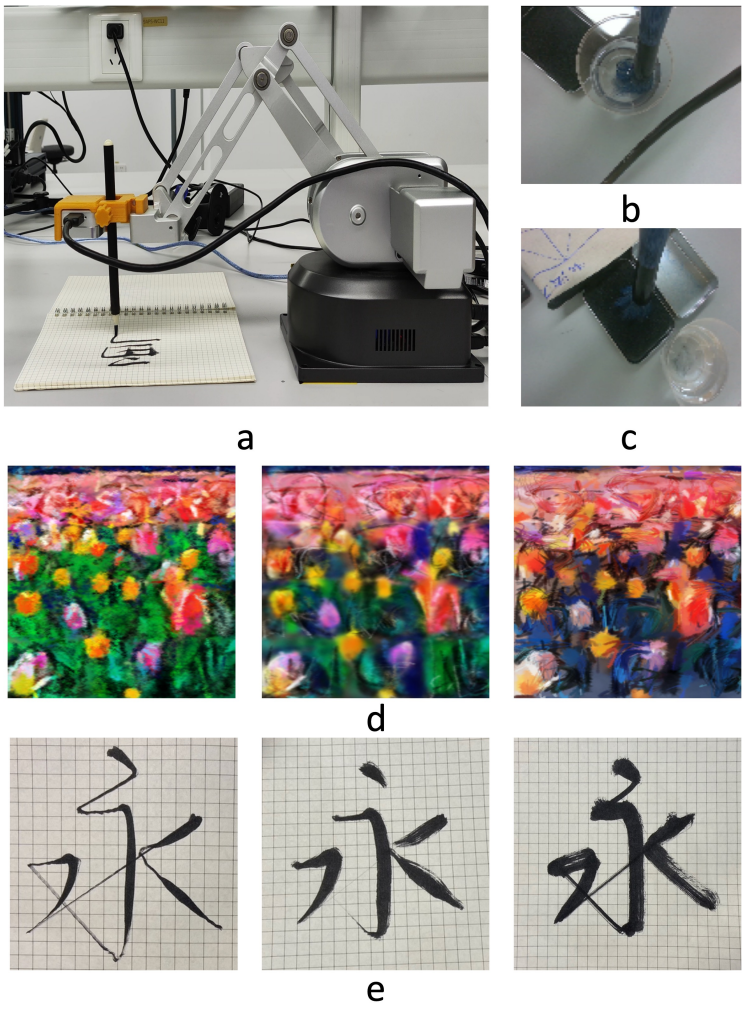


Figure 6.1: Our Robotic Sketching System: Developed through reinforcement learning and behavior cloning, it can recreate identical or transformed versions of a reference image in both simulated and real-world environments. Components used for real robot painting include (a) the Robot setup with Realsense D415, UltraArm robot, and paintbrush; (b) a Water pot; and (c) an Inkpot. The results generated by our robotic sketching system are illustrated in (d) simulated paintbrushes (from left to right: charcoal, pencil, and watercolor), employing 100 strokes; and in (e) real paintbrush, which utilizes three different stroke models, combining 5 long strokes with 68 small strokes.

Main Results: We introduce an innovative robotic sketching system that leverages a painting policy trained via reinforcement learning and behavior cloning for natural media painting. In the simulated environment, our model can acquire intricate painting policies through reinforcement learning, while behavior cloning allows us to fine-tune and adapt these policies. In the real-world context, we have developed a method for transferring the acquired policies while pre-

serving their artistic capabilities, including precise brush manipulation, thus enabling the creation of intricate and expressive sketches by the robotic system.The contributions of our work include:

- Introducing a novel reinforcement learning-based approach to model natural painting media in a simulated environment. Our approach demonstrates the versatility to learn with or without human supervision and excels in navigating continuous high-dimensional action spaces, enabling it to effectively handle large and intricately detailed reference images.

- Developing an adaptive sim-to-real methodology tailored for deformable brushes. This methodology includes estimating contact force and modeling strokes using a Gaussian model. It leverages behavior cloning to initialize policies for painting tasks, facilitating the seamless transfer of learned policies from simulation to reality.

- Creating a robotic sketching system comprising a robotic arm, egocentric view camera, and brush, mirroring the MyPaint virtual environment. This real-world setup empowers us to undertake complex artistic endeavors, including painting various subjects.

We conducted a rigorous evaluation of our results, encompassing a diverse set of reference images that span a wide range of artistic styles, as illustrated in Figure 6.1. Our virtual painting agent demonstrates its capability to generate high-resolution outputs tailored to various painting media. It excels in creating intricate, multi-stroke images, replicating complex patterns with precision. Simultaneously, our robot sketching system adeptly produces long, thin strokes with smooth variations in thickness and pressure, closely resembling the nuanced strokes of a human artist. These achievements are made possible through the combined techniques of behavior cloning and stroke modeling.

6.2 Related Work

6.2.1 Learning-based Drawing

Several related efforts have tackled similar challenges in this field. Xie et al. [155,156,166] introduced methods employing reinforcement learning and inverse reinforcement learning to simulate strokes. These techniques derive policies from either reward functions or expert demonstrations. However, Xie et al. [155, 156, 166] primarily concentrate on creating reward functions for generating oriental painting strokes, necessitating expert demonstrations for supervision. More recently, Ha et al. [158] gathered a substantial dataset comprising millions of basic object sketches, alongside recorded painting actions. They trained a recurrent neural network model in a supervised manner to encode and reproduce action sequences, showcasing the model's capacity to generate new sketches. Building on this, Zhou et al. [162] utilized a combination of reinforcement learning and imitation learning to reduce the supervision required for training similar sketch generation models. In contrast to [158, 162], our painting policies operate in a complex painting environment characterized by a continuous action space encompassing brush width and color. Our approach learns its policy network with minimal human supervision and readily adapts to real robotic systems.

6.2.2 Visual Generative Methods

In the realm of generative image models conditioned on text, significant advancements have enabled high-fidelity, diverse, and controllable image synthesis [184, 185, 190–194]. These improvements are attributed to large-scale, aligned image-text datasets [195] and scalable genera-

tive model architectures. Notably, diffusion models have excelled in learning high-quality image generators with a stable and scalable denoising objective [196–198].

Visual generative methods typically directly synthesize visual output in pixel spaces. Recent approaches employ CNNs and large datasets for learning mapping functions [171]. Some use variational autoencoders [160] to implement style transfer [199]. Others apply generative adversarial networks (GANs) [159], such as Cycle-Consistent Adversarial Networks [161], which excel at generating natural and artistic images [164, 165, 172], as well as videos [173, 174].

However, these generative methods may fall short in achieving high-resolution results. Conversely, our stroke-based approach generates paintbrush trajectories, adaptable to diverse synthetic and real painting environments using robotic arms.

6.2.3 Robotic Sketching Systems

In the development of robotic sketching systems, various approaches have been investigated. In significant work by Lee et al. [186], a hierarchical reinforcement learning (RL) model was proposed for painting tasks, where a high-level controller learns the painting policy and a low-level policy to control the robot arm. This is the most relevant previous work to our research, as both aim to address robotic sketching using RL. The key difference lies in our implementation of a more complex virtual and real painting environment, capable of handling various intricate painting media such as oil painting, watercolor, and ink, as well as deformable paintbrushes through a larger action space and a behavior cloning framework. Consequently, we achieve more intricate and detailed results.

Other studies, such as those by Chen et al. [187], El et al. [188], and Vempati et al. [189],

Symbol	Meaning
t	step index
s_t	current painting state of step t, canvas
s^*	target painting state, reference image
$\hat{s^*}$	reproduction of s^*
o_t	observation of step t
a_t	action of step t, $a_t = [\alpha_t, l_t, w_t, c_t]$
r_t	reward of step t
q_t	accumulated reward of step t
γ	discount factor for computing the reward
p_t	position of the paintbrush of step t
π	painting policy, predict a by o
V_π	value function of the painting policy, predict r by o
$R(a_t, s_t)$	render function, render action to s_t
$O(s^*, s_t)$	observation function, encode the current state and the target state
$L(s, s^*)$	loss function, measuring distance between state s and objective state s^*
α_t	angle of action a_t
l_t	length of action a_t
w_t	stroke width of action a_t
c_t	color descriptor of action a_t

Table 6.1: Notation Summary

have focused on learning low-level manipulation policies to tackle challenges presented by uneven painting surfaces. A distinctive feature of our approach, compared to these studies, is that our method does not require explicit environmental modeling. Consequently, our algorithm exhibits broader applicability in real-world scenarios and a wider range of painting tasks, marking a significant contribution to the field of robotic painting algorithms.

6.3 Training a Painting Policy

In this section, we delve into the technical intricacies of our reinforcement learning-based painting policy. We start by introducing the core components of reinforcement learning, including the action space, observation, reward, and policy network. We then elaborate on our training

and runtime algorithms, discussing techniques aimed at improving learning efficiency, such as curriculum learning. Additional technical details can be found in [200].

6.3.0.1 Action Space

To capture the essence of painting behavior, we represent actions using stroke properties, including angle, length, size, and color. Specifically, we define the action as a 6-dimensional vector, $a_t = [\alpha_t, l_t, w_t, c_{rt}, c_{gt}, c_{bt}] \in \mathbb{R}^6$, with each value normalized to $[0, 1]$. The action space is continuous, enabling us to employ policy gradient-based reinforcement learning algorithms. Notably, when $w = 0$, the brush moves above the canvas without applying paint.

6.3.0.2 Observation

Our approach extends the observation o_t to include both the current state s_t and the reference image s^*. We use an egocentric observation strategy, centering the paintbrush on the canvas. This simplifies the action space, eliminates the need for a replay buffer, and enables training in a continuous action space and large state space. The state observation o_t is defined in Equation 6.1, where (h_p, w_p) represents the paintbrush position, and (h_o, w_o) denote the egocentric window size.

$$o_t = \left\{ s_t \left[h_p - \frac{h_o}{2} : h_p + \frac{h_o}{2}, w_p - \frac{w_o}{2} : w_p + \frac{w_o}{2} \right], \right. \\ \left. s^* \left[h_p - \frac{h_o}{2} : h_p + \frac{h_o}{2}, w_p - \frac{w_o}{2} : w_p + \frac{w_o}{2} \right] \right\}. \tag{6.1}$$

We illustrate our rollout algorithm in Algorithm 8.

Algorithm 7 Rollout Algorithm

Input: Reference image s^* with size (h_{s^*}, w_{s^*}), the learned painting policy π with observation size (h_o, w_o)
Output: Final rendering $\hat{s^*}$

while $||I - I^*|| > Thresh_{sim}$ **do**
  $h = \text{rand}(h_{s^*})$ // Sample a 2-dimensional point within the image to start the stroke
  $w = \text{rand}(w_{s^*})$
  $o = s[h - \frac{h_o}{2} : h + \frac{h_o}{2}, w - \frac{w_o}{2} : w + \frac{w_o}{2}]$ // Get observation
  $r = 1$ // Initialize the predicted reward
  while $r > 0$ **do**
    $a = \pi(o)$ // Predict the painting action
    $r = V_\pi(o)$ // Predict the expected reward
    $s = R(s, a)$ // Render the action
    $h = h + l \times \cos(\alpha)$ // Update the stroke position
    $w = w + w \times \sin(\alpha)$
    $o = s[h - \frac{h_o}{2} : h + \frac{h_o}{2}, w - \frac{w_o}{2} : w + \frac{w_o}{2}]$ // Update the observation
  end while
end while
return s =0

6.3.0.3 Reward

In our setup, the reward for each action is determined by the difference between the canvas and the reference image. A loss function is employed to calculate the action's reward during each reinforcement learning iteration. To incentivize the painting agent to match the color and shape of the reference image precisely rather than aiming for an average color, we slightly modify the L_2 loss into $L_{\frac{1}{2}}$,

$$L_{\frac{1}{2}}(s, s^*) = \frac{\sum_{i=1}^{h} \sum_{j=1}^{w} \sum_{k=1}^{c} |s_{ijk} - s^*_{ijk}|^{\frac{1}{2}}}{hwc}, \tag{6.2}$$

where the image s and the reference image s^* are matrices with dimensions $h \times w \times c$. Here, w and h denote the width and height of the image, while c represents the number of color channels.

After defining the loss between I and I^{ref}, we normalize r_t using Eq. 6.3, such that $r_t \in (-\infty, 1]$.

$$r_t = \frac{L(s_{t-1}, s^*) - L(s_t, s^*)}{L(s_0, s^*)} \tag{6.3}$$

6.3.0.4 Policy Network

The first hidden layer applies convolution with 64 8×8 filters and a stride of 4. The second layer employs convolution with 64 4×4 filters and a stride of 2, followed by the third layer using convolution with 64 3×3 filters and a stride of 1. Subsequently, the network connects to a fully-connected layer comprising 512 neurons. All layers employ the ReLU activation function [180].

6.3.0.5 Curriculum Learning

In the context of a continuous action space $a \in \mathbb{R}^6$, we face challenges with growing sampling space and noise from policy gradient-based reinforcement learning. To address this efficiently, we adopt curriculum learning, progressively increasing sampled trajectories during training. In RL, the optimal policy π^* maximizes the expected long-term reward q_t, which accumulates rewards r_t over a time horizon $t_{\max}$ with a discount factor $\gamma \in \mathbb{R}$:

$$q_t = \sum_{t=1}^{t_{\max}} r_t \gamma^t. \tag{6.4}$$

For painting policies, numerous goal configurations are sparsely distributed in a high-dimensional space, challenging the agent's convergence. We adapt the horizon parameter $t_{\max}$ by introducing a reward threshold r_{thresh}, gradually increasing it during training as:

$$\hat{t}_{\max} = \underset{i}{\mathbf{argmin}} \, (r_i > r_{\text{thresh}}). \tag{6.5}$$

This redefined horizon parameter allows the policy gradient algorithm to converge effectively when dealing with complex goal configurations, encouraging the policy to prioritize rewards within limited time steps, reducing exploration space.

6.4 Sim-to-Real Brush Manipulation

In this section, we explain our sim2real transfer methods from the painting policy in Section 6.3 with the aim of seamlessly applying the policy to real-world robotic drawing tasks for precise brush manipulation and stroke control. Accurate pressure estimation is crucial for controlling stroke shapes and interactions with various painting media like ink and water. Rather than using force sensors, we employ advanced modeling and image analysis techniques for pressure estimation, making it applicable in situations where force sensing is impractical. Our practical experiments combine end-effector image capture to determine optimal pressure ranges with stroke image sampling for precise mapping. We deconstruct our acquired policy into high-level and low-level components. The high-level policy is trained using behavior cloning, standardizing stroke order, especially for handwriting. In contrast, the low-level policy is developed via efficient sampling-based reinforcement learning, translating the original RL policy into real-world actions.

6.4.1 Contact Force Estimation

Accurately estimating the contact force between the pen tip and the painting media is a crucial aspect of robotic brush manipulation. However, precise force sensors are often unavailable. Therefore, we employ image analysis methods to infer pressure values, as described in Alg. **??**.

6.4.1.1 Observation of Stroke Images

This approach involves indirectly observing environmental changes, specifically the stroke images on the paper, to infer variations in pressure. It is an intuitive method where we record the shape of strokes and the configuration of the robotic arm. We can then interpolate to obtain the desired stroke characteristics.

However, finding a suitable arm configuration is not straightforward. Similar to training reinforcement learning in simulation, this method requires extensive sampling, with many instances yielding no positive rewards due to the limited deformation range of the brush.

6.4.1.2 Observation of End-Effector Images

In contrast to observing stroke images, this method offers a more direct approach. It involves capturing the shape changes of the flexible end effector.

While this method may be susceptible to image noise, it provides valuable information about the pressure limit of the flexible object. We utilize linear fitting to identify the point at which deformation no longer occurs, treating it as the pressure limit.

Algorithm 8 Rollout Algorithm

Input: Reference image s^* with size (h_{s^*}, w_{s^*}), the learned painting policy π with observation size (h_o, w_o)
Output: Final rendering $\hat{s^*}$

while $||I - I^*|| > Thresh_{sim}$ **do**
  $h = \text{rand}(h_{s^*})$ // Sample a 2-dimensional point within the image to start the stroke
  $w = \text{rand}(w_{s^*})$
  $o = s[h - \frac{h_o}{2} : h + \frac{h_o}{2}, w - \frac{w_o}{2} : w + \frac{w_o}{2}]$ // Get observation
  $r = 1$ // Initialize the predicted reward
  while $r > 0$ **do**
    $a = \pi(o)$ // Predict the painting action
    $r = V_\pi(o)$ // Predict the expected reward
    $s = R(s, a)$ // Render the action
    $h = h + l \times \cos(\alpha)$ // Update the stroke position
    $w = w + w \times \sin(\alpha)$
    $o = s[h - \frac{h_o}{2} : h + \frac{h_o}{2}, w - \frac{w_o}{2} : w + \frac{w_o}{2}]$ // Update the observation
  end while
end while
return s =0

6.4.2 Mapping Actions from Simulation to Reality

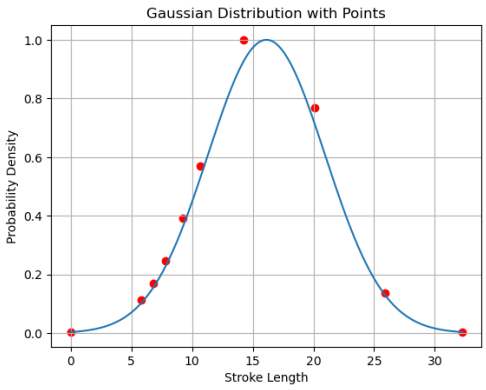
In Section 6.3, we defined actions in a simulated environment, which may differ from the actions required in the real-world environment. Therefore, we need to map robot actions from the simulated environment's action space to the robot's configuration space in the real world.

The first challenge is that the painting plane in the simulated environment differs from the real robot environment. Therefore, we need to find a 2D plane in the 3D configuration space to serve as the painting space. The action mapping formula is computed similarly to the camera's extrinsic calibration.

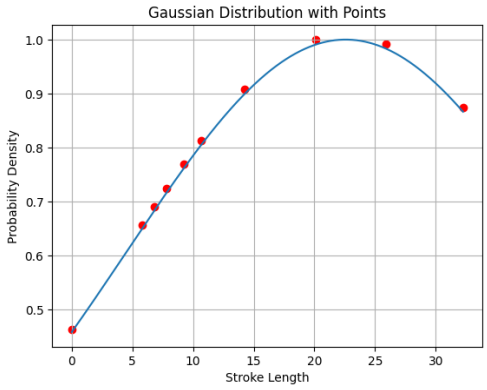
The second challenge arises because certain actions cannot be directly translated into robot movements but still have a limited visual effect. These include:

1. Stroke thickness, which can only be adjusted by changing the brush's contact force.

2. Color, which, in our setup, is limited to monochrome. Color changes are achieved through



(a) Mean: 0.5, Standard Deviation: 0.3.



(b) Mean: 0.7, Standard Deviation: 0.8.

Figure 6.2: *Effect of Gaussian Stroke Model on Stylization:* We model a long stroke composed of segments using a Gaussian distribution. These correspond to the first column in Fig. 6.1, where variations in artistic style are achieved by adjusting the Gaussian parameters.

interactions with the environment, such as dipping in ink, water, or interacting with a sponge.

3. Tilt, which our 3-DoF robot cannot directly achieve due to limited kinematics.

To approximate these effects, we employ the following methods:

6.4.2.1 Gaussian Modeling of Strokes

To achieve an artistic font treatment, we emulate the stroke characteristics of human artists. This is accomplished through Gaussian modeling for each stroke, which captures the distribution of the stroke's centroid and pressure. This approach empowers us to create artistic fonts with diverse styles. By fine-tuning these parameters, we can generate various types and styles of strokes, leading to font diversity as illustrated in Fig. 6.2. For all straight-line strokes, we employ this method to merge them into long, thin strokes, enhancing the overall line's smoothness and natural appearance.

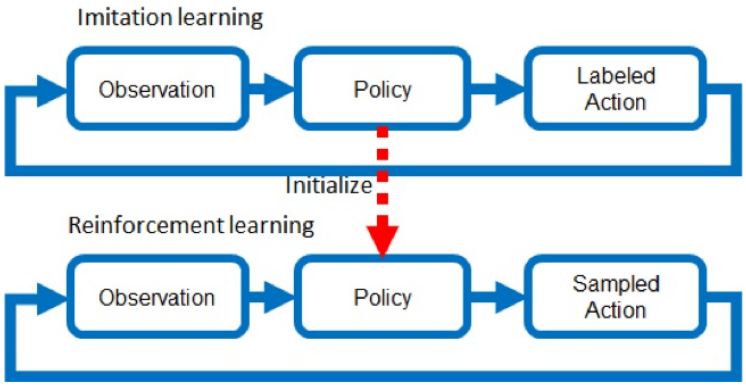


Figure 6.3: *Behavior Cloning for Policy Initialization:* We utilize a behavior cloning algorithm to train the policy, extending the action space to initialize the reinforcement learning (RL) policy within a real environment setup. The action space used in behavior cloning is a subspace of the RL action space and includes direction and on/off canvas actions. This initialization process bridges the gap between behavior cloning and RL, facilitating effective policy learning in the real environment.

6.4.2.2 2D to 3D Action Projection

To match the actions from the simulated environment to the real robot's configuration space, we need to project 2D actions into a 3D configuration space. This projection can be defined using the following equation, which is similar to a camera's extrinsic calibration projection:

$$\begin{bmatrix} x_{\text{robot}} \\ y_{\text{robot}} \\ z_{\text{robot}} \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{painting}} \\ y_{\text{painting}} \\ 1 \end{bmatrix}$$

Here, x_{robot}, y_{robot} and z_{robot} represent the robot's coordinates. x_{painting} and y_{painting} are the desired painting coordinates in 2D space. The transformation matrix $\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$ maps the 2D painting coordinates to the 3D robot configuration, allowing us to generate actions that correspond to the desired painting locations and orientations in the real world.

6.4.3 Behavior Cloning

Behavior cloning leverages a paired dataset comprising observations and corresponding actions to train a policy to mimic expert trajectories or behaviors. In our context, the expert trajectory is encoded in the paired dataset $\{o_{(t)}, a_{(t)}\}$. We employ behavior cloning to initialize the policy network for reinforcement learning, using the supervised policy trained with the paired data. The paired dataset can be generated by a human expert or an optimal algorithm with global knowledge, which our painting agent lacks. Once we obtain the paired dataset $\{o_{(t)}, a_{(t)}\}$, one common approach is to apply supervised learning based on regression or classification to train the policy. The training process can be formulated as an optimization problem:

$$\pi^* = \underset{\sum}{\mathbf{argmin}} \, {}^{N}_{t} ||\pi(o_t) - a_t||. \tag{6.6}$$

Generating an expert dataset for our painting application can be challenging due to the significant variation in reference images and painting actions. However, we can create a paired dataset by rolling out a policy during the RL training process. Additionally, there are existing datasets like KanjiVG and Google's Quick, Draw! that provide paired supervised data [201,202].

When we train behavior cloning with real data, the painting policy tends to produce long, thin strokes rather than simple multiple strokes. This method also enables us to achieve a natural brushwork effect, as demonstrated in Fig. 6.1(e).

6.5 Experiment

6.5.1 Robotic Sketching System Setup

In our simulated painting setup, we have created an environment that allows the painting agent to explore a high-dimensional action space and observation space based on MyPaint [203]. This setup aligns with the description provided in Sec. 6.3.

For the real brush manipulation experiment, we implement our approach using an UltraArm, which features 3 DoFs for movement as shown in Fig. 6.1. The primary experimental setup includes a water pot and foam, allowing the robot to manipulate a paintbrush by absorbing water, squeezing it, or using the object to reshape it. This setup serves to demonstrate that our method can effectively learn the complexity of high DoF end-effector manipulation tasks in a practical and realistic scenario.

By incorporating the water pot and foam into the experimental setup, we introduce additional challenges that the robot must learn to overcome. These include controlling the amount of water absorbed by the paintbrush, adjusting the pressure applied when squeezing or reshaping the brush, and maintaining a stable grip on the brush throughout the manipulation process. These added complexities showcase the adaptability and effectiveness of our approach in handling diverse manipulation tasks involving deformable materials and intricate interactions with the environment.

Figure 6.4: *Illustration of Stroke Order*: demonstration of the stroke order generated by our behavior cloning algorithm (Columns 1 and 3) and the final sketches generated by our robot sketching system (Columns 2 and 4).

6.5.2 Data Preparation

In the scope of our real-robot experiments, we selected the KanjiVG dataset [201] for our training endeavors. This dataset, rich in its depth, provides detailed stroke information for approximately 2,000 distinct characters. This dataset, having been meticulously collated from human participants, establishes itself as a premier choice when leveraging behavior cloning in the domain of robotic calligraphy.

Within the framework of our reinforcement learning (RL) strategy, we leaned on the acclaimed CelebA dataset [204] to facilitate the training of our painting agent. It's important to note that our rollout algorithm was architectured employing MyPaint [203], a decision made to ensure the results seamlessly mirror the characteristics of natural media. The nuances of the painting model are distilled implicitly, rooted in the foundational knowledge embedded in the environment model. The versatility and robustness of our algorithm are showcased in Fig. 6.1.

6.5.3 Evaluation

We demonstrated the advantages of our approach by computing performance and comparing visual effects. We designed two experiments to evaluate the performance of our algorithms.

For the first experiment, we computed the learning curve of the baseline model and the model with curriculum learning (Sec. 6.3.0.5), as shown in Fig. 6.5. Both models converged within $78,000$ episodes. The y-axis denotes the average rewards of the trained model in a validation dataset, and the x-axis denotes the training episodes. As the training process proceeded, the average rewards grew, showing that curriculum learning can improve the reinforcement learning to converge to a better policy.

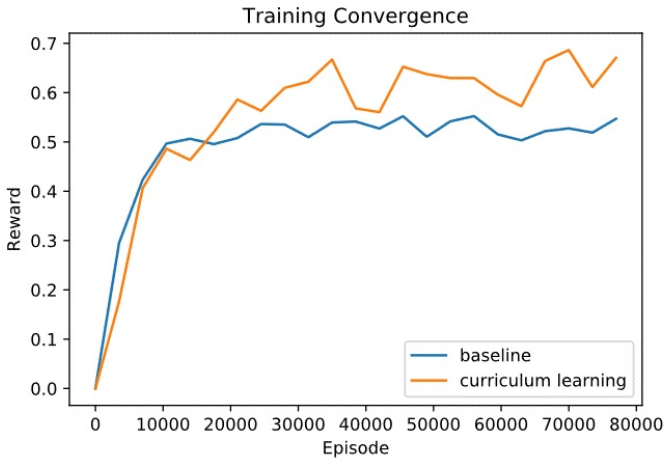


Figure 6.5: *Curriculum Learning* This figure compares the learning curve between the approach using curriculum learning and the baseline. The y-axis denotes the average rewards of the trained model in a validation dataset, and the x-axis denotes the training episodes. Both approaches converged after a certain number of steps, but the approach with curriculum learning performed better with a higher reward value. The total training steps used in both approaches are about 10^6.

For the second experiment, we evaluated the performance of the high-resolution reference images. We computed the L_2 loss and cumulative rewards and compared our approach with behavior cloning, reinforcement learning, and a combined. We drew 1000 400×400 patches

Approaches	Cumulative Rewards	L_2 Loss
Behavior Cloning	20.15	512
Reinforcement Learning	97.74	1920
Our Combined Scheme	98.25	1485

Table 6.2: *Evaluation of Painting Approaches* We evaluated the performance of behavior cloning, reinforcement learning, and our combined scheme by computing the average cumulative reward and L_2 loss between the final rendering and the reference image on the test dataset.

from 10 reference images to construct the benchmark. Moreover, we iteratively applied both algorithms 1000 times to reproduce the reference images. We used the same training dataset with images to train the models. As shown in Table 6.2, self-supervised learning had a lower L_2 loss, although both methods performed well in terms of cumulative rewards.

We also generate various results at different resolutions and with different drawing tools using both the simulated and real robotic sketching systems. For a visual comparison, please refer to the technical report [200].

6.6 Conclusion

In summary, we have introduced a novel reinforcement learning-based approach and an adaptive sim-to-real methodology. This methodology includes contact force estimation and Gaussian stroke modeling, facilitated by behavior cloning for seamless policy transfer between simulation and reality. Our robotic sketching system, mirroring the MyPaint virtual environment, empowers intricate artistic tasks in the real world. These contributions mark advancements in the field of robotic sketching systems, enhancing the creative potential of machines across diverse contexts.

Chapter 7: Conclusion, Limitations, and Future Work

In summary, our work has addressed the control challenges inherent in high-degree-of-freedom (high-DoF) robot systems, taking a multifaceted approach. In the realm of perception, we advanced from rudimentary image processing-based feature extraction to increasingly intricate convolutional neural networks (CNNs). Regarding control algorithms, our journey commenced with learning from demonstration and imitation learning, evolving into the realm of reinforcement learning. Furthermore, we explored related domains including dimensionality reduction, system identification, and the simulation-to-real transfer problem.

Our research spans several key domains: Object Deformation Modeling, Controller Parameterization, System Identification, Control Policy Learning. We initiated our exploration by using cloth manipulation as a case study, which illustrated methods for modeling high-DoF objects and established mapping relationships. Leveraging computer vision and visual feedback-based controllers, we improved our capacity to model and control objects with substantial shape variations, particularly relevant in applications involving deformable materials. Shifting our focus to Controller Parameterization, we aimed to define control parameters for high-DoF objects. Our approach involved a random forest-based controller in conjunction with imitation learning, resulting in more resilient and efficient controllers, critical for high-DoF robot systems. This approach facilitated rapid convergence of imitation learning in as few as 4-5 iterations. Furthermore, we

delved into the reduction of dimensionality for both high-DoF robot systems and objects concurrently. Our system enabled more effective utilization of computationally intensive techniques such as reinforcement learning (RL) and trajectory optimization. Consequently, we devised a system identification method to diminish the need for repeated rendering or experiments, significantly enhancing the efficiency of RL. This breakthrough enabled the resolution of algorithms with exponential computational complexity in linear time. Notably, this part of our work involved real-time collaboration between humans and robots in the manipulation of flexible objects.

In the latter part of our research, we directed our attention to the realm of natural media painting, employing reinforcement learning techniques. Painting itself can be regarded as a high-DoF robot system, involving a multitude of context-dependent actions for task completion. Our objective centered on replicating a reference image using brush strokes, guided by observations. We confronted the challenge of addressing sparse reward distribution within a vast continuous action space. Additionally, we explored the feasibility of transferring learned policies from simulated environments to real-world scenarios, with a specific emphasis on painting tasks. This research acted as a bridge between simulation and practical application, ensuring that the knowledge acquired from our work can be effectively applied in real-world settings. Ultimately, our goal is to demonstrate the deployment of RL-learned painting strategies in both virtual and real robot environments.

While our research successfully tackled increasingly complex problems, it also revealed areas where further refinement and enhancement are warranted.

7.1 Limitations

In this section, we discuss the limitations of our research from a broader perspective.

1. **Context-Specific Solutions:** Our developed solutions are often tailored to specific contexts, and their generalizability to a broader range of applications may be limited.

2. **Diversity of High-DoF Systems:** High-DoF robot systems exhibit significant diversity in kinematics, dynamics, and object interactions. Our research primarily addresses specific instances, leaving open the challenge of developing truly generalizable control policies.

3. **Data-Driven Requirements:** Many aspects of our research rely on data-driven techniques, which can pose limitations when adapting our methods to new situations requiring extensive data collection.

4. **Model Complexity:** Our exploration of control policies, especially in reinforcement learning, involves complex models that may demand substantial computational resources. This complexity can be a limitation for real-time applications and hardware constraints.

5. **Limited Generalization:** Despite efforts to enhance adaptability, our methods may struggle to transition seamlessly between different environments or robot systems, limiting their broader applicability.

6. **Deformable Object Modeling Challenges:** While crucial, modeling highly deformable objects has limitations. Not all deformable materials can be effectively modeled using our current techniques, especially non-Newtonian substances.

7. **Algorithm Validation:** Our algorithms have been validated in specific scenarios, but unexplored high-DoF kinematics, deformable objects, or system configurations may require further algorithmic verification.

8. **Sim-to-Real Transfer Complexity:** Transferring learned policies from simulated environments to real-world settings remains challenging. While progress has been made, limitations exist in achieving seamless transitions.

These overarching limitations underscore the complexity and diversity of high-DoF robot systems, highlighting the need for ongoing research to comprehensively address these challenges.

7.1.1 High-DoF Object Modeling

In the context of high-DoF robotic systems tasked with handling highly deformable objects like cloth, our introduction of the ”histogram of oriented wrinkles” (HOW) feature extraction technique and a visual feedback controller represents a significant advancement. These innovations provide solutions to the intricate challenges associated with precise modeling and manipulation of deformable objects, offering practical applications in various scenarios, as detailed in Chapter 2. Nonetheless, it is imperative to recognize that our approach does come with certain limitations that warrant further investigation and improvement.

Firstly, the effectiveness of the manipulation algorithm heavily relies on the training data specific to the given task, and the definition of the goal state is constrained by the provided demonstrations. This limitation implies that the algorithm’s performance may be limited when applied to tasks that deviate significantly from the training data or require adaptation to new goals.

Moreover, since HOW-features are computed from 2D images, the accuracy of these computations can be sensitive to variations in illumination and the relative colors of the cloth. This sensitivity may affect the algorithm's reliability in real-world settings with changing lighting conditions or diverse cloth materials.

To address these limitations and further enhance our approach, there are several avenues for future research and development. Firstly, efforts should be directed towards making the approach more robust to variations in the training data and the dynamic nature of the environment. Achieving this would enable the algorithm to adapt more effectively to different scenarios and unforeseen changes.

Additionally, exploring more effective methods for collecting training data represents a promising direction. Improving the quality and diversity of the training dataset can lead to better generalization and performance in various tasks.

Furthermore, a potential area of improvement involves creating a unified visual feedback dictionary that can be applied across different tasks. Developing such a dictionary could simplify the algorithm's implementation and potentially enhance its versatility.

7.1.2 High-DoF Controller Parameterization

In this chapter, we introduced a controller parameterization approach that specifically targets high-DoF deformable objects like cloth. Our method utilizes a random forest-based controller and demonstrates its effectiveness in handling complex manipulation tasks, thereby streamlining parameter determination and improving human-robot co-manipulation tasks, as explored in Chapter 3.

However, it is crucial to acknowledge several significant limitations associated with our approach:

One major limitation is the difficulty in extending our method to reinforcement learning scenarios. This challenge arises because our method lacks differentiability when employing a random forest construction. Consequently, reinforcement learning algorithms like the policy gradient method cannot be seamlessly integrated into our framework, limiting its applicability in scenarios that may benefit from reinforcement learning-based control.

Our method's robustness is partly contingent on the stopping criterion used in the random forest construction. The choice of this criterion can influence the performance and generalizability of the controller, and finding an optimal stopping criterion can be a non-trivial task.

Another potential drawback is the requirement for extra dimension reduction using features like the "histogram of oriented wrinkles" (HOW-feature) and action labeling during the random forest construction. While labeling is performed by mean-shift clustering of optimal actions in our work, some applications may benefit from labeling observations rather than actions. For instance, in object grasping tasks, constructing the random forest to classify object types instead of classifying actions may be advantageous, and further exploration of this approach is needed.

Our method may not be suitable for addressing high-level manipulation tasks such as cloth folding and laundry cleaning. These tasks typically involve multiple smaller manipulation tasks that necessitate a meta-algorithm capable of combining these tasks effectively. Additionally, high-level manipulation tasks often require re-grasping between different stages of control, which falls outside the scope of our current approach.

In conclusion, while our controller parameterization approach has shown promise in handling complex manipulation tasks involving high-DoF deformable objects, it is essential to rec-

ognize and address the aforementioned limitations. Future research should focus on overcoming these challenges and extending the applicability of our method to a wider range of scenarios, including those involving reinforcement learning and high-level manipulation tasks.

7.1.3 High-DoF System Identification

To improve the efficiency of reinforcement learning in high-DoF robot systems, we introduced a system identification method. This method reduces dimensionality and addresses modeling challenges while mitigating issues like slow rendering and protracted training times (discussed in Chapter 4).

One significant limitation revolves around the identification of the function $\mathbf{f}$. It is important to note that this function cannot always be identified, and there is no universally applicable method known to identify such a function for all types of robot systems. However, in the context of our two examples, the domain of $\mathbf{f}$ has a dimensionality equal to the number of controlled DoFs, and the function $\mathbf{f}$ itself does not account for dynamics (though dynamics are incorporated when $\mathbf{f}$ is integrated into $\mathbf{g}$). These observations suggest that the forward kinematic function is a suitable candidate for $\mathbf{f}$. Nonetheless, this limitation underscores the need for alternative strategies when dealing with systems where $\mathbf{f}$ is less straightforward to identify.

Another limitation arises from the effectiveness of our method being contingent on a low-dimensional space $\mathbb{C}_c$. In scenarios where $\mathbb{C}_c$ is not sufficiently low-dimensional, the method may not perform optimally, which could limit its applicability to certain high-dimensional control spaces.

Furthermore, we cannot guarantee that the function $\mathbf{f}$ represents a one-to-one mapping. In

practical terms, this means that a single control input can potentially lead to multiple quasistatic poses for a soft robot arm. This issue can complicate the learning process, especially when attempting to achieve specific robot configurations or behaviors.

Our method also faces scalability challenges when applied to systems with a large number of actuators, where the space of control inputs becomes high-dimensional. This limitation restricts the method's feasibility for many-actuator systems, and addressing this limitation is a crucial area for future research.

In terms of future directions, one avenue is to extend our grid-based structure to handle functions with special properties, such as one-to-many function mappings and discontinuous functions. Adapting the method to accommodate these scenarios would significantly enhance its versatility and applicability.

Additionally, to further improve efficiency, we are interested in exploring the use of a spatially varying grid resolution. This approach involves employing higher grid resolutions in regions where the function $\mathbf{f}$ changes rapidly, reducing the number of grid corner points that need to be evaluated. Such an enhancement could lead to more efficient learning and better adaptability in high-DoF robot systems.

In conclusion, while our system identification method represents a valuable contribution to the field of reinforcement learning in high-DoF robot systems, it is important to recognize and address the aforementioned limitations. Overcoming these challenges and exploring future research directions will be essential to maximize the method's utility and applicability in a wider range of robotic applications.

7.1.4 High-DoF System Policy Learning with Reinforcement Learning

Building on imitation learning, we explored more efficient learning strategies for complex control policies in high-DoF robot systems. Our reinforcement learning-based natural media painting algorithm showcased the advantages of combining self-supervised learning and RL to effectively learn complex actions and adapt to real-world environments (as detailed in Chapter 5).

First and foremost, as previously highlighted, our approach heavily relies on the training data for its generalization capabilities. This reliance raises concerns regarding the algorithm's effectiveness in handling unforeseen or novel situations. Ensuring the robustness and adaptability of our approach to a broader spectrum of real-world scenarios represents a critical challenge for future research.

Another noteworthy limitation pertains to the sharpness of the output, particularly in areas of high contrast within reference images. While increasing the number of strokes or image resolutions can offer some improvement, it may not constitute a scalable solution. Addressing this constraint demands the development of more sophisticated reward/loss functions that can guide the robot toward generating crisper and visually more precise results.

Furthermore, the potential expansion of the runtime steps and action space within the painting environment presents a promising avenue for enhancing generalization. This expansion can enable the self-supervised learning process to generate data that aligns more closely with the distribution of unseen data, thereby enhancing the adaptability of the learned policy. However, this extension should be undertaken judiciously to prevent the introduction of excessive complexity and computational demands.

Additionally, the incorporation of supplementary painting parameters, such as pen tilting, pen rotation, and pressure, offers a means to enrich the expressive capabilities of the painting system. These parameters have the potential to introduce subtle nuances to the strokes and further enhance the quality and realism of the artwork. Nonetheless, integrating these parameters poses technical challenges in terms of hardware compatibility and precision of control.

Lastly, the development of a more refined and nuanced definition of reward/loss functions assumes critical importance in mitigating the issue of blurry effects in the generated results. Achieving this necessitates a profound understanding of the artistic and visual aspects of painting. Future research endeavors should explore effective methods to incorporate this knowledge into the reinforcement learning framework.

In conclusion, while our research has laid the groundwork for more efficient learning strategies in complex control policies for high-DoF robot systems, it is imperative to acknowledge and address these substantial limitations. Overcoming these challenges is essential for attaining higher levels of realism and adaptability in robotic painting applications. Therefore, future work should be dedicated to these areas to advance the state of the art.

7.1.5 Transferring Learned Policies to Real Environments

We addressed the challenge of transferring learned policies from virtual to real-world environments. In Chapter 6, we introduced an approach bridging digital and robotic sketching, facilitating a smooth transition. Our innovative method demonstrated the adaptability of learned policies to practical robotic sketching tasks.

However, it's important to acknowledge that our current research has several limitations

that should be addressed in future work. Firstly, the transferability of learned policies from virtual to real-world environments may still face challenges in more complex and unstructured real-world scenarios. While our approach demonstrated adaptability in certain tasks, it might struggle with unforeseen obstacles and variations that are common in practical applications.

Additionally, our method's reliance on digital sketching as an intermediate step may introduce limitations in terms of scalability and efficiency. The computational resources required for the digital-to-robotic transition might be impractical for certain real-time applications or for robots with limited processing capabilities.

Another limitation lies in the need for extensive training data. Our current framework relies on substantial amounts of training data to learn effective policies, and acquiring such data can be time-consuming and expensive. This limitation could be particularly challenging in scenarios where the robot needs to adapt quickly to new tasks or environments.

Furthermore, the integration of additional parameters like pen tilting and pen rotation, while promising, presents challenges in terms of hardware compatibility and precision. Not all robotic platforms may support such capabilities, and ensuring accurate control over these parameters can be non-trivial.

Moreover, our system's performance may be influenced by environmental factors such as lighting conditions, surface textures, and the type of paper or canvas used. These factors can introduce variability and uncertainty into the robotic sketching process, which may require further research to mitigate.

In conclusion, while our research has made significant strides in bridging the gap between virtual and real-world robotic sketching, there are still important limitations to be addressed. Overcoming these limitations will be crucial for the widespread adoption of robotic artistic cre-

ation in practical, real-world settings, and future research should focus on tackling these challenges to push the boundaries of what is achievable in this field.

7.2 Future Work

In this section, we outline several compelling research directions aimed at advancing the field of high-DoF robot systems and deformable object manipulation. The following subsections provide a concise summary of the directions for future investigation:

7.2.1 Enhancing Adaptability and Generalizability

In the pursuit of enhancing the adaptability and generalizability of high-DoF robot systems, it is imperative that future research places a significant emphasis on the refinement of sim-to-real modeling. The crux of this endeavor lies in developing advanced techniques that can adeptly bridge the ever-persistent chasm between simulated environments and real-world applications. To achieve this, researchers must delve into the exploration of methods that not only create simulations but ones that are highly accurate, mirroring the intricate complexities of real-world scenarios with remarkable fidelity. This entails a deep-seated commitment to minimizing the inherent domain gap that exists between simulated data and the real data encountered by physical robots. The overarching goal is to ensure that control policies, meticulously trained within the controlled confines of simulated environments, can seamlessly and effectively transfer their learned knowledge and behaviors to the unpredictable and dynamic settings of physical robots operating in the real world.

However, the journey to enhanced adaptability and generalizability doesn't conclude with

sim-to-real modeling alone. It extends further into the realm of data augmentation and synthetic data generation within the simulation environment. By adopting these strategies, researchers can substantially expand the diversity and breadth of the training data available for these high-DoF robot systems. The increased variety of data enables control policies to adapt not only to known scenarios but also to unforeseen and unpredictable situations, a necessity when dealing with the complex and ever-evolving challenges of industrial applications.

Moreover, as high-DoF robot systems often work alongside human operators in industrial settings, there is a pressing need to explore methods that foster seamless human-robot collaboration. The adaptability of these systems should extend to being able to effectively interpret and respond to human input, enabling them to work in tandem with human workers to optimize efficiency and productivity. Additionally, collaboration with industry partners is crucial to the development of specialized hardware that can augment the adaptability of high-DoF robot systems. This may involve creating flexible, reconfigurable robotic arms or sensor systems capable of handling a wide array of tasks, thus furthering the versatility and adaptability of these robotic platforms.

In conclusion, the enhancement of adaptability and generalizability in high-DoF robot systems is a multifaceted endeavor, encompassing sim-to-real modeling, data augmentation, synthetic data generation, human-robot collaboration, and specialized hardware development. By collectively pursuing these avenues of research, we can empower these robotic systems to navigate and thrive in the dynamic, complex, and ever-changing landscape of industrial applications, thereby unlocking their true potential and advancing the field of robotics.

7.2.2 Integration of Advanced Machine Learning Techniques

The integration of advanced machine learning techniques, including reinforcement learning, deep learning, and transfer learning, presents a promising avenue for elevating the capabilities of control policies in high-DoF robotics [205–208]. These methods hold the potential to revolutionize the field of high-DoF robotics by enabling more robust, adaptable, and intelligent robotic systems.

Furthermore, the advent of large language models (LLMs) has opened up new and exciting opportunities for advancing robotics research and development. Our future work will involve leveraging the capabilities of LLMs in novel ways to enhance various aspects of high-DoF robotic systems. One significant direction is to explore the integration of LLMs for more effective sim-to-real modeling.

Li et al. [205,208] propose an intriguing approach that leverages LLMs for general sequential decision-making problems, which can improve the content of Chapter 2. In this framework, goals and observations are represented as sequences of embeddings, and a policy network initialized with a pre-trained LM predicts the next action. This approach has shown promise in achieving effective combinatorial generalization across different environments and supervisory modalities.

Wang et al. [207] discuss the application of LLMs in robotics and autonomous control systems, providing insights to enhance the content of Chapter 3. They propose a novel paradigm in which few-shot prompts are collected from the physical environment, enabling LLMs to autoregressively generate low-level control commands for robots without task-specific fine-tuning. This approach opens up new possibilities for using LLMs in robotics, especially in scenarios where

adapting to the physical world is crucial.

Ma et al. [206] address the challenge of using LLMs for complex low-level manipulation tasks, offering potential improvements in policy learning in Chapter 5. They introduce the EUREKA algorithm, which exploits the remarkable zero-shot generation, code-writing, and in-context improvement capabilities of state-of-the-art LLMs, such as GPT-4. EUREKA performs evolutionary optimization over reward code, resulting in rewards that can be used to acquire complex skills through reinforcement learning, surpassing expert human-engineered reward functions, thus potentially enhancing the auto-generation of RL rewards.

These developments and insights from recent research in machine learning, combined with a comprehensive focus on sim-to-real modeling, hold the promise of significantly enhancing the adaptability and generalizability of high-DoF robot systems. This, in turn, opens up new horizons for the field of robotics, where robots can seamlessly transition between simulated environments and real-world applications, making them more versatile and adaptable than ever before. We will be focusing on these directions in our future work.

Bibliography

[1] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 15, pages 627–635, 2011.

[2] Biao Jia, Zhe Hu, Jia Pan, and Dinesh Manocha. Manipulating highly deformable materials using a visual feedback dictionary. In *ICRA*, 2018.

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[4] Biao Jia, Chen Fang, Jonathan Brandt, Byungmoon Kim, and Dinesh Manocha. Paintbot: A reinforcement learning approach for natural media painting. *arXiv preprint arXiv:1904.02201*, 2019.

[5] Yoonseok Pyo, Kouhei Nakashima, Shunya Kuwahata, Ryo Kurazume, Tokuo Tsuji, ichi Morooka, and Tsutomu Hasegawa. Service robot system with an informationally structured environment. *Robotics and Autonomous Systems*, 74:148–165, 2015.

[6] Kathrin Cresswell, Sarah Cunningham-Burley, and Aziz Sheikh. Health care robotics: Qualitative exploration of key challenges and future directions. *Journal of Medical Internet Research*, 20(7):1–11, 2018.

[7] Zuyuan Zhu and Huosheng Hu. Robot learning from demonstration in robotic assembly: A survey. *Robotics*, 7(2), 2018.

[8] Michael A. Goodrich and Alan C. Schultz. Human-robot interaction: A survey. *Foundations and Trends in Human-Computer Interaction*, 1(3):203–275, 2007.

[9] Jingpei Lu, Fei Liu, Cedric Girerd, and Michael C. Yip. Image-based Pose Estimation and Shape Reconstruction for Robot Manipulators and Soft, Continuum Robots via Differentiable Rendering. *Proceedings - IEEE International Conference on Robotics and Automation*, 2023-May:560–566, 2023.

[10] Fabio Stroppa, Mario Selvaggio, Nathaniel Agharese, MingLuo, Laura H. Blumenschein, Elliot W. Hawkes, and Allison M. Okamura. Shared-Control Teleoperation Paradigms on a Soft Growing Robot Manipulator. aug 2021.

[11] M. Saha and P. Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007.

[12] M. Moll and L. E. Kavraki. Path planning for deformable linear objects. *IEEE Transactions on Robotics*, 22(4):625–636, 2006.

[13] T. Matsuno, D. Tamaki, F. Arai, and T. Fukuda. Manipulation of deformable linear objects using knot invariants to classify the object condition based on image sensor information. *IEEE/ASME Transactions on Mechatronics*, 11(4):401–408, 2006.

[14] Stephen Miller, Jur van den Berg, Mario Fritz, Trevor Darrell, Ken Goldberg, and Pieter Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31(2):249–267, 2011.

[15] Matthew Bell and Devin Balkcom. Grasping non-stretchable cloth polygons. *International Journal of Robotics Research*, 29(6):775–784, 2010.

[16] Y. Li, Y. Yue, D. Xu, E. Grinspun, and P. K. Allen. Folding deformable objects using predictive simulation and trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6000–6006, 2015.

[17] Y. Li, Xiuhan Hu, D. Xu, Y. Yue, E. Grinspun, and P. K. Allen. Multi-sensor surface analysis for robotic ironing. In *IEEE International Conference on Robotics and Automation*, pages 5670–5676, 2016.

[18] A. X. Lee, S. H. Huang, D. Hadfield-Menell, E. Tzeng, and P. Abbeel. Unifying scene registration and trajectory optimization for learning from demonstrations with application to manipulation of deformable objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4402–4407, 2014.

[19] Andreas Doumanoglou, Tae-Kyun Kim, Xiaowei Zhao, and Sotiris Malassiotis. *Active Random Forests: An Application to Autonomous Unfolding of Clothes*, pages 644–658. Springer International Publishing, 2014.

[20] A. Doumanoglou, A. Kargakos, T. K. Kim, and S. Malassiotis. Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning. In *IEEE International Conference on Robotics and Automation*, pages 987–993, 2014.

[21] Pin-Chu Yang, Kazuma Sasaki, Kanata Suzuki, Kei Kase, Shigeki Sugano, and Tetsuya Ogata. Repeatable folding task by humanoid robot worker using deep learning. *IEEE Robotics and Automation Letters*, 2(2):397–403, 2017.

[22] Stephane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey J. Gordon and David B. Dunson, editors, *International Conference on Artificial Intelligence and Statistics*, volume 15, pages 627–635. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[24] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[25] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[26] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[27] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2021.

[28] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 2020-Decem, 2020.

[29] A. Kapusta, W. Yu, T. Bhattacharjee, C. K. Liu, G. Turk, and C. C. Kemp. Data-driven haptic perception for robot-assisted dressing. In *IEEE International Symposium on Robot and Human Interactive Communication*, pages 451–458, 2016.

[30] Y. Gao, H. J. Chang, and Y. Demiris. Iterative path optimisation for personalised dressing assistance using vision and force information. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4398–4403, 2016.

[31] L. Twardon and H. Ritter. Interaction skills for a coat-check robot: Identifying and handling the boundary components of clothes. In *IEEE International Conference on Robotics and Automation*, pages 3682–3688, 2015.

[32] J. Schrimpf, L. E. Wetterwald, and M. Lind. Real-time system integration in a multi-robot sewing cell. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2724–2729, 2012.

[33] Daniel Kruse, Richard J Radke, and John T Wen. Collaborative human-robot manipulation of highly deformable materials. In *IEEE International Conference on Robotics and Automation*, pages 3782–3787, 2015.

[34] D. Navarro-Alarcon, Y. H. Liu, J. G. Romero, and P. Li. Model-free visually servoed deformation control of elastic objects by robot manipulators. *IEEE Transactions on Robotics*, 29(6):1457–1468, 2013.

[35] D. Navarro-Alarcon, H. M. Yip, Z. Wang, Y. H. Liu, F. Zhong, T. Zhang, and P. Li. Automatic 3-d manipulation of soft objects by robotic arms with an adaptive deformation model. *IEEE Transactions on Robotics*, 32(2):429–441, 2016.

[36] D. Kruse, R. J. Radke, and J. T. Wen. Collaborative human-robot manipulation of highly deformable materials. In *IEEE International Conference on Robotics and Automation*, pages 3782–3787, 2015.

[37] Leon Bodenhagen, Andreas R Fugl, Andreas Jordt, Morten Willatzen, Knud A Andersen, Martin M Olsen, Reinhard Koch, Henrik G Petersen, and Norbert Krüger. An adaptable robot vision system performing manipulation actions with flexible objects. *IEEE transactions on automation science and engineering*, 11(3):749–765, 2014.

[38] Dmitry Berenson. Manipulation of deformable objects without modeling and simulating deformation. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4525–4532. IEEE, 2013.

[39] Michael J Sullivan and Nikolaos P Papanikolopoulos. Using active-deformable models to track deformable objects in robotic visual servoing experiments. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 2929–2934, 1996.

[40] Arnau Ramisa, Guillem Alenya, Francesc Moreno-Noguer, and Carme Torras. Using depth and appearance features for informed robot grasping of highly wrinkled clothes. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1703–1708. IEEE, 2012.

[41] Yinxiao Li, Chih-Fan Chen, and Peter K Allen. Recognition of deformable object category and pose. In *IEEE International Conference on Robotics and Automation*, pages 5558–5564, 2014.

[42] Christian Bersch, Benjamin Pitzer, and Sören Kammel. Bimanual robotic cloth manipulation for laundry folding. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1413–1419, 2011.

[43] Alexander Clegg, Wenhao Yu, Zackory Erickson, C Karen Liu, and Greg Turk. Learning to navigate cloth using haptics. *arXiv preprint arXiv:1703.06905*, 2017.

[44] Yunfei Bai, Wenhao Yu, and C Karen Liu. Dexterous manipulation of cloth. In *Computer Graphics Forum*, volume 35, pages 523–532, 2016.

[45] Dale McConachie and Dmitry Berenson. Bandit-based model selection for deformable object manipulation. *arXiv preprint arXiv:1703.10254*, 2017.

[46] Marco Cusumano-Towner, Arjun Singh, Stephen Miller, James F O'Brien, and Pieter Abbeel. Bringing clothing into desired configurations with limited perception. In *IEEE International Conference on Robotics and Automation*, pages 3893–3900, 2011.

[47] François Chaumette and Seth Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, 2006.

[48] Seth Hutchinson, Gregory D Hager, and Peter I Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996.

[49] Quentin Bateux and Eric Marchand. Histograms-based visual servoing. *IEEE Robotics and Automation Letters*, 2(1):80–87, 2017.

[50] Shinichi Hirai and Takahiro Wada. Indirect simultaneous positioning of deformable objects with multi-pinching fingers based on an uncertain model. *Robotica*, 18(1):3–11, 2000.

[51] Joshua D Langsfeld, Ariyan M Kabir, Krishnanand N Kaipa, and Satyandra K Gupta. Online learning of part deformation models in robotic cleaning of compliant objects. In *ASME Manufacturing Science and Engineering Conference*, volume 2, 2016.

[52] Patrick Beeson and Barrett Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In *IEEE-RAS 15th International Conference on Humanoid Robots*, pages 928–935, 2015.

[53] Jur Van Den Berg, Stephen Miller, Ken Goldberg, and Pieter Abbeel. Gravity-based robotic cloth folding. In *Algorithmic Foundations of Robotics IX*, pages 409–424. Springer, 2010.

[54] Christophe Collewet and Eric Marchand. Photometric visual servoing. *IEEE Transactions on Robotics*, 27(4):828–834, 2011.

[55] Quentin Bateux and Eric Marchand. Direct visual servoing based on multiple intensity histograms. In *IEEE International Conference on Robotics and Automation*, pages 6019–6024, 2015.

[56] Dar-Shyang Lee. Effective gaussian mixture learning for video background subtraction. *IEEE transactions on pattern analysis and machine intelligence*, 27(5):827–832, 2005.

[57] Tai Sing Lee. Image representation using 2d gabor wavelets. *IEEE Transactions on pattern analysis and machine intelligence*, 18(10):959–971, 1996.

[58] John G Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *JOSA A*, 2(7):1160–1169, 1985.

[59] Kimitoshi Yamazaki and Masayuki Inaba. A cloth detection method based on image wrinkle feature for daily assistive robots. In *IAPR Conference on Machine Vision Applications*, pages 366–369, 2009.

[60] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[61] Leonard E Baum. An inequality and associated maximization thechnique in statistical estimation for probabilistic functions of markov process. *Inequalities*, 3:1–8, 1972.

[62] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[63] David L Donoho and Michael Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via 1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.

[64] Alexander Clegg, Jie Tan, Greg Turk, and C. Karen Liu. Animating human dressing. *ACM Transactions on Graphics*, 34(4):116:1–116:9, 2015.

[65] Yinxiao Li, Yonghao Yue, Danfei Xu, Eitan Grinspun, and Peter K. Allen. Folding deformable objects using predictive simulation and trajectory optimization. In *IROS*, 2015.

[66] J. Schrimpf and L. E. Wetterwald. Experiments towards automated sewing with a multi-robot system. In *ICRA*, pages 5258–5263, 2012.

[67] S. Rodriguez, Jyh-Ming Lien, and N. M. Amato. Planning motion in completely deformable environments. In *ICRA*, pages 2466–2471, 2006.

[68] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Resesearch*, 17(1):1334–1373, 2016.

[69] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[70] B. Frank, C. Stachniss, N. Abdo, and W. Burgard. Efficient motion planning for manipulation robots in environments with deformable objects. In *IROS*, pages 2180–2185, 2011.

[71] M. Saha and P. Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007.

[72] Hiroyuki Yuba, Solvi Arnold, and Kimitoshi Yamazaki. Unfolding of a rectangular cloth from unarranged starting shapes by a dual-armed robot with a mechanism for managing recognition error and uncertainty. *Advanced Robotics*, 31(10):544–556, 2017.

[73] Jan Stria, Daniel Průša, Václav Hlaváč, Libor Wagner, Vladimír Petrík, Pavel Krsek, and Vladimír Smutný. Garment perception and its folding using a dual-arm robot. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 61–67. IEEE, 2014.

[74] Y. Li, Xiuhan Hu, D. Xu, Y. Yue, E. Grinspun, and P. K. Allen. Multi-sensor surface analysis for robotic ironing. In *ICRA*, pages 5670–5676, 2016.

[75] J. Schulman, A. Lee, J. Ho, and P. Abbeel. Tracking deformable objects with point clouds. In *ICRA*, pages 1130–1137, 2013.

[76] Bin Wang, Longhua Wu, KangKang Yin, Uri Ascher, Libin Liu, and Hui Huang. Deformation capture and modeling of soft objects. *ACM Transactions on Graphics*, 34(4):94:1–94:12, 2015.

[77] Ibai Leizea, Ainitze Mendizabal, Hugo Alvarez, Iker Aguinaga, Diego Borro, Emilio Sanchez, undefined, undefined, undefined, and undefined. Real-time visual tracking of deformable objects in robot-assisted surgery. *IEEE Computer Graphics and Applications*, 37(1):56–68, 2017.

[78] Andreas Doumanoglou, Tae-Kyun Kim, Xiaowei Zhao, and Sotiris Malassiotis. *Active Random Forests: An Application to Autonomous Unfolding of Clothes*, pages 644–658. 2014.

[79] A. Doumanoglou, A. Kargakos, T. K. Kim, and S. Malassiotis. Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning. In *ICRA*, pages 987–993, 2014.

[80] Arnau Ramisa, Guillem Alenya, Francesc Moreno-Noguer, and Carme Torras. Finddd: A fast 3d descriptor to characterize textiles for robot manipulation. In *IROS*, pages 824–830, 2013.

[81] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[82] Daisuke Tanaka, Solvi Arnold, and Kimitoshi Yamazaki. Emd net: An encode–manipulate–decode network for cloth manipulation. *IEEE Robotics and Automation Letters*, 3(3):1771–1778, 2018.

[83] Z. Hu, P. Sun, and J. Pan. Three-dimensional deformable object manipulation using fast online gaussian process regression. *IEEE Robotics and Automation Letters*, 3(2):979–986, April 2018.

[84] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[85] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computer Survey*, 50(2):21:1–21:35, 2017.

[86] Robert F. Stengel. *Stochastic Optimal Control: Theory and Application*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[87] A. X. Lee, S. H. Huang, D. Hadfield-Menell, E. Tzeng, and P. Abbeel. Unifying scene registration and trajectory optimization for learning from demonstrations with application to manipulation of deformable objects. In *IROS*, pages 4402–4407, 2014.

[88] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IROS*, pages 5026–5033, 2012.

[89] A. Gupta, C. Eppner, S. Levine, and P. Abbeel. Learning dexterous manipulation for a soft robotic hand from human demonstrations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3786–3793, 2016.

[90] D. Mcconachie and D. Berenson. Estimating model utility for deformable object manipulation using multiarmed bandit methods. *IEEE Transactions on Automation Science and Engineering*, 15(3):967–979, July 2018.

[91] D. Berenson. Manipulation of deformable objects without modeling and simulating deformation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4525–4532, Nov 2013.

[92] S. H. Huang, J. Pan, G. Mulcaire, and P. Abbeel. Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 878–885, Sept 2015.

[93] John Schulman, Jonathan Ho, Cameron Lee, and Pieter Abbeel. *Learning from Demonstrations Through the Use of Non-rigid Registration*, pages 339–354. Springer International Publishing, Cham, 2016.

[94] Alex X. Lee, Abhishek Gupta, Henry Lu, Sergey Levine, and Pieter Abbeel. Learning from multiple demonstrations using trajectory-aware non-rigid registration with applications to deformable object manipulation. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5265–5272, 2015.

[95] D. Navarro-Alarcon, H. M. Yip, Z. Wang, Y. Liu, F. Zhong, T. Zhang, and P. Li. Automatic 3-d manipulation of soft objects by robotic arms with an adaptive deformation model. *IEEE Transactions on Robotics*, 32(2):429–441, April 2016.

[96] Rahul Narain, Armin Samii, and James F. O'Brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph.*, 31(6):152:1–152:10, 2012.

[97] G. Rogez, J. Rihan, S. Ramalingam, C. Orrite, and P. H. S. Torr. Randomized trees for human pose detection. In *CVPR*, pages 1–8, 2008.

[98] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, pages 1297–1304, 2011.

[99] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[100] Gérard Biau. Analysis of a random forests model. *J. Mach. Learn. Res.*, 13(1):1063–1095, 2012.

[101] Biao Jia, Zherong Pan, Zhe Hu, Jia Pan, and Dinesh Manocha. Cloth manipulation using random forest-based controller parametrization. *arXiv.org*, page 1802.09661, 2018.

[102] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IROS*, pages 2149–2154 vol.3, 2004.

[103] T. Wada, S. Hirai, and S. Kawamura. Indirect simultaneous positioning operations of extensionally deformable objects. In *IROS*, pages 1333–1338 vol.2, 1998.

[104] Alex X. Lee, Sergey Levine, and Pieter Abbeel. Learning visual servoing with deep features and fitted q-iteration, 2017.

[105] Q. Bateux and E. Marchand. Histograms-based visual servoing. *IEEE Robotics and Automation Letters*, 2(1):80–87, 2017.

[106] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[107] Jan Fras, Mateusz Macias, Yohan Noh, and Kaspar Althoefer. Fluidical bending actuator designed for soft octopus robot tentacle. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 253–257. IEEE, 2018.

[108] J Fras, Y Noh, M Maciaś, HA Wurdemann, and K Althoefer. Bio-inspired octopus robot based on novel soft fluidic actuator. IEEE, 2018.

[109] Zackory M. Erickson, Henry M. Clever, Greg Turk, C. Karen Liu, and Charles C. Kemp. Deep haptic model predictive control for robot-assisted dressing. *CoRR*, abs/1709.09735, 2017.

[110] Alexander Clegg, Wenhao Yu, Zackory M. Erickson, C. Karen Liu, and Greg Turk. Learning to navigate cloth using haptics. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2799–2805, 2017.

[111] E. Kanso, J. E. Marsden, C. W. Rowley, and J. B. Melli-Huber. Locomotion of articulated bodies in a perfect fluid. *Journal of Nonlinear Science*, 15(4):255–289, Aug 2005.

[112] Alexandre Munnier and Bruno Pinçon. Locomotion of articulated bodies in an ideal fluid: 2d model with buoyancy, circulation and collisions. *Mathematical Models and Methods in Applied Sciences*, 20(10):1899–1940, 2010.

[113] Yuan-cheng Fung, Pin Tong, and Xiaohong Chen. *Classical and computational solid mechanics*, volume 2. World Scientific Publishing Company, 2017.

[114] A. George and E. Ng. On the complexity of sparse qr and lu factorization of finite-element matrices. *SIAM Journal on Scientific and Statistical Computing*, 9(5):849–861, 1988.

[115] Peirce A. P. and Napier J. A. L. A spectral multipole method for efficient solution of large-scale boundary element models in elastostatics. *International Journal for Numerical Methods in Engineering*, 38(23):4009–4034.

[116] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[117] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[118] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.

[119] Jia Pan and Dinesh Manocha. Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing. *The International Journal of Robotics Research*, 35(12):1477–1496, 2016.

[120] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[121] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[122] Christopher Williams, Stefan Klanke, Sethu Vijayakumar, and Kian M. Chai. Multi-task gaussian process learning of robot inverse dynamics. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 265–272. Curran Associates, Inc., 2009.

[123] S. Genc. Parametric system identification using deep convolutional neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2112–2119, May 2017.

[124] K.J. Åström and P. Eykhoff. System identification—a survey. *Automatica*, 7(2):123–162, 1971.

[125] Stéphane Ross and J. Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1905–1912, USA, 2012. Omnipress.

[126] Mélina Skouras, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, and Markus Gross. Computational design of actuated deformable characters. *ACM Trans. Graph.*, 32(4):82:1–82:10, July 2013.

[127] Xuesu Xiao, Ellen Cappo, Weikun Zhen, Jin Dai, Ke Sun, Chaohui Gong, Matthew J Travers, and Howie Choset. Locomotive reduction for snake robots. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3735–3740. IEEE, 2015.

[128] D. Rus and M. T. & Tolley. Design, fabrication and control of soft robots. *Nature*, 521:467–475, 2015.

[129] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 167–174, New York, NY, USA, 2013. ACM.

[130] Z. Pan and D. Manocha. Realtime planning for high-dof deformable bodies using two-stage learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, May 2018.

[131] C. Duriez. Control of elastic soft robots based on real-time finite element method. In *2013 IEEE International Conference on Robotics and Automation*, pages 3982–3987, May 2013.

[132] Frederick Largilliere, Valerian Verona, Eulalie Coevoet, Mario Sanz-Lopez, Jeremie Dequidt, and Christian Duriez. Real-time Control of Soft-Robots using Asynchronous Finite Element Modeling. In *ICRA 2015*, page 6, SEATTLE, United States, May 2015.

[133] Chonhyon Park, Jia Pan, and Dinesh Manocha. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'12, pages 207–215. AAAI Press, 2013.

[134] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4569–4574, May 2011.

[135] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494, May 2009.

[136] C. Park, J. Pan, and D. Manocha. Real-time optimization-based planning in dynamic environments using gpus. In *2013 IEEE International Conference on Robotics and Automation*, pages 4090–4097, May 2013.

[137] Dustin J. Webb and Jur van den Berg. Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints. *CoRR*, abs/1205.5088, 2012.

[138] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, Oct 2012.

[139] G. Biagetti, P. Crippa, A. Curzi, and C. Turchetti. Unsupervised identification of non-stationary dynamical systems using a gaussian mixture model based on em clustering of soms. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 3509–3512, May 2010.

[140] Duy Nguyen-Tuong, Matthias Seeger, and Jan Peters. Model learning with local gaussian process regression. 23:2015–2034, 10 2009.

[141] S. R. Chu, R. Shoureshi, and M. Tenorio. Neural networks for system identification. *IEEE Control Systems Magazine*, 10(3):31–35, April 1990.

[142] W. Greblicki and M. Pawlak. Hammerstein system identification with the nearest neighbor algorithm. *IEEE Transactions on Information Theory*, 63(8):4746–4757, Aug 2017.

[143] Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

[144] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[145] Kevin Carlberg, Charbel Bou-Mosleh, and Charbel Farhat. Efficient non-linear model reduction via a least-squares petrov–galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering*, 86(2):155–181.

[146] Jean Chenevier, David González, J. Vicente Aguado, Francisco Chinesta, and Elías Cueto. Reduced-order modeling of soft robots. *PLOS ONE*, 13(2):1–15, 02 2018.

[147] Cristian Constantin Lalescu. Two hierarchies of spline interpolations. practical algorithms for multivariate higher order splines. *arXiv preprint arXiv:0905.3564*, 2009.

[148] Biao Jia, Zherong Pan, and Dinesh Manocha. Fast motion planning for high-dof robot systems using hierarchical system identification. *arXiv preprint arXiv:1809.08259*, 2018.

[149] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[150] Guoxin Fang, Christopher-Denny Matte, Tsz-Ho Kwok, and Charlie C.L. Wang. Geometry-based direct simulation for multi-material soft robots. In *ICRA*, 2018.

[151] Russell Gayle, Paul Segars, Ming C. Lin, and Dinesh Manocha. Path planning for deformable robots in complex environments. In *In Robotics: Systems and Science*, 2005.

[152] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[153] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460. ACM, 1998.

[154] Georges Winkenbach and David H Salesin. Rendering parametric surfaces in pen and ink. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 469–476. ACM, 1996.

[155] Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *CoRR*, abs/1206.4634, 2012.

[156] Ning Xie, Tingting Zhao, Feng Tian, Xiao Hua Zhang, and M Sugiyam. Stroke-based stylization learning and rendering with inverse reinforcement learning. *IJCAI*, 2015.

[157] Fan Tang, Weiming Dong, Yiping Meng, Xing Mei, Feiyue Huang, Xiaopeng Zhang, and Oliver Deussen. Animated construction of chinese brush paintings. *IEEE transactions on visualization and computer graphics*, 24(12):3019–3031, 2018.

[158] David Ha and Douglas Eck. A neural representation of sketch drawings. *CoRR*, abs/1704.03477, 2017.

[159] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[161] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2223–2232, 2017.

[162] Tao Zhou, Chen Fang, Zhaowen Wang, Jimei Yang, Byungmoon Kim, Zhili Chen, Jonathan Brandt, and Demetri Terzopoulos. Learning to doodle with deep q networks and demonstrated strokes. *British Machine Vision Conference*, 2018.

[163] He Huang, Philip S. Yu, and Changhu Wang. An introduction to image synthesis with generative adversarial nets. *CoRR*, abs/1803.04469, 2018.

[164] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[165] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

[166] Ning Xie, Tingting Zhao, and Masashi Sugiyama. Personal style learning in sumi-e stroke-based rendering by inverse reinforcement learning. *Information Processing Society of Japan*, 2013.

[167] Oliver Deussen, Stefan Hiller, Cornelius Van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. In *Computer Graphics Forum*, volume 19, pages 41–50. Wiley Online Library, 2000.

[168] Michael P Salisbury, Sean E Anderson, Ronen Barzel, and David H Salesin. Interactive pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 101–108. ACM, 1994.

[169] Kun Zeng, Mingtian Zhao, Caiming Xiong, and Song Chun Zhu. From image parsing to painterly rendering. *ACM Trans. Graph.*, 29(1):2–1, 2009.

[170] Thomas Lindemeier, Jens Metzner, Lena Pollak, and Oliver Deussen. Hardware-based non-photorealistic rendering using a painting robot. In *Computer graphics forum*, volume 34, pages 311–323. Wiley Online Library, 2015.

[171] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[172] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems*, pages 386–396, 2017.

[173] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pages 613–621, 2016.

[174] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Flow-grounded spatial-temporal video prediction from still images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 600–615, 2018.

[175] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. *arXiv preprint arXiv:1901.09005*, 2019.

[176] Andrew Owens and Alexei A Efros. Audio-visual scene analysis with self-supervised multisensory features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–648, 2018.

[177] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[178] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[179] Eric Jang, Coline Devin, Vincent Vanhoucke, and Sergey Levine. Grasp2vec: Learning object representations from self-supervised grasping. *arXiv preprint arXiv:1811.06964*, 2018.

[180] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[181] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[182] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[183] Biao Jia, Jonathan Brandt, Radomir Mech, Byungmoon Kim, and Dinesh Manocha. Lpaintb: Learning to paint from self-supervisionlpaintb: Learning to paint from self-supervision. *Pacific Graphics*, 2019.

[184] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *ICML*, 2022.

[185] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *ICML*, 2021.

[186] Ganghun Lee, Minji Kim, Minsu Lee, and Byoung-Tak Zhang. From scratch to sketch: Deep decoupled hierarchical reinforcement learning for robotic sketching agent. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5553–5559. IEEE, 2022.

[187] Gerry Chen, Sereym Baek, Juan-Diego Florez, Wanli Qian, Sang-won Leigh, Seth Hutchinson, and Frank Dellaert. Gtgraffiti: Spray painting graffiti art from human painting motions with a cable driven parallel robot. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4065–4072. IEEE, 2022.

[188] Majed El Helou, Stephan Mandt, Andreas Krause, and Paul Beardsley. Mobile robotic painting of texture. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 640–647. IEEE, 2019.

[189] Anurag Sai Vempati, Roland Siegwart, and Juan Nieto. A data-driven planning framework for robotic texture painting on 3d surfaces. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9528–9534. IEEE, 2020.

[190] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. 2022.

[191] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. 2022.

[192] Chitwan Saharia, William Chan, Huiwen Chang, Chris A. Lee, Jonathan Ho, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. 2021.

[193] Tianyang Hu, Zixiang Chen, Hanxi Sun, Jincheng Bai, Mao Ye, and Guang Cheng. Stein neural sampler. 2022.

[194] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement. 2021.

[195] Christoph Schuhmann, Romain Beaumont, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Patrick Schramowski, Srivatsa R Kundurthy, Katherine Crowson, Richard Vencu, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. Laion-5b: An open large-scale dataset for training next-generation image-text models. 2022.

[196] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. 2020.

[197] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. 2015.

[198] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *CoRR*, 2021.

[199] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[200] Biao Jia and Dinesh Manocha. Sim-to-real robotic sketching using behavior cloning and reinforcement learning. *arXiv preprint*, September 2023.

[201] Kanjivg Contributors. Kanjivg, 2023.

[202] J. Jongejan, H. Rowley, T. Kawashima, J. Kim, and N. Fox-Gieg. The quick, draw! - a.i. experiment, 2016.

[203] libmypaint contributors. libmypaint. *https://github.com/mypaint/libmypaint*, 2018.

[204] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[205] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, et al. Pre-trained language models for interactive decision-making. *Advances in Neural Information Processing Systems*, 35:31199–31212, 2022.

[206] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[207] Yen-Jen Wang, Bike Zhang, Jianyu Chen, and Koushil Sreenath. Prompt a robot to walk with large language models. *arXiv preprint arXiv:2309.09969*, 2023.

[208] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv*, 2022.