Teach a robot to assemble a bolt to a nut with a handful of demonstrations

by

Xueyang Yao

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
System Design Engineering

Waterloo, Ontario, Canada, 2025

© Xueyang Yao 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Animesh Garg Assistant Professor, College of Computing, Georgia Institute of Technology
Supervisor(s):	Bryan Tripp Associate Professor, Department of Systems Design Engineering, University of Waterloo
Internal Member:	John Zelek Professor, Department of Systems Design Engineering, University of Waterloo
Internal Member:	Nasser Lashgarian Azad Professor, Department of Systems Design Engineering, University of Waterloo
Internal-External Member:	Jeff Orchard Associate Professor, David R. Cheriton School of Computer Science, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contribution

I am the sole author of Chapters 1, 2, 5, and 6.

The work described in Chapter 3 was carried out in collaboration with Saeejith Nair, Peter Blouw, and Bryan Tripp. The simulated environment was jointly implemented by Xueyang Yao and Saeejith Nair. The experimental design and analyses were conceived by Xueyang Yao, Peter Blouw, and Bryan Tripp. All experiments and analyses were performed by Xueyang Yao under the supervision of Bryan Tripp. Figure 3.1 was designed by Bryan Tripp, while all other figures and tables were created by Xueyang Yao.

The research presented in Chapter 4 was conducted in collaboration with Yinghan Chen and Bryan Tripp. The experiments and analyses were conceived by Xueyang Yao and Bryan Tripp. Data collection was carried out jointly by Xueyang Yao and Yinghan Chen. Yinghan Chen developed the outlier detection algorithm and the Dynamic Time Warping module, while all other algorithms were implemented by Xueyang Yao. The analyses were performed by Xueyang Yao under the supervision of Bryan Tripp. All figures and tables were prepared by Xueyang Yao.

Abstract

This thesis investigates data-efficient methods for learning and executing complex, multi-step robotic manipulation tasks in unstructured environments. A two-level hierarchical framework is first proposed, in which high-level symbolic action planning is performed using Vector Symbolic Architectures (VSA), and low-level 6D gripper trajectories are modeled using Task-parameterized Probabilistic Movement Primitives (TP-ProMPs). This approach enables both interpretable planning and motion generalization from limited human demonstrations.

Building on this foundation, the thesis introduces the Task-parameterized Transformer (TP-TF), a unified model that jointly predicts gripper pose trajectories, gripper states, and subtask labels conditioned on object-centric task parameters. Inspired by the parameterization strategy of Task-parameterized Gaussian Mixture Models (TP-GMMs), the TP-TF retains the data efficiency of classical Programming by demonstration (PbD) methods while leveraging the expressiveness and flexibility of transformer-based architectures. The model is evaluated on a real-world bolt–nut assembly task and achieves a 70% success rate with only 20 demonstrations when combined with visual servoing for precision-critical phases.

The results highlight the potential of combining structured representations with deep sequence modeling to bridge symbolic reasoning and continuous control. This work contributes a step toward scalable, more interpretable, and data-efficient learning frameworks for autonomous robotic manipulation.

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor, Professor Bryan Tripp, for his exceptional guidance, thoughtful feedback, and continuous support throughout my Ph.D. journey. His mentorship challenged me to think critically and independently, and played a pivotal role in shaping my development as a researcher.

I am also grateful to the current and former members of the Bio-Robotics, Artificial Intelligence, and Neuroscience (BRAIN) Lab at the University of Waterloo. The collaborative spirit, technical discussions, and shared enthusiasm for research created a stimulating environment that I was fortunate to be a part of.

I would also like to thank my parents for their constant support and unwavering belief in me during this whole process. Finally, I would like to thank my partner, Yingchen Fan, for her unconditional love, support, and patience during this journey.

Dedication

This is dedicated to Yingchen Fan. Her love and support made this challenging journey much more enjoyable.

Table of Contents

Examining Committee ii

Author's Declaration iii

Statement of Contribution iv

Abstract v

Acknowledgements vi

Dedication vii

List of Figures xii

List of Tables xiv

List of Abbreviations xv

1 Introduction 1

2 Background 3

2.1 Program by Demonstration 4

2.1.1 Demonstration method 4

2.1.2 Encoding level 5

2.2 Deep Learning . . . 15
2.2.1 Convolutional Neural Networks . . . 15
2.2.2 Transformers . . . 17
2.3 Object Pose Estimation . . . 22
2.3.1 Classical computer vision method . . . 22
2.3.2 Deep learning method . . . 24
2.4 Visual Servoing . . . 25
2.4.1 Position-based visual servoing . . . 25
2.4.2 Image-based visual servoing . . . 27
2.5 Discussion and Summary . . . 28

3 High-Level Action Planning **30**
3.1 Introduction . . . 31
3.2 Methods . . . 32
3.2.1 Recording and Clustering States . . . 32
3.2.2 Vector-Symbolic Architecture . . . 33
3.2.3 Assembly Task . . . 33
3.2.4 Collecting Demonstration Data . . . 36
3.2.5 Building Action Representations . . . 36
3.2.6 Planning process . . . 39
3.3 Results . . . 40
3.3.1 Robustness of clustering . . . 40
3.3.2 Robustness of the HRR structures . . . 40
3.3.3 Minimal Demonstrations and Failure Modes . . . 41
3.4 Discussion . . . 42

4 Task-parameterized Movement Primitives for low-level trajectory planning 43

4.1 Introduction . . . 44

4.2 Methods . . . 46

4.2.1 Tasks . . . 46

4.2.2 Data collection . . . 46

4.2.3 Preprocessing . . . 48

4.2.4 Task-parameterized Probabilistic Movement Primitive (TP-ProMPs) 48

4.2.5 Extensions of TP-ProMPs . . . 53

4.2.6 Evaluation . . . 54

4.3 Results . . . 55

4.3.1 Pick and place . . . 55

4.3.2 Water pouring . . . 56

4.3.3 Shooting puck . . . 58

4.3.4 Sweeping . . . 59

4.4 Conclusion . . . 61

5 Task-parameterized Transformer 62

5.1 Methods . . . 62

5.1.1 Physical task . . . 62

5.1.2 Experiment setup . . . 64

5.1.3 Data collection . . . 65

5.1.4 Data preprocessing . . . 66

5.1.5 Data Augmentation . . . 67

5.1.6 Transformer . . . 71

5.1.7 Baselines . . . 74

5.1.8 Metrics . . . 75

5.1.9 Stereo Image-Based Visual Servoing . . . 76

5.2 Results . . . 79
5.2.1 Model Comparison . . . 79
5.2.2 Success rate . . . 82
5.3 Conclusions . . . 85

6 Conclusion **87**

References **90**

APPENDICES **106**

A Pose in 3D **107**
A.1 Position in $\mathbb{R}^3$ . . . 107
A.2 Orientation in $\mathbb{R}^3$ . . . 107
A.2.1 Rotation Matrix . . . 107
A.2.2 Euler angles . . . 109
A.2.3 Axis-angle representation (rotation vector) . . . 110
A.2.4 Unit Quaternions . . . 110
A.3 Pose in $\mathbb{R}^3$ . . . 111
A.3.1 Passive transformation vs. Active transformation . . . 112

List of Figures

2.1 A Block World example 8
2.2 An example of modeling a trajectory using DMPs 10
2.3 One example of modeling a trajectory using ProMPs 12
2.4 TP-GMMs model trajectories in local reference frames. 13
2.5 LeNet-5 architecture [69]. An early and compact example of a convolutional neural network. 17
2.6 Transformer architecture. Adapted from [131]. 18
2.7 3D projection of word embeddings illustrating semantic relationships between concepts such as gender. Adapted from [85]. 19
2.8 A hypothetical example of an attention pattern. 22
2.9 Pose graph for PBVS example 26
2.10 Two different views of a square object: initial and goal 29

3.1 Use p-norms with negative exponents to cluster states. 34
3.2 Simulation environment and main stages of the robotic assembly task . . . 35
3.3 Clusters of position information for bolts and nuts obtained from five demonstrations. 40
3.4 Test results for the state models obtained from clustering 41

4.1 Demonstration of the tasks 47
4.2 Object ose estimation Using stereo vision 49
4.3 Conditioning on different via-points 52

4.4 Trajectory predictions of the 5 models for 4 different tasks. 57

5.1 Subtasks of the bolt-and-nut assembly task. 63
5.2 Assembly objects close-up: (a) shows empty jig, (b) shows bolt and nut separated, and (c) shows the jig with bolt and nut inserted. 64
5.3 Robotic Setup for the Assembly Task 65
5.4 Generated trajectories with different methods. 70
5.5 Architecture of the transformer network. 71
5.6 Position Comparison 80
5.7 Orientation Comparison 80
5.8 Sample Prediction Trajectories for Each Subtask 81
5.9 Sample quaternion predictions for Subtask 3. 82

A.1 An Example of Rotation Order in 3D 108
A.2 Ambiguity of Euler angle representation at a singularity 110
A.3 Global and Local Coordinate Frames 112

List of Tables

2.1 Characteristics of different demonstration method. Table is reproduced based on [110]. 5

3.1 Global state vectors before and after the action *PUT_NUT_IN_JIG* for demonstration 1 37

3.2 Global state vectors before and after the action *PUT_NUT_IN_JIG* for demonstration 2 38

3.3 Global state vector before and after the action *PUT_NUT_IN_JIG* for demonstration 3 39

3.4 Success rate of the planner with different HRR structures with respect to different noise levels. 42

4.1 Performance comparison for the pick-and-place task with 6 training demonstrations 56

4.2 Performance comparison for the pouring water task with 10 training demonstrations 59

4.3 Performance comparison for the shooting task with 6 training demonstrations 60

4.4 Performance comparison for the sweeping task with 6 training demonstrations 60

5.1 Success rates using TP-Transformer only evaluated over 20 test trials. . . . 83

5.2 Success rates using TP-Transformer + Visual Servoing evaluated over 20 test trials. 84

List of Abbreviations

ADE Average Displacement Error 75

CNEP Conditional Neural Expert Processes 14, 74, 75, 79, 84, 85

CNMP Conditional Neural Movement Primitives 14, 74, 75, 79, 84, 85

CNN Convolutional Neural Networks 4, 15–17, 24

DLC DeepLabCut 71, 78

DMPs Dynamic Movement Primitives 44

DMPs Dynamic movement primitives 1, 10, 11

DTW Dynamic Time Warping 66, 75

EM Expectation-Maximization 53

GMM Gaussian Mixture Model 10, 53

GMR Gaussian Mixture Regression 10

GP Gaussian Process 44

GPR Gaussian Process Regression 10

HRR Holographic reduced representation 33, 38, 40, 41

IBVS Image-Based Visual Servo 25, 27–29, 76

ICP Iterative Closest Point 22–25, 28

KMPs Kernelized Movement Primitives 1, 44, 45, 54–60

LSTM Long short-term memory 6

LWR Locally Weighted Regression 10

MAP Maximum A Posteriori 53, 58

MLE Maximum Likelihood Estimate 53

MPs Movement Primitives 44

NDQ Normalized Difference of Quaternions 75, 79

NLP Natural Language Processing 17, 18

NNs Neural Networks 45

PbD Programming by demonstration v, 1, 3, 4, 9, 14, 29, 32, 36, 44–46, 62, 74, 85, 87–89

PBVS Position-Based Visual Servo 25–27

PDDL Planning Domain Definition Language 7

PnP Perspective-n-Point 22–25, 28

ProDMPs Probabilistic Dynamic Movement Primitives 44

ProMPs Probabilistic Movement Primitives 1, 11, 44, 45, 48, 50, 51, 53–61, 66

RANSAC Random sample consensus 24

RL Reinforcement Learning 89

RT Robotics Transformer 14, 86

STRIPS Stanford Research Institute Problem Solver 6, 7

SVD Singular Value Decomposition 23

TP-GMMs Task-parameterized Gaussian Mixture Models v, 1, 12, 13, 44, 45, 48, 54–61, 74, 79, 85, 87, 88

TP-ProMPs Task-parameterized Probabilistic Movement Primitives v, 2, 45, 52, 54–62, 67, 74, 79, 80, 84, 85, 87, 88

TP-TF Task-parameterized Transformer v, 2, 62, 75, 78–80, 82–89

VSA Vector Symbolic Architectures v, 1, 33, 43, 87, 88

Chapter 1

Introduction

Robots are increasingly expected to operate in unstructured environments and perform complex tasks with minimal manual programming. Traditional control pipelines, while powerful in structured industrial settings, often lack the flexibility and adaptability required for general-purpose manipulation in the real world. This challenge has driven interest in Programming by demonstration (PbD) paradigms, which enable non-expert users to teach robots by example rather than through explicit coding.

Within the PbD framework, a robot observes and generalizes from a small number of task demonstrations, allowing it to learn both high-level action structures and low-level motor behaviors. However, the field faces key challenges in trajectory generalization, symbolic abstraction, and task scalability. Traditional PbD methods—such as Dynamic movement primitives (DMPs), Task-parameterized Gaussian Mixture Models (TP-GMMs), and Probabilistic Movement Primitives (ProMPs), and Kernelized Movement Primitives (KMPs)—offer interpretable and data-efficient solutions. However, they often lack the expressiveness required for complex tasks and typically necessitate training a new model for each new task, limiting their scalability and flexibility. Conversely, deep learning-based approaches offer powerful function approximation capabilities but are typically data-hungry and struggle with generalization outside their training distribution.

This thesis first explores a two-level hierarchical framework for robotic manipulation. At the high level, Vector Symbolic Architectures (VSA) are used to infer and plan abstract actions directly from sensory data, offering symbolic representations that are both structured and biologically plausible. Early in this work, I considered biologically inspired approaches because biological systems perform well with limited training data. However, most of the thesis focuses on practical performance with limited training data, independent

of biology. At the low level, Task-parameterized Probabilistic Movement Primitives (TP-ProMPs) are employed to model 6D gripper trajectories based on task parameters, enabling generalization across varying object poses with only a small number of demonstrations.

Building on the strengths and limitations of this two-level design, the thesis then introduces the Task-parameterized Transformer (TP-TF)—a unified model that integrates both high-level subtask inference and low-level trajectory generation within a single architecture. Conditioned directly on task parameters, the TP-TF is capable of predicting gripper trajectories, gripper states, and subtask labels from a limited number of demonstrations. This unified approach offers greater flexibility, reduced system complexity, and improved performance in long-horizon manipulation tasks.

The proposed models are validated on a challenging real-world bolt-nut assembly task that requires high precision, demonstrating robust performance in environments with variable object configurations and contact-sensitive constraints. Overall, this work contributes a pathway toward scalable, mroe interpretable, and data-efficient robotic learning systems.

Chapter 2

Background

Traditional PbD has proven effective in both symbolic action planning [29, 42, 64, 81, 45] and trajectory modeling [120, 95, 17, 50] in robotic manipulation tasks. In the context of trajectory modeling, these methods achieve high sample efficiency by appropriately constraining the state space, particularly when applied to single, well-defined tasks. However, despite their success, accurately modeling both position and orientation jointly remains an open challenge.

In contrast, neural network-based approaches [113, 14, 149] leverage large datasets and the representational power of deep networks to learn a wide range of tasks from raw inputs such as images and text. These methods prioritize generality and flexibility, often at the cost of sample efficiency and interpretability.

This work aims to address the following research questions in the context of robotic manipulation tasks:

1. How can both position and orientation be effectively modeled together in a unified framework while maintaining data efficiency?

2. Is it possible to develop a model that unifies high-level symbolic planning with low-level trajectory generation to support long-horizon manipulation tasks?

3. Can a middle ground be established between traditional PbD and modern neural network-based methods—combining the sample efficiency of the former with the flexibility and expressiveness of the latter?

This chapter presents the technical foundations relevant to this work and reviews state-of-the-art methods across different domains in their respective sections. Section 2.1 introduces both traditional PbD methods and neural alternatives that have been applied to robotic learning. Section 2.2 provides a brief overview of deep learning, with a focus on two architectures: Convolutional Neural Networks (CNN), which are central to the vision module in this work, and Transformers, which later serve as the core model unifying symbolic and trajectory-level learning. Section 2.3 reviews popular techniques for object pose estimation, an essential component of the perception pipeline. Finally, Section 2.4 presents a background on visual servoing, which is used as a refinement mechanism during precision-critical phases of the task.

2.1 Program by Demonstration

PbD, also known as learning from demonstration or imitation learning, is a well-established research paradigm in robotics that enables robots to acquire skills through human demonstrations. Conceptually, PbD can be framed as a supervised learning problem, where the demonstrated tasks serve as labeled data for training. The approach offers several advantages, with two primary benefits standing out. First, PbD eliminates the need for robotics or programming expertise to program robots, making it highly appealing for applications in industrial and service robotics. Second, unlike reinforcement learning, which relies on trial-and-error exploration, PbD constrains the state space effectively, thereby enhancing sample efficiency and reducing the computational overhead associated with learning [63, 126, 62, 132, 90].

A PbD system usually works in a manner described as follows: First, a certain task will be demonstrated by a human or a robot controlled by a human. During the demonstration stage, some information or states of the environment and/or the robot itself will be recorded. The recorded data will then be used to learn a mapping between the states and the actions. The actions could be low-level motor action, trajectories, or high-level primitives.

2.1.1 Demonstration method

The demonstration method is one of the key decisions that need to be made when designing a PbD system. Three approaches are of the most interest: kinesthetic teaching, teleoperation, and passive observation [110].

Kinesthetic teaching involves a human physically guiding the robot through the desired configurations. This method is highly intuitive for demonstrators, as it directly involves physical interaction with the robot. Additionally, since the data is recorded directly from the robot, this approach inherently avoids the correspondence problem [5, 91].

Teleoperation allows a human to control a robot remotely via an interface, such as a joystick. Unlike kinesthetic teaching, the demonstrator does not need to be physically present with the robot, enabling remote operation [80, 102]. Furthermore, a well-designed interface can facilitate the control of robots performing more complex tasks, such as grasping [136, 92], assembly [119]. Similar to kinesthetic teaching, teleoperation eliminates the correspondence problem as the learning robot itself is used during the demonstration.

In passive observation, the learning robot does not participate in the demonstration stage. Instead, a human equipped with tracking devices performs the tasks, and the trajectory data of the trackers is recorded. This approach is particularly suitable for high-degree-of-freedom (DOF) or humanoid robots [133, 54, 25]. However, this requires a further mapping between human action space to robot action space.

Table 2.1 shows the advantages and disadvantages of each method. In this work, teleoperation and passive observations were used based on the characteristics of the tasks.

Demonstration	Ease of Demonstration	High DOF	Ease of Mapping
Kinesthetic	✓		✓
Teleoperation		✓	✓
Observation	✓	✓	

Table 2.1: Characteristics of different demonstration method. Table is reproduced based on [110].

2.1.2 Encoding level

In general, a skill could be presented in two levels: symbolic level and trajectories level[147]. Different encoding continuity levels not only determine the learning method but also decide the learning output.

Symbolic encoding

Symbolic encoding provides a high-level representation of skills by decomposing them into sub-tasks or action primitives [29, 42, 64, 81, 45]. In these approaches, continuous data

is first segmented into symbolic states, followed by learning a mapping between discrete states and action primitives from demonstrations. The action primitives can either be predefined or learned.

It is important to note that trajectory segmentation and state-action mapping are not independent tasks; they are closely interrelated. The level of abstraction for action primitives determines the required segmentation granularity. Consider, for example, a pick-and-place task where a robot moves a box from point A to point B. If the predefined action primitives are "Move above X," "Pick up box," and "Put down box," the segmentation would need to define sub-tasks such as "Move above A," "Pick up box," "Move above B," and "Put down box." On the other hand, if the predefined actions are more abstract, such as "Pick up box from X and place it at Y," no segmentation is required. This highlights that there is no universally correct segmentation approach; it is a design decision based on the specific task and the level of abstraction of the action primitives. For instance, in [77], segmentation is performed by iteratively identifying the most likely predefined primitives corresponding to the current segment. While [109] derives semantic rules for segmentation based on the demonstrator's hand motions observed in videos.

Given the segmented trajectory, the next step is to learn the state-action mapping which addresses the question: When to execute which primitive? Typically, this is set up as a classification problem, using various machine learning approaches. The "state" here is derived from the recorded information about the environment and/or the robot itself- and usually manually selected based on the specific task and the demands of the learning algorithm.

For example, [42] encodes the state as a 23-dimensional input that includes the objects' class, 2D location, size, and both the robot's head and tail poses, then uses a finite state machine to learn the state-action mapping in a robot soccer scoring task. In [81], a bulb-unscrewing task is considered, where the state is defined by the distance between the gripper and the goal position, as well as the gripper's velocity; a support vector machine is employed to classify the actions. Meanwhile, [34] takes a different approach by predicting the subsequent state from the current state and executed action via a convolutional Long short-term memory (LSTM) network. In this setup, the state is represented by an image depicting the scene.

State-action mapping can also be framed within a Classical AI Planning paradigm. Stanford Research Institute Problem Solver (STRIPS) [33], introduced by Fikes and Nilsson in 1971, is one of the earliest systems in this domain and remains highly influential. In STRIPS, a world state is represented as a conjunction of local states, for instance, $At(Block,\ Table) \wedge At(Ball,\ Table)$ might indicate that a block and a ball are both on the

table. Actions in STRIPS are specified through preconditions—conditions that must be satisfied before an action can be carried out—and effects, which include an add list and a delete list describing how the action changes the world state. The add list indicates the local states that might not be true before the action but will be true after the action. The delete list works the opposite way. For example, consider an action *Move(a,b,c)* for moving object a from b to c. It could be described as:

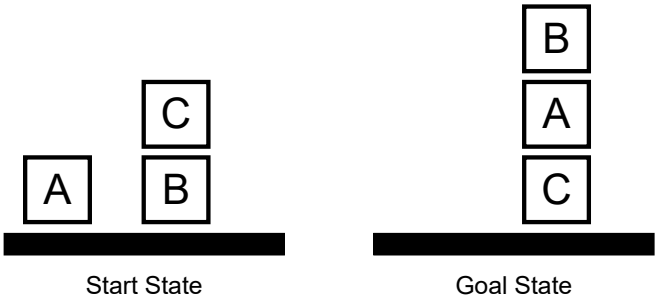
Move(a, b, c):
Precondition: On(a, b) $\wedge$ *Clear(c)*
Add list: On(a, c) $\wedge$ *Clear(b)*
Delete list: Clear(c)

where, *On(a,b)* indicates the object *a* is on object *b* and *Clear(c)* is true if nothing is on object c. Given a goal and an initial state, the planner will automatically generate a sequence of actions that transit the world model from initial state to goal state.

Planning Domain Definition Language (PDDL) [83] is an AI planning language inspired by STRIPS. By allowing negative literals, it enables to combine the add list and delete list into one effect list. For example, the same action *Move(a,b,c)* will be represented as:

Move(a, b, c):
Precondition: On(a, b) $\wedge$ *Clear(c)*
EFFECT: On(a, c) $\wedge$ *Clear(b)* $\wedge\neg$ *Clear(c)*$\wedge\neg$ *On(a, b)*

Block world is the most famous planning domain in classical planning. In the block world, one block can be stacked on another block by a robot gripper. The gripper cannot pick up a block if there is a block on top of it. Figure 2.1a shows one example of such planning problem and Figure 2.1b shows how it is defined in PDDL. One solution of the shown problem is the sequence [*MoveToTable(C, B)*, *Move(A, Table, C)*, *Move(B, Table, A)*].



(a)

$$Init(On(C,B) \wedge On(B,Table) \wedge On(A,Table) \\ \wedge Block(A) \wedge Block(B) \wedge Block(C) \\ \wedge Clear(A) \wedge Clear(C))$$

$$Goal(On(B,A) \wedge On(A,C))$$

$$Action(Move(b,x,y), \\ \textbf{Precond}: On(b,x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \\ \wedge Block(y) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y), \\ \textbf{Effect}: On(b,y) \wedge Clear(x) \wedge \neg On(b,x) \wedge \neg Clear(y))$$

$$Action(MoveToTable(b,x), \\ \textbf{Precond}: On(b,x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x), \\ \textbf{Effect}: On(b,Table) \wedge Clear(x) \wedge \neg On(b,x))$$

(b)

Figure 2.1: **A Block World example.** (a) A planning problem in block world domain. The goal is to stack the three blocks from the start state. (b) The planning problem is expressed in PDDL syntax. Adapted from [118].

[29] applies Classical AI Planning to a pick-and-place task by leveraging object pose data and k-means clustering to define states. Classical planning systems, in general, are domain-independent and rely on abstract state-action representations, making them applicable to a broad range of problems and naturally interpretable.

Trajectory Learning

Trajectory learning addresses how to map sensory inputs to low-level actions on a robot. These actions can be represented in either the joint space (e.g., motor signals or joint angles) or the task space (e.g., the end-effector's path in a global coordinate frame). Representing motions in the task space often simplifies transfer between different robots by circumventing the correspondence problem—i.e., the challenge posed by varying mechanical structures. To execute task-space trajectories on a specific robot, inverse kinematics is required to convert them into joint-space commands.

A central difficulty in PbD for trajectory learning is modeling motions that generalize to new conditions from only a limited set of demonstrations. In deep learning for computer vision, a common strategy for improving generalization with small datasets is data augmentation, which is relatively straightforward (e.g., image transformations). In contrast, data augmentation in trajectory learning, especially when using raw images as input [121], is more complicated, because creating images of novel scene configurations—together with corresponding trajectories—presents a significant challenge.

A different approach for overcoming limited data is to embed prior knowledge or structural constraints into the model. The remainder of this section reviews several of the most widely explored methods in this area.

Dynamic movement primitives [120] model a trajectory as a nonlinear dynamic system defined as

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) + f(x), \tag{2.1}$$

where y is the system state (e.g. end-effector position in task-space), g is the goal state (e.g. goal position of the end-effector), α_y and β_y are gain terms, $f(x)$ is the forcing term that governs the shape of the trajectory. The variable x (the phase) evolves according to

$$\dot{x} = -\alpha_x x. \tag{2.2}$$

and typically decreases monotonically from 1 to 0, denoting the start and end of a movement. In Eqn. 2.1, the first term $\alpha_y(\beta_y(g - y) - \dot{y})$ is a point attractor ensuring that y converges to g. The second term $f(x)$ specifies trajectory's shape. $f(x)$ is typically modelled by distributing N basis functions ψ over x:

$$f(x) = \frac{\sum_{i=1}^{N} \psi_i(x)\omega_i}{\sum_{i=1}^{N} \psi_i(x)} x, \tag{2.3}$$

where ω_i are the weights of the ith basis function ψ_i. For discrete (non-rhythmic) movements, the basis functions are often chosen to be Gaussian,

$$\psi_i(x) = exp(-h_i(x - c_i)^2), \tag{2.4}$$

with center c_i and width h_i. Periodic basis functions can be used for rhythmic movements. The weights ω_i can be learned through Locally Weighted Regression (LWR) [7] or by alternative methods such as Gaussian Mixture Model (GMM)/Gaussian Mixture Regression (GMR) [101] and Gaussian Process Regression (GPR) [32].

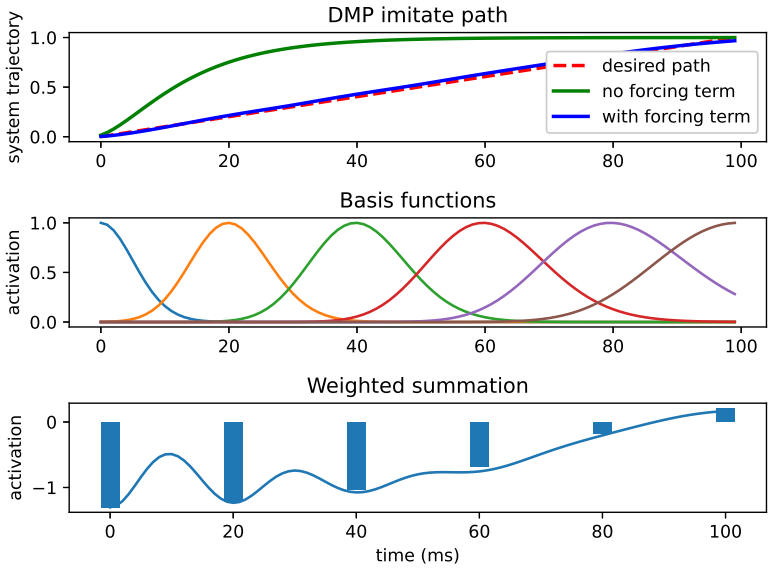


Figure 2.2: **An example of modeling a trajectory using DMPs. Top:** The desired trajectory, the trajectory generated without the forcing term, and the trajectory generated with the forcing term. **Middle:** The set of basis functions used to model the forcing term. **Bottom:** The weighted sum of the basis functions, where the bars represent the corresponding weights for each basis function.

Among the advantages of DMPs are its data efficiency—it can learn from just a single demonstration—and its ability to generalize to new positions by adjusting the goal. Furthermore, the temporal scaling factor τ can be introduced so that the learned motion can be accelerated or decelerated by modifying τ.

Nevertheless, DMPs also exhibit several limitations. For example, there is no straightforward way to encode orientation in a singularity-free manner in task space, the framework

is inherently deterministic and does not capture stochastic variations in trajectories, and it scales poorly to very high-dimensional inputs. Many of these issues have been addressed by subsequent research [95, 50, 130]. A remaining drawback, however, is that DMPs requires a goal to be specified explicitly, rather than learning it automatically from the scene.

Figure 2.2 provides an illustrative example. Without the forcing term, the model behaves purely as an attractor that converges on the goal position. In contrast, incorporating the forcing term enables the model to learn and reproduce the shape of the demonstration trajectory.

Probabilistic movement primitives [95] present a probabilistic formulation of movement primitives concept that maintains a distribution over trajectories. ProMPs model trajectories by directly assigning basis functions in the position space—unlike DMPs, which incorporates an attractor and assigns basis functions in the acceleration space—and using the weights ω as a compact representation of a trajectory. Therefore, a trajectory point $\mathbf{y}_t$ at time t can be written as

$$\mathbf{y}_t = \mathbf{\Phi}_t^\top \omega + \epsilon_y, \tag{2.5}$$

where $\mathbf{\Phi}_t^\top$ is the basis matrix at time t and $\epsilon_y \in \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_y)$ is zero-mean Gaussian noise. The probability of observing a trajectory $\tau = \{\mathbf{y}_t\}_{t=0\ldots T}$ with length T is

$$p(\tau|\omega) = \prod_t \mathcal{N}(\mathbf{y}_t|\mathbf{\Phi}_t^\top \omega, \mathbf{\Sigma}_y). \tag{2.6}$$

Now, assume ω is drawn from a Gaussian distribution $p(\omega;\theta) = \mathcal{N}(\omega|\mu_\omega, \mathbf{\Sigma}_\omega)$, the probability of observing $\mathbf{y}_t$ at time t is obtained by marginalizing ω:

$$p(\mathbf{y}_t;\theta) = \sum \mathcal{N}(\mathbf{y}_t|\mathbf{\Phi}_t^\top \omega, \mathbf{\Sigma}_y)\mathcal{N}(\omega|\mu_\omega, \mathbf{\Sigma}_\omega)d\omega \tag{2.7}$$

$$= \mathcal{N}(\mathbf{y}_t|\mathbf{\Phi}_t^\top \mu_\omega, \mathbf{\Phi}_t^\top \mathbf{\Sigma}_\omega \mathbf{\Phi}_t + \mathbf{\Sigma}_y). \tag{2.8}$$

The parameters $\theta = \{\mu_\omega, \mathbf{\Sigma}_\omega\}$ can be learned from multiple demonstrations via maximum likelihood estimation. Figure 2.3 gives a representative example. ProMPs capture the trajectory variance observed in the demonstrations. Similar to DMPs, adaptation is achieved by explicitly conditioning on a specified position at a particular time. ProMPs were initially developed to represent trajectories in joint space and later expanded for use in task space [40]. However, their capacity for extrapolation remains limited.

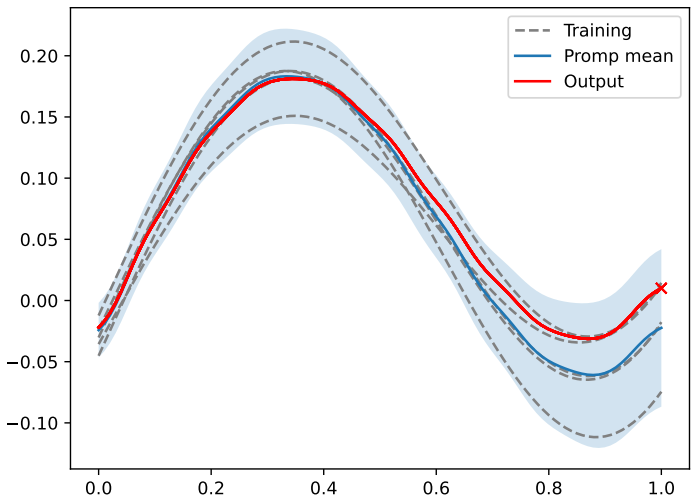


Figure 2.3: **One example of modeling a trajectory using ProMPs.** The shaded area denotes the standard deviation of the training trajectories at each time step. A output trajectory is obtained by conditioning on the goal position denoted by the red cross.

Task-parameterized Gaussian Mixture Models [17] also provides a probabilistic framework for trajectory learning, emphasizing representations in task space. These models encode the joint distribution of inputs and outputs through a Gaussian mixture. In their time-based form, TP-GMMs use time as the input and output positions (along with velocities and accelerations). By contrast, the dynamic-based formulation takes position as the input and predicts velocity or acceleration.

In Figure 2.4, TP-GMMs are illustrated by initially transforming trajectories from the global reference frame into 2 local reference frames. This procedure enables the model to capture low-variance features that might be obscured when viewed solely in the global reference frame. Within each local frame, a mixture of Gaussians is employed to represent the trajectories.

During reproduction, the Gaussians in each local reference frame are transformed back to the global frame according to the new poses of those local frames. The final trajectory is then obtained by taking the product of these Gaussians, which can be interpreted as a "weighted voting" mechanism. In this scenario, the covariance of each Gaussian serves as a weight: higher covariance reduces its influence, whereas lower covariance increases it.

For example, at the beginning of the trajectories shown in Figure 2.4, Frame 1 exhibits smaller covariance and thus exerts greater influence in the global reference frame. Near the end, Frame 2 has lower covariance and therefore becomes more influential. By leveraging multiple local reference frames, TP-GMMs achieve strong extrapolation capabilities while obviating the need to explicitly define a goal position; instead, the goal is implicitly encoded in the poses of the local frames.

In [144], the TP-GMMs framework is extended to handle quaternions within a 3-dimensional Riemannian manifold, ensuring that the inherent geometry of quaternion space is preserved. This extension enables TP-GMMs to model not only the position but also the orientation. Nevertheless, performing Gaussian product and conditioning on a Riemannian manifold necessitates iterative computations in the tangent space. Specifically, logarithmic mapping is used to project data points from the manifold onto the tangent space, which serves as a local linear approximation at a reference point. Gaussian operations—including the computation of means and covariances—are then carried out within this tangent space. Finally, the results are mapped back onto the manifold using exponential mapping, ensuring consistency with the manifold's geometry. Despite its theoretical elegance, a publicly available implementation that jointly models position and orientation using the task parameters is not provided.

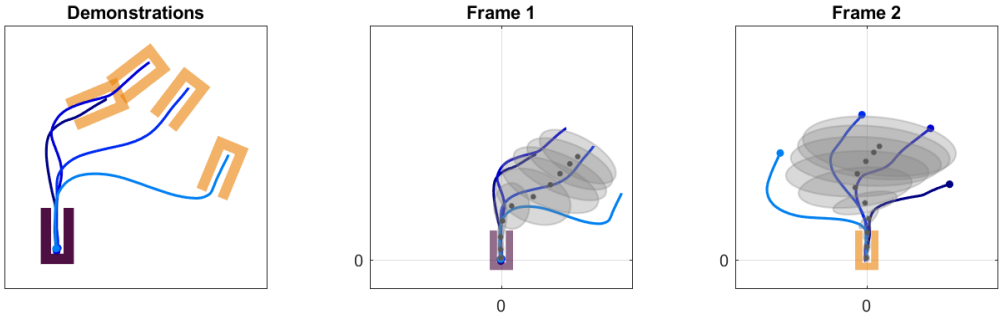


Figure 2.4: **TP-GMMs model trajectories in local reference frames. Left:** 2D trajectories shown in the global reference frame for a task involving motion from the purple box to the yellow box. **Middle:** Trajectories expressed in Frame 1, which is attached to the purple box. **Right:** Trajectories expressed in Frame 2, attached to the yellow box. The shaded regions represent the Gaussian mixture components. Figure adapted from [17].

Neural network-based methods have recently been used extensively to model trajectories. Gato [113], introduced in 2022, is an example of a generalist agent capable of a wide

range of tasks: playing Atari games, captioning images, chatting, and stacking blocks with a real robot. PaLM-E [27], also introduced in 2022, is another broadly applicable model that handles tasks such as grasping and stacking objects, table-top pushing, and mobile manipulation. Around the same time, Robotics Transformer (RT) [14, 149] was proposed with a focus on robotic tasks including picking and knocking objects, placing them upright, moving them around, and opening and closing drawers, as well as transferring objects into and out of drawers.

Although these methods differ slightly in network architectures and task sets, they share several key features. First, they all employ the Transformer framework [131]. Second, they utilize multi-modal inputs, such as visual data (camera images) and textual task descriptions. Third, they capitalize on extensive datasets, which facilitates the transfer of robust visual and semantic representations. For instance, [149] illustrates that their model achieves better generalization to unseen objects, backgrounds, and environments compared to more traditional methods. However, this strength comes at the cost of requiring massive amounts of data for training. For example, Gato uses 387k successful trajectories in simulation and 15k real-world trajectories to learn a block-stacking task. RT-2 is trained on data collected over 17 months from 13 different robots. It is worth noting that these training corpora exclude natural language and general image data, which tend to be more readily available than specialized robotic control datasets. This stands in contrast to classical PbD approaches, where a handful of demonstrations can be sufficient. For example, [18] uses just 4–7 demonstrations to learn a pick-and-place task.

Some neural network-based methods focus more on data efficiency. A notable example is Conditional Neural Movement Primitives (CNMP)[121], introduced in 2019, along with its extension, Conditional Neural Expert Processes (CNEP)[142], proposed in 2024 to address tasks with multimodal solutions. In an obstacle avoidance task, CNMP required only 8 demonstrations to generalize to new situations within the training distribution. For this particular task, a key difference is that instead of relying on image embeddings, CNMP uses the heights of both the target object and the obstacle as inputs, which removes the need to learn a mapping between images to scene configuration.

Although the task itself is realativly simple—because the objects remain stationary and the only parameters are their heights—it prompts a question: What inputs should the network receive? Using images is simple and generalizes well across various tasks, as it requires minimal preprocessing. However, it also closely couples vision with control. For instance, in RT-2, the model's ability to handle unseen objects, backgrounds, and environments depends mainly on vision, which must locate unfamiliar objects in new contexts; once the objects are identified, the control task does not change significantly.

For several reasons, it may therefore be beneficial to separate the vision component from the control component. First, vision data tends to be much easier to obtain than control data, so decoupling them reduces the need for large control datasets. Second, it enables direct input of relevant information—such as object color, size, and pose—to the network, which can simplify the control learning process and improve the interpretability. Finally, it opens up new possibilities for data augmentation for the control task.

2.2 Deep Learning

In 2012, the backpropagation-based deep neural network known as "AlexNet" [66] achieved groundbreaking results in the ImageNet competition, igniting the rapid expansion of deep learning research. Since then, Deep learning has advanced at a rapid pace, especially in the fields of computer vision and natural language processing. Its advantages over traditional machine learning approaches include the ability to automatically extract features from unstructured data, achieve high accuracy on complex tasks, and scale effectively with larger datasets. This section introduces key architectures and specific models relevant to the work presented in this thesis.

2.2.1 Convolutional Neural Networks

The core component of a CNN is its convolutional layers. Depending on the application, the architecture may also incorporate pooling layers and a series of dense (fully connected) layers. Figure 2.5 illustrates a simple CNN architecture—LeNet-5 [69]—which consists of convolutional, pooling, and dense layers. In this example, the input is a $3 \times 32 \times 32$ tensor, representing a 32×32 RGB image, and the network produces a 10-dimensional output. Activation functions (e.g., ReLU) are omitted for clarity. Originally developed for handwritten digit recognition, LeNet-5 introduced several foundational components that continue to influence modern CNN design.

Convolutional Layer

At the core of a CNN is the convolutional layer, which performs a convolution operation using two inputs: an input tensor and a kernel (or filter) tensor. The kernel, usually much smaller than the input, performs a dot-product with localized regions of the input as it slides over it (see Figure 2.5), producing what is known as a feature map. A convolutional layer is

characterized by its kernel size K, the number of channels C (i.e., the number of kernels), the stride S, and the padding P. Here, K represents the kernel's dimensions (assuming square kernels, though non-square options such as 1D kernels for time-series tasks [138] are also possible). The stride S indicates how far the kernel moves after each dot product computation, while the padding P specifies the number of extra entries (typically zeros) added around the input's border to allow the kernel to operate at the edges and preserve spatial dimensions. For an input of size $H_{in} \times W_{in}$, the output spatial dimensions for each channel are given by:

$$H_{out} = \frac{H_{in} - K + 2P}{S} + 1, \tag{2.9}$$

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1. \tag{2.10}$$

Thus, the output tensor has dimensions $C \times H_{out} \times W_{out}$. In Figure 2.5, the convolutional layer maintains the spatial size of the input (achieved, for example, with $K = 3$, $S = 1$, $P = 1$, and $C = 8$).

The convolution operation is essentially a feature extraction process, where the kernel scans the input to identify and match specific patterns. Different layers capture features of increasing complexity: shallow layers may detect edges, middle layers recognize shapes, and deeper layers identify objects. Unlike dense layers—where each neuron connects to every neuron in the previous layer—a convolutional layer's neurons are connected only to a local region, facilitating the detection of spatial patterns. Moreover, by applying the same kernel across the entire input, CNNs reduce the number of parameters and achieve translation invariance.

Pooling Layer

Pooling layers are widely used in CNNs to summarize local statistics of feature maps [41]. For example, max pooling [146] selects the maximum value within each pooling window, while average pooling [13] computes the mean. Unlike convolutional layers, pooling layers do not contain learnable parameters. Instead, a fixed operation (e.g., max or average) is applied. The output size can be computed using the same equations as in Equations 2.9 and 2.10. Pooling layers typically reduce the spatial dimensions, improve computational efficiency, and introduce invariance to small translations. Additionally, pooling contributes to handling inputs of varying sizes.

CNN have achieved remarkable performance in a wide range of computer vision tasks, such as image classification [67, 124, 46], semantic segmentation [78, 114], and object detection/localization [39, 112]. They have also been extensively utilized as vision modules in downstream robotic applications [71, 76, 12], enabling end-to-end perception and control pipelines.

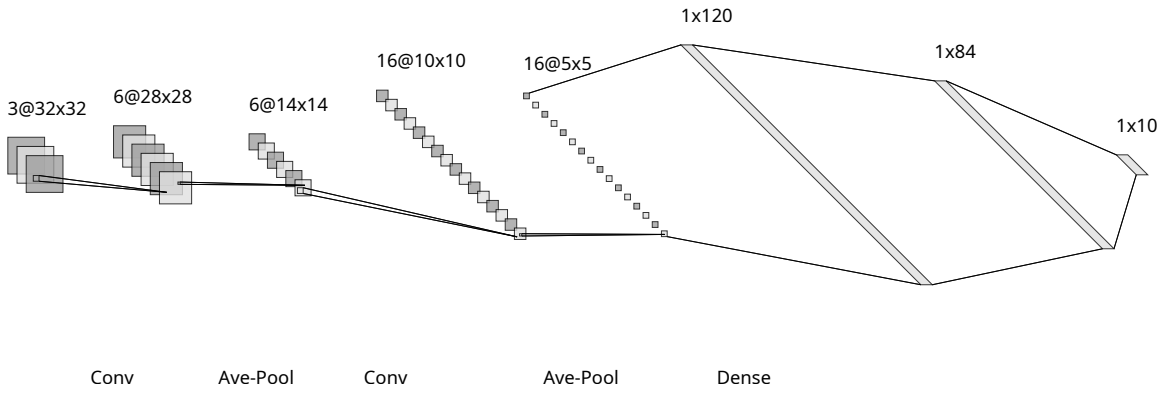


Figure 2.5: LeNet-5 architecture [69]. An early and compact example of a convolutional neural network.

2.2.2 Transformers

Originally introduced by Vaswani *et al.* in their work *"Attention Is All You Need"* [131], the Transformer architecture has achieved state-of-the-art performance across a wide range of Natural Language Processing (NLP) tasks [106, 107, 15, 2, 108, 24]. Beyond NLP, it has also been successfully extended to computer vision [26] and robotics [14, 149, 27, 113]. Figure 2.6 illustrates the fundamental structure of a Transformer.

In the following, we discuss the key components of the Transformer using a simplified language modeling task as a toy example. Specifically, given the prompt *"A black hex nut is on the"*, the model must predict the most likely next word.

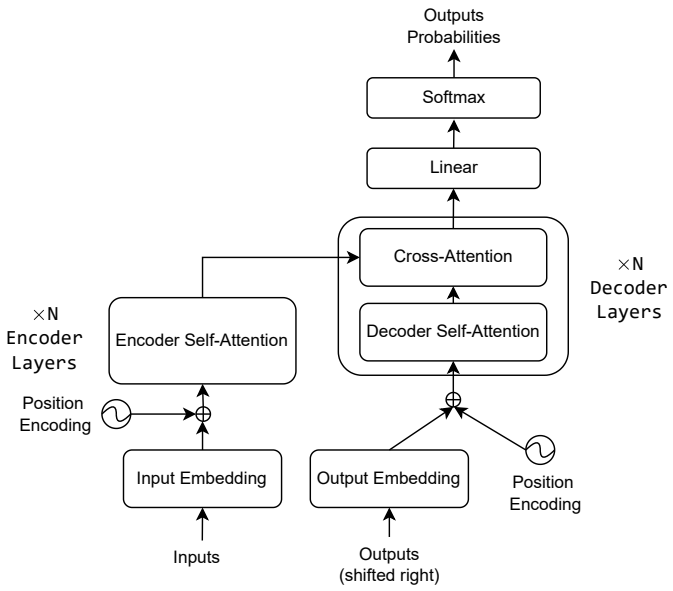


Figure 2.6: Transformer architecture. Adapted from [131].

Tokenization

Tokenization involves segmenting the input data into smaller, discrete units known as tokens. In the context of computer vision, this often entails dividing an image into patches, whereas in NLP tasks, text may be split into words, characters, or subword units (e.g., *playing* $\rightarrow$ [*play*, *ing*]). Each token is then mapped to a numerical token ID, which the neural network can interpret and process. Subword tokenization methods, such as Byte-Pair Encoding [122], are commonly employed in Transformer-based models to handle the vast vocabulary space efficiently.

In our toy example—predicting the next word given the prompt *"A black hex nut is on the"*—applying the GPT-4 tokenizer results in the sequence of tokens [*"A"*, *" black"*, *" hex"*, *" nut"*, *" is"*, *" on"*, *" the"*], corresponding to token IDs [32, 3776, 12651, 10184, 374, 389, 279]. These numeric IDs are subsequently fed into the embedding layer for further processing by the Transformer.

Embedding

The embedding process maps discrete tokens to continuous, high-dimensional vectors. The idea is that in high dimensional space, the direction of the vector correspond with a semantic meaning. For example, if $E(x), E(y)$ denote the embeddings of tokens x and y, respectively, then their dot product $E(x) \cdot E(y)$ provides a measure of similarity. Positive values indicate semantic similarity (i.e., vectors pointing in roughly the same direction), negative values suggest opposing semantics, and values near zero imply orthogonality. A commonly used example [85] is that the difference vector $E(king) - E(queen)$ often aligns closely with the vector $E(man) - E(woman)$, indicating that there is a distinct dimension capturing the sematic of "gender" (see Figure 2.7 for a 3D visualization).

Formally, the embedding matrix has dimensions $d_{voc} \times d_{model}$, where d_{voc} is the vocabulary size and d_{model} is the embedding dimension. Each token ID serves as an index into this matrix, retrieving a vector embedding that is learned through backpropagation.

In our toy example, the sequence of token IDs $[32, 3776, 12651, 10184, 374, 389, 279]$ corresponds to embeddings $E_1, E_2, ..., E_7$. The overall embedding of the sequence is then given by $\mathbf{E} = [E_1, E_2, ..., E_7]^\top$, resulting in a tensor with shape $(7 \times d_{model})$.

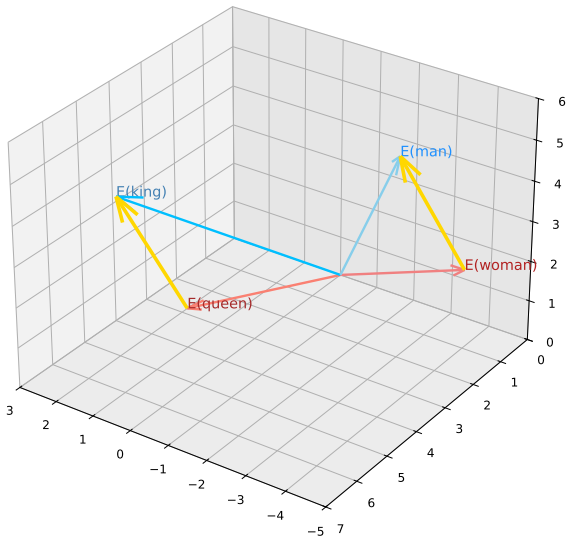


Figure 2.7: 3D projection of word embeddings illustrating semantic relationships between concepts such as gender. Adapted from [85].

Positional Encoding

Positional encoding allows the model to incorporate information about the ordering of tokens in a sequence. In our toy example, it ensures that the model recognizes the first token as "A," the second token as " black," and so on. Following [131], one standard approach computes the positional encodings as:

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{model}}})$$
$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{model}}}),$$

where *pos* denotes the position of a token within the sequence and i indexes the embedding dimension. Hence, sine is used to encode information for even indices, and cosine for odd indices. Alternatively, learned positional encodings [38] may be employed. These positional encodings are added to the token embeddings, preserving the overall dimensionality. Therefore, after the positional encoding, the output for our toy example retains its shape of $(7 \times d_{model})$.

Attention

In Transformer-based models, the token embedding layer can be viewed as a simple lookup table, such that the embedding of a single token only captures a generic semantic representation of that token. The attention mechanism enables the network to incorporate the contextual references necessary for disambiguating meaning. For instance, consider the token " nut," which can refer either to "a fruit consisting of a hard or tough nutshell protecting a kernel" or to "a piece of metal used alongside a mating bolt." Only when preceded by the token " hex" does the context make it clear that the latter sense is intended. In other words, attention facilitates the transfer of information from one token's embedding to another, enriching the overall representation with context.

An immediate question, then, is how the model determines which tokens in a sequence matter to a given token and how those relevant tokens ultimately modify that token's meaning. In [131], this is achieved with the equation:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V, \tag{2.11}$$

where Q, K, V are queries, keys, and values matrix. d_k is the dimension of the key vector. This compact formula lies at the heart of the Transformer, yet it can be challenging to fully grasp. A helpful way to interpret it is to start by looking at how queries and keys are formed. Each of the token embeddings E_i (dimension d_{model}) is multiplied by a $d_{model} \times d_q$ query matrix W_Q to produce a query vector Q_i (dimension d_q). Typically, d_q is much smaller than d_{model}. Similarly, each E_i is also multiplied by a $d_{model} \times d_k$ key matrix W_K to yield a key vector K_i.

Intuitively, if the i-th token is related to the j-th token, the dot product $Q_i \cdot K_j$ will be positive, and if they are unrelated, it will tend toward zero or a negative value. By taking the dot product of Q_i and all key vectors and then applying the softmax function, we obtain a possibility distribution indicating which tokens are related to the i-th token. In our toy example, the token "nut" is related to the token "hex" and the token "black". Therefore softmax($Q_4 \cdot K_3$) and softmax($Q_4 \cdot K_2$) are expected to have have a positive value. Repeating this for every Q_i generates an attention pattern or attention weights over the entire sequence (See Figure 2.8 for a hypothetical attention pattern for the toy example). In practice, all Q_i K_i vectors are stacked into matrices Q and K, and the operation softmax($QK^\top$) is performed for all tokens in parallel. The division by $\sqrt{d_k}$ in Eq. 2.11 is primarily for numerical stability.

The next step is to update each token embedding based on these attention weights. Returning to the "nut" example, its embedding is adjusted to capture the context of "black hex nut". Concretely, the $7 \times d_{model}$ embedding is multiplied by the $d_{model} \times d_v$ value matrix to produce the $7 \times d_v$ values V. Multiplying the attention weights with the values V yields a context-aware embedding (dimension $(7 \times d_v)$).

The mechanism described thus far corresponds to a single "head" of the attention block. In practice, there are $h = \frac{d_{model}}{d_q}$ heads (Usually we have $d_q = d_k = d_v = \frac{d_{model}}{h}$), each producing its own attention pattern and embedding updates. This multi-headed structure provides the network with greater flexibility in capturing different types of dependencies between tokens. Hence, for the toy example, Q, K, V would each be $7 \times d_{model}$, which then split into $7 \times h \times \frac{d_{model}}{h}$ for attention calculation. Therefore, the output of equation 2.11 would be $7 \times h \times d_v$. Concatenate different heads and then multiply by the $(d_{model} \times d_{model})$ output matrix W_O to produce the $7 \times d_{model}$ output of the attention block. A residual connection is then applied before passing the result to a feed-forward layer.

Transformers have achieved remarkable success for several key reasons. First, the attention mechanism enables the model to capture context-dependent representations of each token, allowing a deeper understanding of the entire sequence. Second, unlike recurrent architectures, attention operations across tokens and heads can be executed in parallel,

significantly accelerating the training process.

	A (Q_1)	black (Q_2)	hex (Q_3)	nut (Q_4)	is (Q_5)	on (Q_6)	the (Q_7)
A (K_1)	1.0	0.0	0.0	0.0	0.0	0.0	0.0
black (K_2)	0.0	1.0	0.0	0.45	0.0	0.0	0.0
hex (K_3)	0.0	0.0	1.0	0.55	0.0	0.0	0.0
nut (K_4)	0.0	0.0	0.0	0.0	0.0	0.0	0.0
is (K_5)	0.0	0.0	0.0	0.0	1.0	0.0	0.0
on (K_6)	0.0	0.0	0.0	0.0	0.0	1.0	0.0
the (K_7)	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Figure 2.8: A hypothetical example of an attention pattern.

2.3 Object Pose Estimation

Many robotic tasks involve manipulating physical objects, which requires an understanding of their positions and orientations. As a result, estimating the pose of objects within a scene is a critical first step in a robotic control pipeline.

2.3.1 Classical computer vision method

In classical computer vision, object pose estimation is achieved in a multi-staged process including object segmentation, feature extraction, and feature matching or point cloud alignment. Object segmentation extracts the object of interest from the background. Feature extraction finds keypoints of the object of interest. SIFT [79], SURF [8], ORB [116] are commonly used for feature extraction. Depending on the availability of depth information, the final step involves either feature matching or point cloud alignment to determine the object's 6D pose. Feature matching often employs variants of the Perspective-n-Point (PnP) algorithm, while point cloud alignment frequently utilizes Iterative Closest Point (ICP). Although classical methods do not require training data, they are sensitive to environmental conditions and tend to struggle with textureless or symmetric objects.

Perspective-n-Point

The PnP method was first introduced for camera calibration [35, 48, 1, 105], where its objective was to determine the camera's position and orientation relative to an object in a scene, based on n corresponding points. In the context of object pose estimation, PnP instead seeks to determine the pose of the object in the camera's reference frame. Concretely, if we have a set of known 3D points $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ - for example, keypoints on a CAD model-and their corresponding 2D projections $(\mathbf{u}, \mathbf{v})$, PnP aims to estimate the rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ satisfying

$$s \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{1} \end{bmatrix} = \mathbf{K}(\mathbf{R} \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \\ \mathbf{Z} \end{bmatrix} + \mathbf{t}),$$

where $\mathbf{K}$ is the intrinsic parameter matrix of the camera and s is a scaling factor. This can be solved iteratively for robustness using RANSAC [35] or through efficient closed-form solutions [70].

Iterative Closest Point

ICP [9, 117, 145] estimates the optimal rotation $\mathbf{R}$ and translation $\mathbf{t}$ that align two point clouds. This process involves two primary steps repeated until convergence. First, correspondence is established by identifying, for each point in the source cloud, the closest point in the target cloud. Second, an alignment procedure solves for $\mathbf{R}$ and $\mathbf{t}$ that minimize the objective

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} ||\mathbf{R} \cdot \mathbf{p}_i + \mathbf{t} - \mathbf{q}_i||,$$

where $\mathbf{p}_i$ and $\mathbf{q}_i$ are points from the source and target point clouds, respectively, and N is the total number of points. Typically, this step is solved using either Singular Value Decomposition (SVD) or a least squares approach. Because the success of ICP depends heavily on an accurate initial pose estimate, a poor initial guess can lead the algorithm to converge to a local minimum or fail to converge entirely. Consequently, ICP is often employed as a refinement step, with an initial estimate provided by a neural network [139, 59]. When the correspondences are known in advance, ICP reduces to a straightforward alignment problem.

2.3.2 Deep learning method

As deep learning continues to evolve, neural networks—particularly CNN architectures—have proven highly effective for object pose estimation, typically using RGB or RGB-D images and achieving state-of-the-art performance in this domain. Recent work provides further classification of 6D object pose estimation tasks [47]. For instance, 6D localization assumes prior knowledge of the number and types of objects (e.g., given a list L with n_i instances of each object o_i), and the objective is to produce an equivalent list of pose estimates E. Meanwhile, 6D detection focuses on the scenario where only images of a scene are provided, without prior information on object types or quantities. In this thesis, when referring to 6D object pose estimation, we specifically mean 6D detection. Broadly, 6D object pose estimation methods can be categorized as either direct or indirect. Direct methods take scene images as input and output object poses directly. Indirect methods first identify keypoints in the image plane, then estimate poses using algorithms such as PnP or ICP. In essence, indirect methods replace the feature-extraction stage in classical computer vision with a neural network.

Direct methods

Direct methods [139, 134, 74, 59, 60, 68, 16] take one or several images as input fed to the network and directly output the 6D poses of the objects along with the object category. For example, PoseCNN [139] is a CNN-based neural network that predicts object poses given an image. PoseCNN has 3 subtasks: Semantic labeling where the network classifies each image pixel into object class, translation vector $\mathbf{T}$, and rotation $\mathbf{R}$ represented by quaternion for each object. Interestingly, PoseCNN does not directly regress to the translation vector $\mathbf{T}$, instead, it regresses to an image-plane object center (c_x, c_y) obtained by voting from all the pixels in an object class and the depth T_z. $\mathbf{T}$ is then recovered given the camera intrinsic parameters. [134] utilizes dense 2D-3D correspondence between pixels in the image plane and the object coordinate obtained from an object CAD model to directly regress object 6D pose. A learnable Patch-PnP module is used to replace the traditional PnP/Random sample consensus (RANSAC) process. Despite the differences in architecture and intermediate tasks, these methods share common characteristics. First, although the gap is getting smaller, direct methods show poorer results compared to indirect methods [134]. Second, direct methods are challenging to implement for objects not present in standard benchmark datasets (e.g., Linemod, Linemod-Occlusion, YCB-V). This is because obtaining ground-truth 6D poses requires RGB-D cameras and corresponding object meshes, a process that is neither simple nor easily scalable.

Indirect Methods

Indirect methods [129, 100, 99, 143, 128] reduce the problem of 6D pose estimation to detecting 2D keypoints first, followed by a PnP or ICP step. Different approaches define keypoints in different ways. For instance, [129] and [128] use the vertices of the bounding box as keypoints. [99] sample keypoints directly on the surface of the 3D object using the farthest point sampling algorithm. Other methods such as [97, 143] aim to predict dense per-pixel correspondences between the 2D image and the 3D model.

Because the first step is purely keypoint detection, any network capable of identifying 2D keypoints, in principle, can be integrated into this pipeline. For example, Deeplabcut [82], originally developed for body part detection in human and animal pose estimation, can be repurposed to detect user-defined keypoints on novel objects. Its backbone network, pretrained on ImageNet [46], requires only a small number of manually labeled training images ($\sim$ 200) to learn to detect new classes of keypoints effectively.

2.4 Visual Servoing

Visual servoing [51, 19, 22] is a robotic control method wherein visual data from one or more cameras is employed to guide a robot's motion. By continuously updating the robot's actions based on real-time visual feedback, this method enables highly precise positioning, tracking, or manipulation. Although, visual servoing can be applied in different kinds of applications (e.g. mobile robots [58, 72], unmanned aerial vehicles [36, 61]), this thesis focus on robot manipulation tasks. There are two principal configurations for placing the camera relative to the robot: one in which the camera is fixed in the environment, and another—known as the eye-in-hand configuration—where the camera is attached to the robot's end effector. This thesis focuses exclusively on the latter configuration.

Visual servoing typically employs two different control approaches: Position-Based Visual Servo (PBVS) and Image-Based Visual Servo (IBVS). The following subsections discuss both methods in detail.

2.4.1 Position-based visual servoing

In PBVS, the target object pose with respect to the camera, denoted as $T_O^{C^*}$ is assumed to be known, and the current object pose with respect to the camera, denoted as T_O^C is estimated (See Figure 2.9). Here, C^* and C refer to the target and current camera reference

frames, respectively, while O denotes the object reference frame. Methods for estimating the object pose are discussed in Section 2.3. The required pose change for transitioning from the current camera pose to the target camera pose in the camera's reference frame is given by:

$$T^{\Delta} = T_O^C (T_O^{C^*})^{-1}. \tag{2.12}$$

Finally, this relative pose can be mapped into the global reference frame by multiplying T^{Δ} with T_C^G.

In practice, this movement is not executed in a single step. Instead, a parameter $\lambda \in (0, 1)$ is introduced to determine the fraction of the overall motion performed at each iteration. At every time step, the relative pose T^{Δ} is interpolated, and a displacement scaled by λ is executed. Furthermore, the pose of the target object can be re-estimated after a specified number of steps, thus providing feedback and compensating for inaccuracies that may arise—such as pose estimation errors when the camera is far from the object.

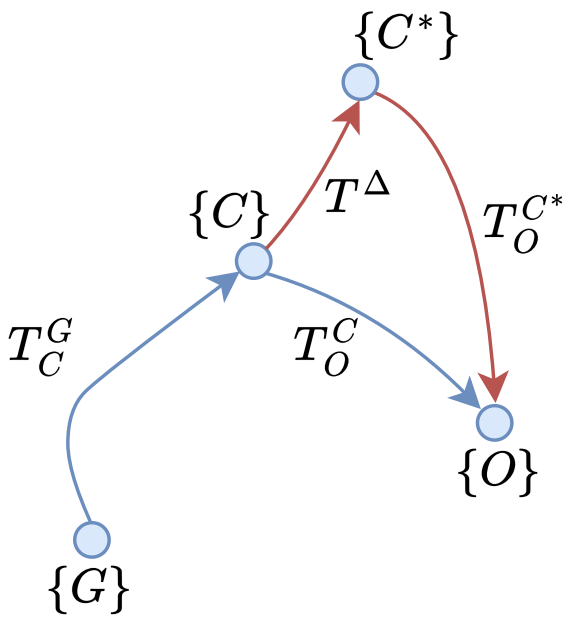


Figure 2.9: **Pose graph for PBVS example**. Frame $\{G\}$ is the global reference frame. Frame $\{C\}$ and Frame $\{C^*\}$ are the current and desired camera's reference frames, respectively. Frame $\{O\}$ is the object's reference frame. Adapted from [22].

Although PBVS provides efficient, straight-line Cartesian motion in the workspace, it can cause image features to drift out of the camera's field of view (FOV). Moreover, PBVS

relies on accurate object pose estimation, which typically requires a dense feature detection or tracking process on the object.

2.4.2 Image-based visual servoing

Unlike PBVS, which controls the camera to align the object's current pose with the desired pose in the camera frame, IBVS achieves control by aligning feature points within the image plane. Figure 2.10 illustrates two perspectives of a square object defined by its four corner points: circles indicate the initial camera view, and stars indicate the goal camera view. It can be seen that the initial camera position is farther from the object than the goal position and requires rotation about its optical axis. The control objective is to move the camera until the four corner points from the initial view overlap with those from the target view.

Given that the camera moves in 3D space with velocity $\mathbf{v} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z) \in \mathbb{R}^6$ expressed in the camera reference frame—where (v_x, v_y, v_z) represent translational velocities and $(\omega_x, \omega_y, \omega_z)$ represent rotational velocities—it is necessary to model the resulting motion of points projected onto the image plane. The velocity $\dot{\mathbf{p}} = (\dot{u}, \dot{v})$ of a point $\mathbf{p} = (u, v)$ on the image plane is described by:

$$\dot{\mathbf{p}} = L(\mathbf{p}, Z)\mathbf{v} \tag{2.13}$$

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} -\frac{f_x}{Z} & 0 & \frac{u-u_0}{Z} & \frac{(u-u_0)(v-v_0)}{f_y} & -\frac{f_x^2+(u-u_0)^2}{f_x} & v-v_0 \\ 0 & -\frac{f_y}{Z} & \frac{v-v_0}{Z} & \frac{f_y^2+(v-v_0)^2}{f_y} & -\frac{(u-u_0)(v-v_0)}{f_x} & -(u-u_0) \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}, \tag{2.14}$$

where f_x and f_y represent the focal lengths in pixels along the x and y axes, (u_0, v_0) is the principal point (optical center) in pixels, and Z denotes the depth of the point relative to the camera frame. The matrix L is known as the interaction matrix or image Jacobian. Equation 2.13 applies to a single point; for multiple points, these equations can be vertically concatenated. For instance, given N points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_N$ and their associated depths $Z_0, Z_1, \ldots, Z_N$, we have:

$$\begin{pmatrix} \dot{\mathbf{p}}_0 \\ \vdots \\ \dot{\mathbf{p}}_N \end{pmatrix} = \begin{pmatrix} L(\mathbf{p}_0, Z_0) \\ \vdots \\ L(\mathbf{p}_N, Z_N) \end{pmatrix} \mathbf{v}. \tag{2.15}$$

To compute the camera velocity from pixel velocities, the pseudo-inverse of the stacked interaction matrix is employed:

$$\mathbf{v} = \begin{pmatrix} L(\mathbf{p}_0, Z_0) \\ \vdots \\ L(\mathbf{p}_N, Z_N) \end{pmatrix}^+ \begin{pmatrix} \dot{\mathbf{p}}_0 \\ \vdots \\ \dot{\mathbf{p}}_N \end{pmatrix}. \tag{2.16}$$

At least three non-collinear feature points are necessary to ensure a well-defined interaction matrix. The pixel velocity is typically computed as $\dot{\mathbf{p}} = \lambda(\mathbf{p}^* - \mathbf{p})$, where $\mathbf{p}^*$ is the target point position in the image plane, and $\lambda \in (0, 1)$ regulates the convergence speed from the current position towards the target position. Combined with Eq. 2.16 we get:

$$\mathbf{v} = \lambda \begin{pmatrix} L(\mathbf{p}_0, Z_0) \\ \vdots \\ L(\mathbf{p}_N, Z_N) \end{pmatrix}^+ \begin{pmatrix} \mathbf{p}_0^* - \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_N^* - \mathbf{p}_N \end{pmatrix}. \tag{2.17}$$

Importantly, IBVS does not require explicit knowledge of the camera or object poses, relying solely on image-plane features. Although the interaction matrix includes the depth Z, IBVS demonstrates robustness to inaccuracies in depth estimation [21].

2.5 Discussion and Summary

This chapter reviewed the foundations relevant to robotic manipulation in the context of this thesis. Broadly, the methods can be divided into two categories: perception and control.

For perception, the primary focus is object pose estimation. Classical computer vision approaches are lightweight but often brittle, as they are sensitive to environmental conditions and perform poorly on textureless or symmetric objects. Deep learning–based methods address some of these limitations. Direct approaches attempt to regress object poses end-to-end from images, but require large amounts of ground-truth 6D pose data, which is costly to obtain. Indirect methods have emerged as a practical compromise: a neural network (typically CNN-based) detects 2D keypoints, and traditional algorithms such as PnP or ICP are then used to estimate the full 6D pose. This hybrid strategy combines the robustness of learning-based perception with the geometric reliability of classical pose estimation.

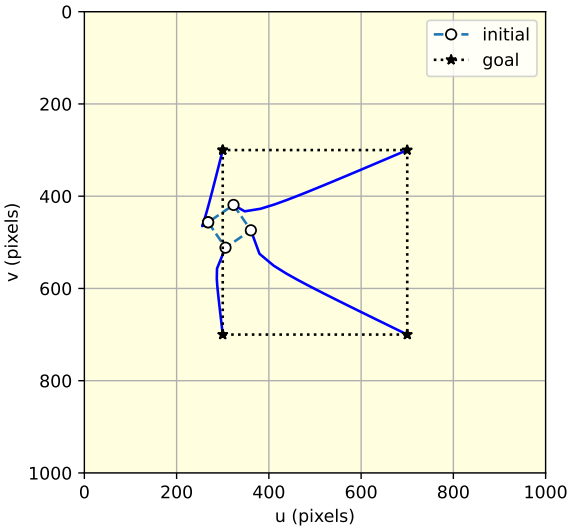


Figure 2.10: **Two different views of a square object: initial and goal.** Figure is reproduced using code from [21].

For control, PbD-based methods achieve strong sample efficiency by constraining the state space. However, jointly modeling position and orientation remains an open challenge, and new models are often required for each new task. In contrast, Transformer-based approaches demonstrate remarkable flexibility: a single model can learn diverse tasks across domains such as vision, language, and robotics. Yet, current Transformer-based methods are typically trained in an end-to-end fashion, demanding massive datasets that are impractical to collect in real-world robotic settings. These observations suggest the need for a middle ground between PbD and Transformer-based methods—one that preserves the sample efficiency of PbD while also leveraging the flexibility and generality of modern neural architectures. Developing such an approach forms the central motivation for this thesis.

Finally, visual servoing is employed in this work as a refinement module to adjust gripper poses before grasping or placing. Since object keypoints can be detected more reliably than full object poses in the proposed pipeline, IBVS is adopted to guide the robot using image-plane features during these precision-critical phases.

Chapter 3

High-Level Action Planning

Statement of Contribution

The research presented in this chapter was conducted in collaboration with Saeejith Nair, Peter Blouw, and Bryan Tripp. The simulated environment was co-implemented by Xueyang Yao and Saeejith Nair. Xueyang Yao, Peter Blouw, and Bryan Tripp conceived the experiments and the analyses that were carried out. Xueyang Yao conducted all the experiments and analyses under the supervision of Bryan Tripp. Bryan Tripp designed Figure 3.1. The remaining figures and tables were designed by Xueyang Yao.

> The content of this chapter has been previously published as:
>
> Xueyang Yao, Saeejith Nair, Peter Blouw, and Bryan Tripp, "Inferring symbols from demonstrations to support vector-symbolic planning in a robotic assembly task," in *Proceedings of ICDL 2020 – 1st SMILES (Sensorimotor Interaction, Language and Embodiment of Symbols) Workshop*, 2020.

The bolt–nut assembly task is first approached using a two-level hierarchical framework consisting of high-level action planning and low-level trajectory generation. This chapter focuses on the high-level action planning component, which is responsible for determining a sequence of abstract actions—such as placing the nut into the jig, screwing the bolt into the nut, and placing the assembled pair into a box—required to complete the multi-step assembly task. These symbolic actions provide structured guidance to the low-level trajectory planner, which is discussed in the following chapter. At the beginning of the

project, a biologically plausible system was employed, given its effectiveness under limited training data. However, the remainder of the thesis shifts focus toward achieving practical performance under data constraints, independent of biological plausibility.

3.1 Introduction

Many of the shared concepts that allow us to communicate with each other are grounded in our experiences. Deep reinforcement learning [6] allows artificial agents to convert experience into sophisticated behavior, via the development of task-related internal representations. However, these representations are not explicit or discrete enough to support communication. In contrast, classical planning [137] achieves sophisticated behavior using explicit symbolic representations that are compatible with communication and instruction, but the symbols in these systems are grounded manually.

Previously, Blouw et al.[11] introduced a model of vector-symbolic planning implemented as a recurrent network of spiking neurons. Vector-symbolic architectures represent symbols using high-dimensional vectors and support the binding of multiple concept vectors to form new, composite representations. The model was developed using the Semantic Pointer Architecture (SPA)[30], which integrates vector-symbolic reasoning with biologically plausible neural constraints. In this framework, semantic pointers were used to encode both states and actions, enabling the system to reliably generate novel action sequences to achieve specified goals. The approach demonstrated scalability to environments of realistic complexity, including scenarios involving tens of thousands of objects and possible actions. Since the model was realized entirely in spiking neurons, it presents a plausible account of how human vector-symbolic representations may be neurally instantiated in everyday cognition. However, that prior work did not address the grounding of these symbolic representations in sensory data.

Recent work in robotics suggests an approach to filling that gap. Many studies have dealt with the problem of robotic learning from human demonstration (reviewed by [111, 148, 4]). Some of these studies have used symbol-based systems, along with symbolic planners [28, 87, 75] or motion grammars [23]. In such systems, heuristics are used to identify relevant states, and typically (e.g. [28, 87]) clustering is used to group such states so that they can be represented symbolically. In the present study, a system is developed and integrated with a vector-symbolic planning framework—a simplified version of the previously proposed model—to control a simulated assembly task. This is a small step toward grounding the biologically plausible model of action-based representations in a non-symbolic sensory stream. Given both the importance of action-based representations

in contemporary theories of linguistic cognition [44], and the role that vector-symbolic representations often play in these theories [125, 57], the proposed approach helps to set the stage for grounded yet structured models of the representations involved in higher level cognitive processes.

3.2 Methods

Previous work [11] suggested the biological plausibility of STRIPS-style planning in daily activities. While the planner used in this study does not generate globally optimal plans, it is capable of producing viable short-term plans within approximately 200 ms of simulated time, so that re-planning could occur frequently during a complex activity (e.g. preparing a meal). The goal of the present work is to experiment with a PbD approach, to produce action representations that consist of preconditions and effects, compatible with the planning model. A variety of related approaches has been taken in robotics. Perhaps most closely related, [28] used PbD to infer relevant discrete states, and constraints on the ordering of state transitions. In contrast to that prior work, the present study explores a different task and produces explicit action representations—rather than ordering constraints—that are consistent with the approach described in [11]. These representations could also perhaps, in future work, be re-used in multiple tasks, and/or used as pointers to lower-level action-specific skills developed through practice. Finally, these representations are integrated and evaluated within a vector-symbolic planning framework.

3.2.1 Recording and Clustering States

In PbD systems that operate on discrete object states, the states are recorded only at certain times, and they are clustered to produce models of state categories. A key design decision is when to record object states for clustering. Robotic manipulation tasks are considered, with object states recorded at the moments when the gripper establishes contact with an object (e.g., to pick it up) and when contact is released (e.g., to place it down). This approach could be generalized to include inflection points in the gripper trajectory, if the trajectory does not take a direct path between start and end states.

The clustering approach is another key decision. A number of past studies have used k-means clustering with an Euclidean distance metric. A limitation of that approach is that it assumes hyper-spherical clusters, whereas practical states are often relatively invariant to certain dimensions. For example, the hooks of coat hangers have tight clustering around the

bar, but widely varied positions along the bar. Sometimes, an object's precise orientation is an important part of a given state (e.g. when a bolt is inserted into a nut), and sometimes the orientation is a highly variable property of a given state (e.g. when a bolt is a bin). In more naturalistic scenarios, even the states of simple objects may have many dimensions, such as positions and orientations relative to various other objects, as well as temperature, color, etc., and many of these dimensions may be irrelevant to a given goal. To address this, a distance metric is sought in which states that differ in only a few dimensions yield small distances. For this purpose, p-norms of the differences between positions are used.

$$d_{u,v} = \left(\sum_{i=1}^{n} |u_i - v_i|^p \right)^{\frac{1}{p}}, \tag{3.1}$$

with $p < 0$, where n is the state dimension. Figure 3.1 further motivates this choice.

3.2.2 Vector-Symbolic Architecture

VSA [37] have been applied to a broad range of tasks related to manipulating structured information. In a VSA, concepts are represented by high dimensional vectors. Atomic concepts are typically represented as random vectors (which are nearly orthogonal in high dimensions), and more complex concepts are composed via addition and binding operations. For example, symbols *RED* and *BALL* could be represented as random 256-dimensional vectors, and the concept *RED_BALL* could then be composed from the *RED* and *BALL* vectors with a binding operation. Here Holographic reduced representation (HRR) [103] is used, a kind of VSA that uses circular convolution (written as $\circledast$) as a binding operator. Using HRRs, the concept *RED_BALL* can be represented as $RED\_BALL = RED \circledast BALL$, and the objects in a simple scene might be represented as $SCENE = RED \circledast BALL + BLUE \circledast BLOCK$. Importantly, an inverse operation can be used to query information from a compound HRR. For example, due to the near-orthogonality of the high-dimensional vectors, the result of $SCENE \circledast BALL^{-1}$ will yield the vector for *RED* approximately, answering the question, "What color is the ball in the scene?"

3.2.3 Assembly Task

The proposed approach was evaluated in a robotic assembly task simulated using PyBullet (Figure 3.2). The task begins with nuts and bolts on a table, and the goal is to attach a

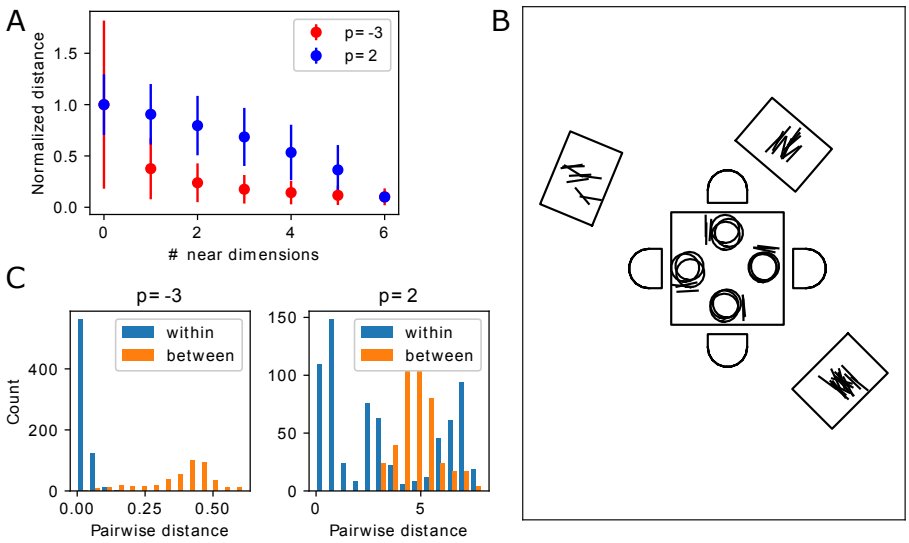


Figure 3.1: **Use p-norms with negative exponents to cluster states**. A: With a negative exponent, the distance between states is fairly low even if the states are only similar in one or two dimensions. To illustrate, pairwise distances (mean +/- SD) between 6D states are shown, with tight Gaussian distributions ($\sigma = .05$) in some dimensions, and broad distributions ($\sigma = .5$) in other dimensions. Using the standard Euclidean norm ($p = 2$; blue), pairwise distances are large even if all the states are close in three or four dimensions. However, with $p = -3$, pairwise distances are lower when the states are close in only one or two dimensions. B: To further illustrate, a simplified simulation of a table-setting task is presented. Positions of plates, knives, and a cart are overlaid from three example tables. Given such observations, the goal is for the robot to recognize that knives are sometimes located on the cart and sometimes placed next to plates, oriented orthogonally to the table edge. C: Pairwise distances between knife states in the table-setting example. The 8D state vector includes positions and orientations relative to the cart and the closest plate and table edge. Using a Euclidean norm with $p = 2$, pairwise inter-state distances within a group (i.e. on the cart or beside a plate) are similar to pairwise distances between groups (right), but these inter-state distances are well separated with $p = -3$ (left).

nut and bolt together. A jig is available to ensure that the nuts and bolts are precisely vertical, so that they fit together properly. Figure 3.2 illustrates the PyBullet simulation. Because this work is preliminary and exploratory, several simplifications were made to better focus on the most relevant aspects of the system. In particular, ground-truth object type, position, and orientation data from the simulator were used. This is a reasonable simplification, insofar as state-of-the-art methods of estimating these from images are reasonably accurate. For grasping the nuts and bolts, known-good grasp points with opposing flat surfaces were used, because simple heuristics are often effective for choosing grasp points on isolated objects from vision, e.g. [92]. Finally, the nut and bolt were simplified by removing the threads, and the task was considered complete when the bolt shaft was inserted through the nut. This eliminated the need to simulate the physics of thread interactions, and to monitor forces to detect thread binding. Probably the main effect of these simplifications was to reduce the error rate. The simulator did occasionally drop a nut or bolt, prompting the system to re-plan, but re-planning would have been needed more often if there had been additional sensory and motor errors. The process of formulating vector-symbolic representations for the task involves two main stages, demonstration and model building, which are described next.

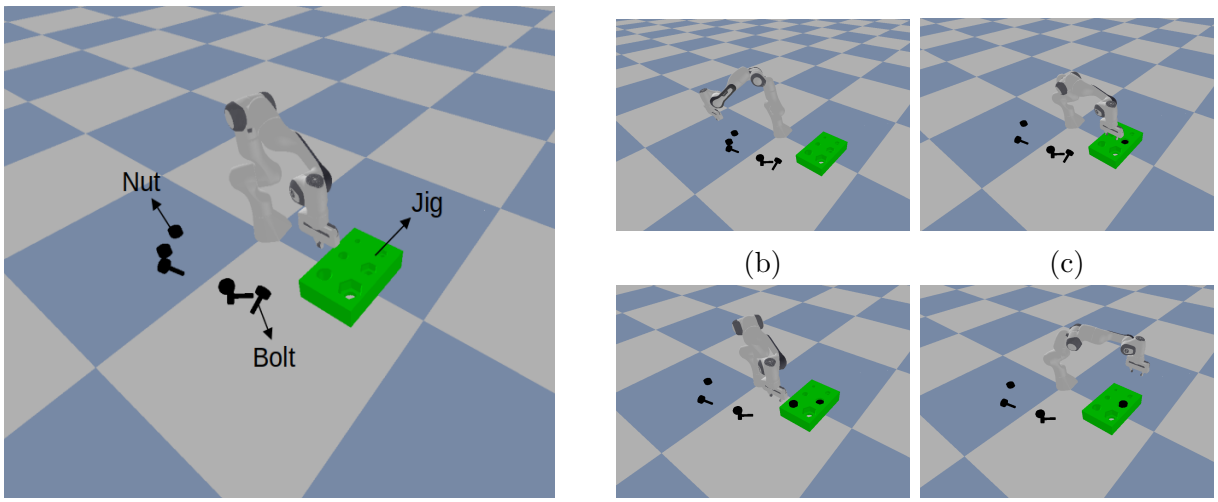


Figure 3.2: **Simulation environment and main stages of the robotic assembly task**. (a) A view of the simulation environment with main objects labeled. (b) Bolts and nuts are on the table. (c) One of the nuts is placed in the jig. (d) One of the bolts is placed in the jig. (e) Task complete: bolt and nut are assembled.

3.2.4 Collecting Demonstration Data

In the demonstration stage, the system is shown examples of a 7-DOF robotic arm performing the assembly task. In PbD systems, robots are often controlled by humans to collect demonstration data; however, for efficiency, the task was scripted in this study. The task requires the following actions (note that the first and second actions can be performed in either order):

1. If there is a nut on the table and the nut slot in the jig is empty, the robot should place a nut in the jig.
2. If there is a bolt on the table and the bolt slot in the jig is empty, the robot should place a bolt in the jig (the bolt shaft protrudes down through a hole in the jig, so that gravity ensures the bolt is vertical).
3. If the jig contains both a nut and a bolt in their respective slots, the robot should assemble the bolt into the nut.

3.2.5 Building Action Representations

After a few demonstrations are complete, the next step is to cluster the states. 3D positions were used to cluster the states of each type of object, and positions and orientations of each cluster were modeled with 6D Gaussians. Although some states are tightly clustered, the nuts and bolts have wide horizontal distributions when they are on the table. Following the motivation discussed earlier, the distance between a pair of positions was defined as the p-norm of the difference between positions, using $p = -2$. This resulted in the nuts and bolts on the table joining a single cluster each. Given n 3D positions, a $n \times n$ distance matrix was computed between pairs of states. A bottom-up hierarchical clustering algorithm [98] was applied to the distance matrix, recursively merging together pairs of points that minimally increase the average pre-computed linkage. No hints are provided regarding the expected number of clusters or the size of each cluster. Instead, the method generates clusters of points satisfying a given distance threshold. Experimental results on randomly sampled data batches of sizes ranging from just 15 points to 300 points showed that the method could generate perfectly accurate clusters using a threshold of (1.169 ± 0.001)mm. However, perfect accuracy is not necessary as robust estimates of the cluster statistics are used.

The discrete high-level states of the nuts and bolts, used in planning, correspond to the resulting clusters. For each cluster, a robust mean state was computed and saved after

discarding the points in the top and bottom decile of each dimension. The position means were used to classify unseen data points during the planning stage. Both the position and orientation means were used to determine the target configurations of the robot when executing an action.

After learning the possible states of each object type, the system builds action representations, associating one action with each state that occurs at the end of gripper contact. For example, since a nut is sometimes in the jig when the gripper releases it, a corresponding action is defined (one might call it, $PUT\_NUT\_IN\_JIG$). For each action in the demonstration data, a global state vector G is then obtained of length m before and after each action, where m is the total number of states across all object types (in this task, $m = 5$). Each element in G is a binary quantity indicating whether or not there is an object in the scene which satisfies the corresponding state. Table 3.1 shows an example of a global state vector for the action $PUT\_NUT\_IN\_JIG$, when there is a nut on the table.

States	Before Action	After Action
NUT_ON_TABLE	1	0
BOLT_ON_TABLE	0	0
NUT_IN_JIG	0	1
BOLT_IN_JIG	0	0
BOLT_ASSEMBLED	0	0

Table 3.1: **Global state vectors before and after the action** $PUT\_NUT\_IN\_JIG$ **for demonstration 1**. The value assigned to each local state is 1 if an object exists in that state, and 0 otherwise. The state labels in the table have been added manually to help interpretation (they are not part of the system).

The global state is important when an action has preconditions related to multiple objects, as before the final assembly step when both a nut and bolt must be in the jig. Given the global state vectors, a set of preconditions as well as an associative memory is built. The preconditions indicate the object states necessary for an action to be executed and can be derived from the enabled bits of the global state vectors at the beginning of each instance of an action. Table 3.1 shows an example in which the state *NUT_ON_TABLE* preceeded the action *PUT_NUT_IN_JIG*. However, as seen in Table 3.2, the states {*NUT_ON_TABLE, BOLT_ON_TABLE*} were present before a different instance of the same action. It is evident that *BOLT_ON_TABLE* is not a requirement for this action. To address such coincidences, the preconditions for an action are found by $argmax\,(\sum_{i=1}^{n} G_i)$, where n is the number of demonstrations, and G_i is the global state vector preceding the i^{th} instance

of the action. This way, only the states that consistently occur before an action are interpreted as its preconditions. It is worth noting that $argmax\,(\sum_{i=1}^{n} G_i)$ might yield multiple solutions, which is reasonable because the one action could have multiple preconditions (e.g. the preconditions of action *ASSEMBLE* is *NUT_IN_JIG* and *BOLT_IN_JIG*). While an action's preconditions were defined based on states consistently present prior to its execution, an alternative approach could also consider states that are consistently absent as part of the precondition set. This would allow the system to learn, for example, that the bolt part of the jig must be empty before putting a bolt in it. Instead, an additional heuristic was applied: an object cannot be placed into a position that is already occupied.

After the true preconditions were obtained, two HRRs, A_s and A_d were then defined for each action. A_s was a shallow representation, a randomly assigned 256-dimensional vector that identified the action A. A_d was the corresponding deep representation, which had the same dimension as A_s but contained information about its preconditions. For example, if the precondition of action *PUT_NUT_IN_JIG* were found to be *NUT_ON_TABLE*, the deep representation of action *PUT_NUT_IN_JIG* would be defined as: $PUT\_NUT\_IN\_JIG_d = PRECONDITION_s \circledast NUT\_ON\_TABLE_s$.

States	Before Action	After Action
NUT_ON_TABLE	1	0
BOLT_ON_TABLE	1	1
NUT_IN_JIG	0	1
BOLT_IN_JIG	0	0
BOLT_ASSEMBLED	0	0

Table 3.2: **Global state vectors before and after the action** *PUT_NUT_IN_JIG* **for demonstration 2**. Note the unnecessary local state *Bolt on Table*.

The associative memory is a dictionary that maps an object's state to the action which caused it, and can be derived from the transition between two consecutive global state vectors. For example, in Table 3.2, the value of the *NUT_IN_JIG* changes from 0 to 1, which indicates that the effect of action *PUT_NUT_IN_JIG* is that the local state *NUT_IN_JIG* becomes enabled. However, a single demonstration would not be robust to coincident events, e.g. actions by other people in the environment. Table 3.3 shows state transition for the action *PUT_NUT_IN_JIG* where a bolt was put in the jig at the same time by a person. It is obvious that *BOLT_IN_JIG* should not be the effect of the action *PUT_NUT_IN_JIG*. To address this, the index to the associative memory was defined as $argmax\,(\sum_{i=1}^{n}(H_i - G_i))$, where n is the number of demonstrations, G is the global state

vector preceding an action and H is the global state vector after an action. Given the 3 demonstrations shown in Table 3.1-3.3, the system would derive that the effect of the action *PUT_NUT_IN_JIG* is *NUT_IN_JIG*. In summary, task demonstrations were used to build vector-symbolic representations of states and actions, and an associative memory that maps desired states to actions that lead to those states.

Labels	Before Action	After Action
NUT_ON_TABLE	1	0
BOLT_ON_TABLE	0	0
NUT_IN_JIG	0	1
BOLT_IN_JIG	0	1
BOLT_ASSEMBLED	0	0

Table 3.3: **Global state vector before and after the action *PUT_NUT_IN_JIG* for demonstration 3**. Note that a bolt was put in the jig by another subject at the same time.

3.2.6 Planning process

Using these state and action representations, and the associative memory, the planning process is executed as follows: For each step in the simulation, the planner assigns each object to the nearest state (the state with center at the shortest p-norm distance from the object's position). After all objects are clustered and the global state vector G is built, the planner considers the goal (e.g. *BOLT_ASSEMBLED*). First, the planner checks G to see if the goal is already met. If not, an action that leads to the goal is queried from the associative memory. Then, the planner finds the preconditions of the action by calculating $action_d \circledast PRECONDITION_s^{-1}$. The action is executed only if all of its preconditions are satisfied in G. Every unsatisfied precondition becomes a new goal, and the planning process repeats. In [11], a new goal was obtained at each step by subtracting the action's effect from the *GOAL* vector and adding its preconditions, whereas in the present simplified system, the preconditions replace the previous goal. Another difference is that in [11], actions were added to a stack (implemented as a recurrent network) during planning, whereas in the present system, only the earliest action in the plan is retained, with re-planning performed at every step.

3.3 Results

3.3.1 Robustness of clustering

Figure 3.3 shows how the clustering algorithm identifies different states given the objects' position information. The state labels were added manually to help interpretation. The algorithm tended to cluster data points into one group as long as they were close in any of the dimensions. For instance, nuts on the table were clustered into one group because their z coordinates were similar, even though their x, y coordinates varied. After clustering, means and covariance matrices were calculated to build a Gaussian distribution model for each of the state clusters. These models were then tested on a test set that contained 602 unseen data points. As can be seen in Figure 3.4, the generated state models yielded 100% success rate on the test set.

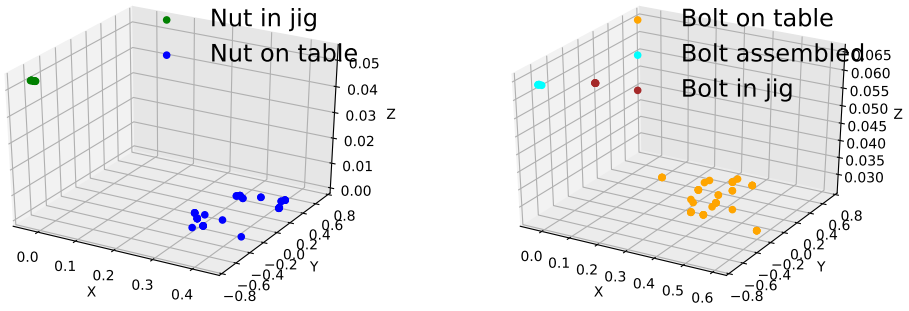


Figure 3.3: **Clusters of position information for bolts and nuts obtained from five demonstrations**. Left: Positions of the nuts are clustered into two states. Right: Positions of the bolts are clustered into three states.

3.3.2 Robustness of the HRR structures

As described in Section 3.2, actions were represented with HRRs: one shallow representation and one deep representation. States, on the other hand, were represented with only one HRR because different states were clustered based only on position information. However, such information is generally not sufficient to describe states in a task agnostic manner. To set the stage for more generic state representations that reuse shared features,

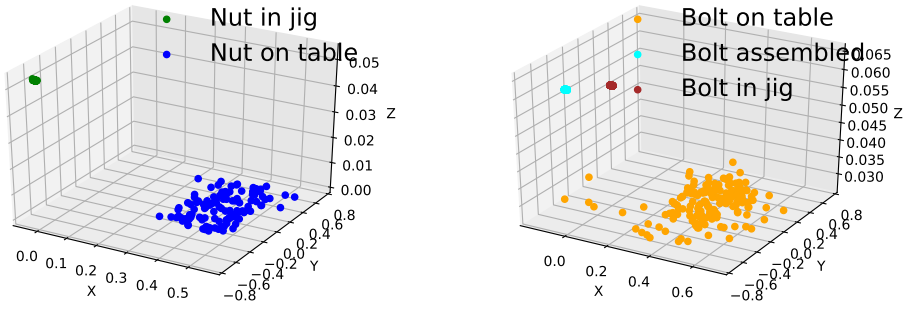


Figure 3.4: **Test results for the state models obtained from clustering.** The test set contained 602 data points achieving 100% success rate.

an approach was explored in which a shallow HRR was used to identify a state, and a deep HRR was defined based on the object kind (e.g. *BOLT*, *NUT*), location of the object (e.g. *TABLE*, *JIG*), and the assembly state of the object (e.g. *NOT_READY*, *READY*, *ASSEMBLED*).

Three variants of this approach were tested. In the first, the deep HRR for a state took the form $object \circledast (location + assemble\_state)$, while in the second this deep HRR took the form $OBJECT_s \circledast object + LOCATION_s \circledast location + ASSEMBLE\_STATE_s \circledast assemble\_state$. In the third variant, the deep HRR took the form $object \circledast (location + assemble\_state) + EFFECT\_OF_s \circledast action$, where the action was the cause of the state being represented. The planning success rate of a system using each structural variant as a function of representational noise is shown in n Table 3.4. A plan is considered successful only if the correct action is executed at every step. The first structural variant, which contains fewer components reused across representation, yields a higher success rate as expected. The concepts were represented as 256-dimensional vectors in the test.

3.3.3 Minimal Demonstrations and Failure Modes

The system can reliably model and plan the task after a single demonstration. However, the system fails if a test scenario differs too much from demonstration scenarios. In particular, if there are always multiple nuts or bolts at demonstration time, the system infers that at least one nut and bolt has to be on the table, as a precondition to any action. This can be avoided by demonstrating the task with only one nut and one bolt. However, this

	$\sigma = 0.5$	$\sigma = 0.4$	$\sigma = 0.3$	$\sigma = 0.2$	$\sigma = 0.1$
Structure 1	80.5%	94.6%	96.9%	98.3%	98.8%
Structure 2	44.9%	68.3%	83.1%	90.6%	95.6%
Structure 3	13.5%	40.0%	68.7%	83.8%	91.5%

Table 3.4: **Success rate of the planner with different HRR structures with respect to different noise levels.** σ is the standard deviation of the Gaussian noise added. The standard deviation of the clean HRR is 0.0625.

limitation should be addressed in future work.

3.4 Discussion

This work investigated the use of programming-by-demonstration methods for grounding vector-symbolic representations. The results show that this approach enables efficient learning of state and action representations that support robust vector-symbolic planning, using only a small number of task demonstrations. While other work has often used k-means clustering of states, along with Euclidean distance, we found that p-norms with negative exponents can naturally account for dimensions that are semantically irrelevant to a state.

Limitations of the proposed approach largely fall into two categories. The first concerns realistic sensory estimation of object positions and orientations, which is a central problem that needs to be solved prior to transitioning out of simulation and into a real-world robotic device. The second limitation concerns generalizing to a wider range of tasks involving more complicated action sequences with variably structured preconditions.

The use of vector-symbolic representations in the proposed planner may aid generalization in such cases, as these representations support both soft pattern-matching and structural comparisons that can aid in reasoning over action preconditions. Additionally, these representations could later be extended to include linguistic elements, such that in a scenario of the sort shown in Figure 3.1, the planning system could cue the movement of a utensil with a term like "beside a plate". Exploring such extensions will be a focus for future work.

This chapter explores the high-level action planning based on sensory data, serving as a foundational step in the two-level framework for the robotic bolt–nut assembly task.

Chapter 4

Task-parameterized Movement Primitives for low-level trajectory planning

Statement of Contribution

The research presented in this chapter was conducted in collaboration with Yinghan Chen, and Bryan Tripp. Xueyang Yao and Bryan Tripp conceived the experiments and the analyses that were carried out. Xueyang Yao and Yinghan Chen collaborated on data collection. Yinghan Chen developed the outlier detection algorithm and the Dynamic Time Warping module, while all remaining algorithms were implemented by Xueyang Yao. Analyses were performed by Xueyang Yao under the supervision of Bryan Tripp. All figures and tables were prepared by Xueyang Yao.

The content of this chapter has been previously published as:

© 2023 IEEE. Reprinted, with permission, from Xueyang Yao, Yinghan Chen, and Bryan Tripp, "Improved generalization of probabilistic movement primitives for manipulation trajectories," in *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 287–294, January 2023.

In the previous chapter, it was shown that p-norms are effective for state clustering and that VSA provides a suitable framework for action planning. This combination enables the

decomposition of the long-horizon, multi-step assembly task into manageable subtasks. In this chapter, the focus shifts to modeling low-level trajectories with limited demonstration data.

4.1 Introduction

PbD [10] is an imitation learning approach that typically requires less than ten training examples per task. For example, [18] showed that a robot could learn a pick-and-place task on the basis of 4 to 7 examples. Similarly, in the work of [50], a collaborative object transportation task was learned using only 5 demonstrations. These approaches are examples of few-shot learning [135] that rely on engineering design to restrict the solution space. In contrast, reinforcement learning does not require human demonstrations, but it may require tens of thousands of trials, even for simple and narrow tasks such as bin picking.

In recent years, Movement Primitives (MPs) have garnered significant attention for their success in manipulation tasks. Dynamic Movement Primitives (DMPs) [120, 53] model a trajectory as a spring-damper dynamic system with a nonlinear forcing term. DMPs excel in generating smooth trajectories from any initial state. ProMPs [95, 94] directly learn a probabilistic model over the trajectories. ProMPs not only provide an estimate of the underlying trajectory derived from multiple demonstrations but also capture its variability through the covariance matrix. Probabilistic Dynamic Movement Primitives (ProDMPs) [73] provide a unified framework that combines the principles of both DMPs and ProMPs. ProDMPs achieve this by recovering a linear basis-function representation for trajectories through the solution of a dynamical system. Notably, DMPs, ProMPs, and ProDMPs all share the requirement for manually defining basis functions. KMPs [50] and Gaussian Process (GP) based methods[55, 3] alleviate the need for manual basis function definition by employing kernel techniques. It is worth noting that all of the aforementioned methods achieve adaptation by relying on via-points, a concept that may not align naturally with real-life manipulation tasks. Determining the appropriate via-points typically involves expert knowledge and heuristics.

TP-GMMs [17] employ a distinctive approach for adaptation. TP-GMMs leverage multiple reference frames to represent the end-effector's trajectory. Within each local reference frame, a statistical model is learned, and the global trajectory model is obtained by multiplicatively combining the local models. This adaptation method aligns more naturally with manipulation tasks since it allows TP-GMMs to adapt based on the poses of objects. These poses can be acquired using a vision module or by attaching markers to objects,

eliminating the need for additional heuristics. Furthermore, TP-GMMs show robust extrapolation capabilities by encoding movements from multiple reference frames, enabling it to adapt to object poses beyond the training distribution. However, it is important to note the time-based TP-GMMs approach is unsuitable for cyclical movements and struggles to handle tasks with multiple modes, such as multiple paths around an obstacle. In contrast, dynamic-based TP-GMMs model the joint distribution of dynamic features, including position, velocity, and acceleration. This enables the dynamic-based TP-GMMs to effectively handle cyclical movements and trajectories with multiple options. [144] extends the TP-GMMs approach by modeling quaternions on a 3-dimensional Riemannian manifold that respects the geometry of quaternion space. However, computing Gaussian product and conditioning on a Riemannian manifold requires iterative computations on the tangent space.

Inspired by TP-GMMs, Local Kernelized Movement Primitives [50] were introduced to enhance KMPs' extrapolation capabilities. Furthermore, KMPs respecting the geometry of quaternions were introduced in [49]. However, akin to time-based TP-GMMs, KMPs are unable to infer and model multiple modes from the dataset.

Another family of approaches is to use Neural Networks (NNs) to abstract the task parameters from high-dimensional modalities, e.g. scene images, to support end-to-end trajectory learning. These approaches [121, 140, 104, 127] eliminate the need for manually engineered task parameters. However, the task of inferring parameters from a restricted dataset is inherently challenging, and modeling trajectories atop it presents an even greater difficulty. In [96], representation learning and behavior learning are decoupled. When presented with a new scene image, this approach identifies the nearest neighbors in the representation space, and the agent's behavior is obtained by a weighted average of actions from these neighbors. However, interpolating between nearest neighbors poses challenges when tested outside the training distribution[43].

Our proposed method, TP-ProMPs, addresses limitations of previous approaches by introducing multiple local reference frames to the ProMPs framework. By combining the strengths of ProMPs and TP-GMMs, TP-ProMPs offers several key advantages, including extrapolation capability, adaption to different task parameters, and via-point modulation. Furthermore, TP-ProMPs demonstrates accurate learning of 6D end effector trajectories. To enhance the modeling of time-based trajectories with multiple modes, we extend TP-ProMPs by representing weights as a mixture of Gaussian distributions. While previous works have explored this idea [31], our method introduces a more robust approach. In addition, we incorporate additional reference frames that encompass pairs of objects. We show that this extension yields more accurate trajectory predictions in aiming tasks. Finally, departing from recent practice in PbD, we do not affix markers to objects but instead sense

their configurations using deep networks, eliminating a physical task constraint and broadening the range of tasks our method can handle. This approach introduces vision-related errors. However, these additional errors are consistent across different PbD methods, allowing a fair comparison. Overall, the results we report here are practical lower bounds on the performance of these methods.

We make our demonstration datasets public to facilitate future work (https://github.com/x35yao/TP-ProMP/tree/main/data).

4.2 Methods

4.2.1 Tasks

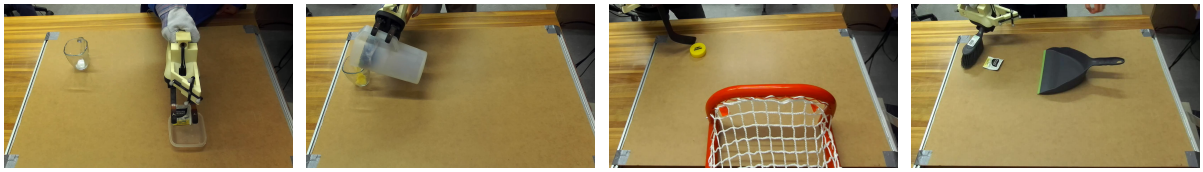
We evaluated PbD methods with 4 different tasks:

- Pick-and-place task testing extrapolation performance.
- Water pouring task consisting of different modes of pouring from both the left and right of the cup. This task evaluated the methods' ability to accurately model orientation and its resilience to mode collapse.
- Shooting hockey puck task involves modeling the motion of the gripper holding a hockey stick and striking the puck into a net from various directions.
- Sweeping task involving modeling the motion of the gripper holding broom and cleaning rubbish into a dustpan.

Figure 4.1 shows images of these tasks.

4.2.2 Data collection

The workspace was instrumented with both a Zed2 stereo camera (used to sense object positions and orientations) and an NDI Polaris optical motion tracker (used to sense gripper position and orientation). Two multi-marker "tools" were mounted to the gripper handle at different positions to avoid occlusion such that the motion tracker could track the 6DOF configuration of the gripper [93]. The camera and motion tracker were synchronized and both signals were resampled to a common time step of 0.08 seconds. DeepLabCut [82]



(a) Pick and place (b) Pour water (c) Shoot Puck (d) Sweep

Figure 4.1: **Demonstration of the tasks.** (a) The pick-and-place task involves placing the gripper above a teabag, grasping and moving the gripper above a cup, and placing the teabag into the cup. (b) The water-pouring task involves picking up a pitcher filled with water, pouring water into a cup, and then placing the pitcher back down on a table. (c) The shooting-puck task where the demonstrator shoots a puck into a net on the table. (d) The sweeping task where the demonstrator cleans a rubbish into a dustpan.

was employed to estimate the two-dimensional positions of object parts in the image space of both the left and right cameras. We derived their three-dimensional coordinates by triangulating these positions.

We used the following procedure to robustly estimate object poses even when certain parts were occluded. Prior to each task, the relevant objects were placed on the table with all object parts visible. The triangulated 3D object-part positions, denoted as $\boldsymbol{p_i}_{i=1}^{N}$, were used to create an object template. The object center was determined as the mean of all object-part positions, and a coordinate system was assigned to each object template (see Figure 4.2 for an example). The pose of each object template in the global reference frame was represented by a rotation matrix $\boldsymbol{A_1}$ and a translation vector $\boldsymbol{b_1}$. By aligning the template's axes with the global reference frame, $\boldsymbol{A_1}$ was simplified as an identity matrix and $\boldsymbol{b_1}$ represented the center position.

During each demonstration, a different set of object-part positions $\boldsymbol{p'_i}_{i=1}^{N'}$ was obtained, with $N' \leq N$. Since the correspondence between $\boldsymbol{p}$ and $\boldsymbol{p'}$ is known, solving the alignment problem is equivalent to approximating the rotation matrix $\boldsymbol{A_2}$ and translation vector $\boldsymbol{b_2}$ such that (see Section 2.3.1):

$$err = \Sigma_{i=1}^{N'} ||\boldsymbol{p_i}' - (\boldsymbol{A_2 p_i} + \boldsymbol{b_2})||^2 \tag{4.1}$$

$\boldsymbol{A_2}$ and $\boldsymbol{b_2}$ is the active transformation that moves the template object-part positions to match with the current object-part positions (See Appendix A.3). We obtained the object's pose (task parameters) in the global reference frame as $\boldsymbol{A} = \boldsymbol{A_1 A_2}$ and $\boldsymbol{b} = \boldsymbol{A_1 b_2} + \boldsymbol{b_1}$. These task parameters were used to convert trajectories between local and global reference

frames as described in [17]. For tasks where the object's orientation did not significantly impact the trajectory modeling, $\boldsymbol{A}$ was simplified as an identity matrix.

4.2.3 Preprocessing

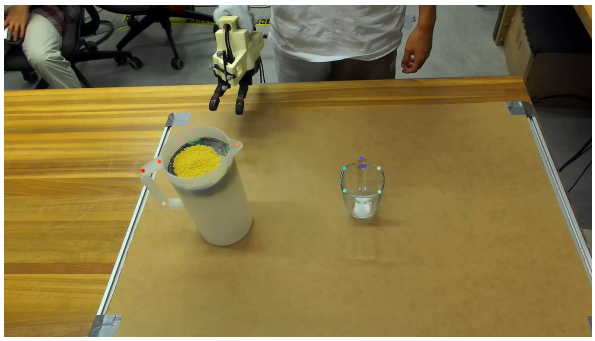
Dynamic time warping was used to realign the trajectories and normalize the durations of different demonstrations of the same task. The demonstration with the median duration was used as the reference demonstration that others were realigned around. The minimized cost metric was the gripper's speed difference versus that in the reference demonstration. To avoid extreme jumps in trajectory, step sizes were constrained within 1/2 and 2 times corresponding step sizes in the reference trajectory [89].

To deal with outlier demonstrations from noisy sensors, outliers were removed based on whether the corresponding trajectory's start and end points deviated significantly from the mean of the current set of demonstrations by standard deviation threshold of 3. Due to the more extreme outliers skewing the distributions, this was repeated after the detected outliers were removed and halted when the next iterations generated no new outliers.

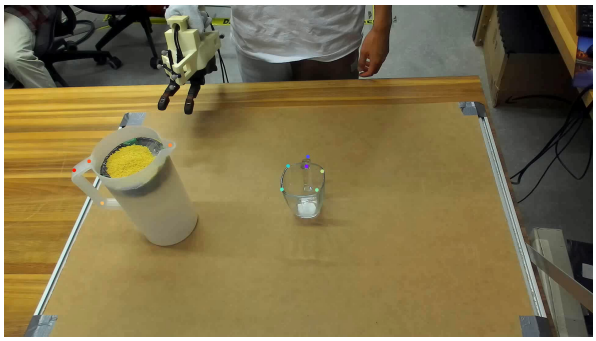
4.2.4 Task-parameterized Probabilistic Movement Primitive (TP-ProMPs)

Learning from Demonstrations

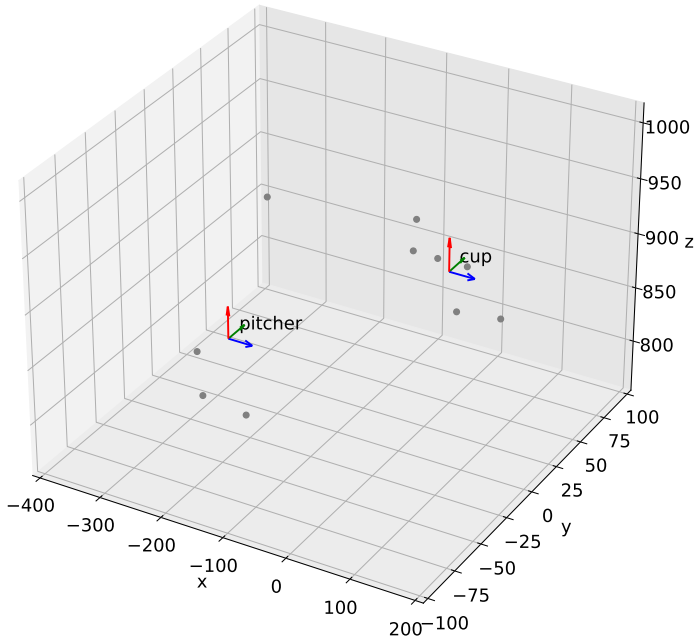
Our methods are based on both ProMPs and TP-GMMs. A trajectory $\tau = \{\boldsymbol{x}_t\}_{t=1}^{T}$ represents a trajectory in the global reference frame at different moments in time, where $\boldsymbol{x}_t$ is a D dimensional vector. In other words, a trajectory is represented as the matrix $\tau \in \mathbb{R}^{D \times T}$. In this paper, if both the position and orientation are modeled, $\boldsymbol{x}_t$ is a 7D dimensional vector, where the first three dimensions represent the position of the end-effector and the remaining four dimensions represent the quaternions that describe its orientation. When only the end-effector's position is modeled, $\boldsymbol{x}_t$ is simplified to a three-dimensional vector. We assigned R local reference frames to objects in the scene that are related to the task. For the jth reference frame, we obtained the task parameters $\boldsymbol{A_j}$ and $\boldsymbol{b_j}$ following the procedure outlined in 4.2.2. By utilizing these task parameters, we converted the trajectory from the global reference frame to the jth local object reference frame, resulting in $\boldsymbol{x}_t^{(j)}$, as explained in [17]. Here, the superscript j signifies that the trajectory is expressed in the jth reference frame.



(a) Left camera view



(b) Right camera view



(c) 3D objects' positions

Figure 4.2: **Object ose estimation Using stereo vision Top:** Marker positions predicted by DeepLabCut in the image planes of the left and right cameras. **Bottom:** Triangulated 3D positions of objects in the camera reference frame. Local reference frames are defined at the centroid of the corresponding 3D marker positions.

Trajectories from all reference frames were then stacked to get $\tau' = \{\boldsymbol{\chi}_t\}_{t=1}^{T}$, where $\boldsymbol{\chi}_t = \begin{bmatrix} x_{1,t}^1 & \dots & x_{D,t}^1 & \dots & x_{1,t}^R & \dots & x_{D,t}^R \end{bmatrix}^\top$ combines trajectories from all reference frames and $x_{d,t}^j$ denotes the gripper trajectory value for the dth dimension in the jth reference frame. $\tau' \in \mathbb{R}^{DR \times T}$ can be interpreted as a trajectory with $M = DR$ dimensions. We then fit τ' to a ProMPs model. As described in [95], ProMP represents a trajectory with a weight vector and variability of the given demonstrations is modelled in the weight space. The combined weight vector $\boldsymbol{\omega} = \begin{bmatrix} \boldsymbol{\omega}_1^\top, & \boldsymbol{\omega}_2^\top, & \dots & , \boldsymbol{\omega}_M^\top \end{bmatrix}^\top$ for a trajectory consists of M weight vectors $\boldsymbol{\omega}_m$ that correspond to each dimension m. The trajectory at time t given the weight vector $\boldsymbol{\omega}$ is defined as

$$\boldsymbol{\chi}_t = \boldsymbol{\Phi}_t \boldsymbol{\omega} + \boldsymbol{\epsilon}_\chi, \tag{4.2}$$

where, $\boldsymbol{\epsilon}_\chi \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\epsilon)$ is Gaussian white noise, $\boldsymbol{\Phi}_t$ is an $M \times KM$ matrix defined as

$$\boldsymbol{\Phi}_t = \begin{bmatrix} \boldsymbol{\phi}_1(t) & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \boldsymbol{\phi}_M(t) \end{bmatrix}, \tag{4.3}$$

$\boldsymbol{\phi}_m(t)$ defines the K dimensional time-dependent basis vector for the mth dimension and K defines the number of basis functions. In this work, we simplified $\boldsymbol{\Sigma}_\epsilon$ to be a diagonal matrix. For basis functions, we used 21 evenly spaced Gaussian radial functions and polynomial terms up to 3rd-order. The Gaussian widths were set empirically task-by-task. The probability of observing τ' is defined as

$$p(\tau' \mid \boldsymbol{\omega}) = \prod_t \mathcal{N}(\boldsymbol{\chi}_t \mid \boldsymbol{\Phi}_t \boldsymbol{\omega}, \boldsymbol{\Sigma}_\epsilon) \tag{4.4}$$

Now, suppose we have N demonstrations and use subscript $n \in \{1 \dots N\}$ to identify each demonstration. For the nth demonstration, $\boldsymbol{\chi}_{nt}$ represents the combined trajectory state from all reference frames at time t, and $\boldsymbol{\omega}_n$ is the compact representation of the combined trajectory. Assume $\boldsymbol{\omega}_n$ is sampled from a Gaussian distribution $\mathcal{N}(\boldsymbol{\omega}_n \mid \boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega)$, the likelihood of observing N combined trajectories is given by

$$p(\boldsymbol{X} \mid \boldsymbol{\theta}_\omega) = \prod_{n=1}^{N} \int \mathcal{N}(\boldsymbol{\omega}_n \mid \boldsymbol{\theta}_\omega) \prod_{t=1}^{T_n} \mathcal{N}(\boldsymbol{\chi}_{nt} \mid \boldsymbol{\Phi}_t \boldsymbol{\omega}_n, \boldsymbol{\Sigma}_\epsilon) d\boldsymbol{\omega}_n, \tag{4.5}$$

where $\boldsymbol{\theta}_\omega = \{\boldsymbol{\mu}_\omega, \boldsymbol{\Sigma}_\omega\}$, $\boldsymbol{X}$ is a set of N combined trajectories $\boldsymbol{\chi}_n$, and T_n denotes the trajectory length of the nth demonstration. The parameters $\boldsymbol{\mu}_\omega$, $\boldsymbol{\Sigma}_\omega$, and $\boldsymbol{\Sigma}_\epsilon$ can be learned through the Maximum A-Posteriori estimates(MAP) algorithm[40].

Given the learned parameters, the combined trajectory at any time point t can be evaluated by

$$p(\boldsymbol{\chi}_t \mid \boldsymbol{\theta}_\omega) = \int \mathcal{N}(\boldsymbol{\omega} \mid \boldsymbol{\theta}_\omega)\mathcal{N}(\boldsymbol{\chi}_t \mid \boldsymbol{\Phi}_t\boldsymbol{\omega}, \boldsymbol{\Sigma}_\epsilon)d\boldsymbol{\omega} \tag{4.6}$$

$$= \mathcal{N}(\boldsymbol{\chi}_t \mid \boldsymbol{\Phi}_t\boldsymbol{\mu}_\omega, \boldsymbol{\Phi}_t\boldsymbol{\Sigma}_\omega\boldsymbol{\Phi}_t^\top + \boldsymbol{\Sigma}_\epsilon). \tag{4.7}$$

The distribution of $\boldsymbol{\chi}_t$ is a Gaussian distribution with mean $\boldsymbol{\mu}_{\chi t} = \boldsymbol{\Phi}_t^\top\boldsymbol{\mu}_\omega$ and covariance $\boldsymbol{\Sigma}_{\chi t} = \boldsymbol{\Phi}_t^\top\boldsymbol{\Sigma}_\omega\boldsymbol{\Phi}_t + \boldsymbol{\Sigma}_\epsilon$. $\boldsymbol{\mu}_{\chi t}$. $\boldsymbol{\Sigma}_{\chi t}$ can be decomposed by corresponding reference frames $\boldsymbol{\mu}_{\chi t} = \begin{bmatrix}\boldsymbol{\mu}_{xt}^{(1)}, & \dots & ,\boldsymbol{\mu}_{xt}^{(R)}\end{bmatrix}$ and $\boldsymbol{\Sigma}_{\chi t} = \begin{bmatrix}\boldsymbol{\Sigma}_{xt}^{(1)} & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \boldsymbol{\Sigma}_{xt}^{(R)}\end{bmatrix}$. At time t, we obtain a Gaussian distribution with mean $\boldsymbol{\mu}_{xt}^{(j)}$, and covariance matrix $\boldsymbol{\Sigma}_{xt}^{(j)}$ for the trajectory in the jth reference frame . We then predict the trajectory in the global reference frame. This involves converting the local models to the global reference frame and multiplicatively combining the resulting Gaussian models. Therefore, given new task parameters $A^{(j)}$ and $b^{(j)}$ for the jth reference frame, we first convert $\boldsymbol{\mu}_{xt}^{(j)}$ and $\boldsymbol{\Sigma}_{xt}^{(j)}$ to the global reference frame as $\boldsymbol{\mu}_{xt,j}$ and $\boldsymbol{\Sigma}_{xt,j}$ respectively, where the subscript j indicates it is converted from the jth reference frame. The predicted covariance matrix in the global reference frame can be computed with

$$\boldsymbol{\Sigma}_{xt} = (\sum_{j=1}^{R} \boldsymbol{\Sigma}_{xt,j}^{-1})^{-1}, \tag{4.8}$$

and the predicted mean in the global reference frame can be computed with

$$\boldsymbol{\mu}_{xt} = \boldsymbol{\Sigma}_{xt} \sum_{j=1}^{R} \boldsymbol{\Sigma}_{xt,j}^{-1}\boldsymbol{\mu}_{xt,j} \tag{4.9}$$

Via-point modulation

As previously discussed, ProMPs achieves adaptation by implicitly conditioning on a via-point at a specific time t. While this approach may not be natural for adapting to new object poses, there are situations where implicit conditioning can be beneficial. For instance, when a new obstacle is introduced into the environment after the model has been trained, conditioning the model on a position far from the obstacle allows it to continue

performing the task without requiring the learning of a new model. Similar to [95], the proposed method achieves via-point modulation by conditioning. Given a desired point $\boldsymbol{x}_t^*$ in the global reference frame, we can compute the combined desired point in all reference frames $\boldsymbol{\chi}_t^*$. Then, for a specific accuracy matrix of the desired observation $\boldsymbol{\Sigma}_\chi^*$, the conditional distribution $p(\boldsymbol{\omega} \mid \boldsymbol{\chi}_t^*)$ is Gaussian with mean and variance [95]

$$\boldsymbol{\mu}_\omega^{[new]} = \boldsymbol{\mu}_\omega + \boldsymbol{\Sigma}_\omega \boldsymbol{\Phi}_t (\boldsymbol{\Sigma}_\chi^* + \boldsymbol{\Phi}_t^\top \boldsymbol{\Sigma}_\omega \boldsymbol{\Phi}_t)^{-1} (\boldsymbol{\chi}_t^* - \boldsymbol{\Phi}_t^\top \boldsymbol{\mu}_\omega) \tag{4.10}$$

$$\boldsymbol{\Sigma}_\omega^{[new]} = \boldsymbol{\Sigma}_\omega - \boldsymbol{\Sigma}_\omega \boldsymbol{\Phi}_t (\boldsymbol{\Sigma}_\chi^* + \boldsymbol{\Phi}_t^\top \boldsymbol{\Sigma}_\omega \boldsymbol{\Phi}_t)^{-1} \boldsymbol{\Phi}_t^\top \boldsymbol{\Sigma}_\omega. \tag{4.11}$$

Figure 4.3 provides a visual example of conditioning a TP-ProMPs model to different via points. TP-ProMPs maintains local low-variance movement patterns while passing through the via points. This demonstrates the ability of TP-ProMPs to adapt and incorporate via points with minimal deviations from the learned movement pattern.

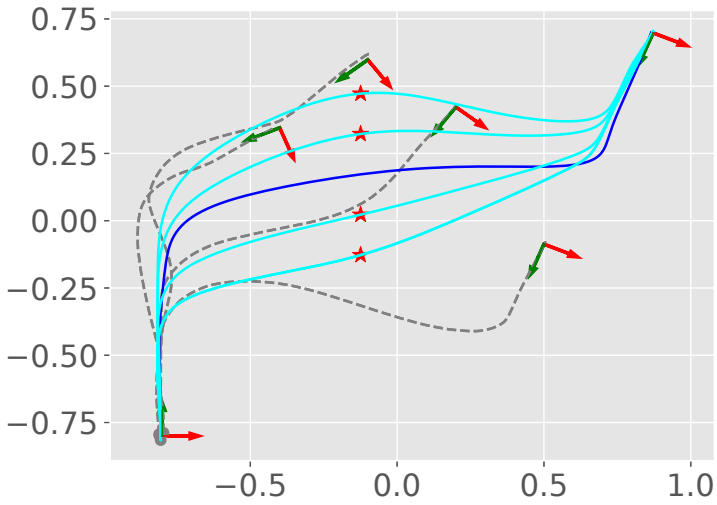


Figure 4.3: **Conditioning on different via-points.** The dashed gray trajectories are the training demonstrations. The blue trajectory is the prediction of the original model given new task parameters. The cyan trajectories are modulated trajectories conditioned on different via points (red stars).

4.2.5 Extensions of TP-ProMPs

Model $\boldsymbol{\omega}$ as a mixture of Gaussians

To accommodate tasks with multiple modes, we extended our method by representing the weight vector $\boldsymbol{\omega}_n$ as a mixture of Gaussians. In previous approaches, such as the one proposed in [31], a point estimate of $\boldsymbol{\omega}_n$ was first obtained using least squares, and then an Expectation-Maximization (EM) algorithm was used to learn a GMM in the weight space. However, this point estimate ignored the uncertainty of $\boldsymbol{\omega}_n$ and resulted in reduced robustness in tasks with high sensor noise. Additionally, the Maximum Likelihood Estimate (MLE) used in the M-step of the EM algorithm led to numerical issues when computing the covariance matrix $\boldsymbol{\Sigma}_\omega$.

To overcome these limitations, an alternative approach was presented in [40], which incorporated uncertainty estimation by iteratively updating $\boldsymbol{\omega}_n$ in the E-step based on the optimized parameters from the previous M-step. Furthermore, a prior distribution was introduced to achieve a Maximum A Posteriori (MAP) estimate and address the numerical issues. However, the method in [40] still modeled $\boldsymbol{\omega}$ as a Gaussian distribution, making it incompatible with trajectories exhibiting multiple modes.

In our extension, we have made modifications to the algorithm proposed in [40] by introducing a mixture of Gaussians model for $\boldsymbol{\omega}$. The extended algorithm consists of two steps. In the first step, we employed an EM algorithm to cluster the N demonstrations into C groups. During the E-step, each Gaussian component yields an estimated $\overline{\boldsymbol{\omega}}_{ni}$ as described in [40]. The subscript i denotes it is for the ith component. The overall estimated value $\overline{\boldsymbol{\omega}}_n$ was then computed as the weighted sum of $\overline{\boldsymbol{\omega}}_{ni}$ across all Gaussian components. In the M-step, we used a MAP estimate to update the parameters for each Gaussian component. We utilized a Normal-Inverse-Wishart distribution with parameters k_0, $\boldsymbol{m}_0$, v_0, and $\boldsymbol{S}_0$. In this work, as guided by the recommendations in [40], we set $k_0 = 0$, $\boldsymbol{m}_0 = \mathbf{0}$, $v_0 = \dim(\boldsymbol{\omega}) + 1$, and $\boldsymbol{S}_0 = (v_0 + KM + 1)\text{blockdiag}(\boldsymbol{\Sigma}^*_{\omega i})$, where $\boldsymbol{\Sigma}^*_{\omega i}$ is the MLE of the covariance matrix of the ith Gaussian component. In this step, since our goal was simply to cluster demonstrations based on execution modes, we forced the covariance matrix of each Gaussian component $\boldsymbol{\Sigma}_{\omega i}$ to be diagonal for better cluster performance.

In the second step, C ProMPs models were further trained on the corresponding data using the parameters $\boldsymbol{\mu}_\omega$, $\boldsymbol{\Sigma}_\omega$, and $\boldsymbol{\Sigma}_\epsilon$ obtained from the first step as initial parameters, following the training procedure described in [40].

Paired-object parameterized ProMPs

We extended the TP-ProMPs model to allow greater flexibility which we will now describe. For each source object $\boldsymbol{o} \in \boldsymbol{O}$, where $\boldsymbol{O} = \{\boldsymbol{o_i}\}_{i=1}^{R}$ and some target object $\boldsymbol{o'} \in \boldsymbol{O} \setminus \{\boldsymbol{o}\}$, we generated a reference frame $(\boldsymbol{o}, \boldsymbol{o'})$. The origin of the new coordinate system in the pairwise reference frame was the center of the source object $\boldsymbol{o}$. The axes were aligned as follows. The x-axis was in the direction of the line passing from the center of $\boldsymbol{o}$ to that of $\boldsymbol{o'}$. The new y-axis was in the direction of a unit vector $\boldsymbol{u'}$ orthogonal to the x-axis such that $\max_{\boldsymbol{u'}=\boldsymbol{u}} \boldsymbol{u} \cdot \boldsymbol{u}_{\boldsymbol{height}}^{\boldsymbol{world}}$, where $\boldsymbol{u}_{\boldsymbol{height}}^{\boldsymbol{world}}$ is the unit vector in the world coordinate parallel to the height axis. The z-axis was given by the right-hand rule.

This procedure produced $R_{pop} = R^2$ reference frames, consisting of R individual reference frames centered on the object and aligned with world axes, plus $R(R-1)$ pairwise reference frames. To counter the quadratic growth of complexity and eliminate useless reference frames, we introduced a selection process to pick out the most task-relevant frames for each object. The mean and variance of the demonstration trajectories in each candidate reference frame were calculated, and the candidate for a given object with the tightest clustering at any moment in the trajectory (i.e. the smallest variance over candidates and time) was selected. The reference frames selected by this method may align with a task constraint at some point of time during the task. The resulting model has the same number of reference frames and parameters as the TP-ProMPs model.

4.2.6 Evaluation

To evaluate the effectiveness of our proposed method for a given task, we randomly selected N demonstrations from our dataset as the training set, 1 demonstration as the validation trajectory, and 1 demonstration as the test trajectory. The validation and test trajectories were not in the training set. As the task objects were positioned differently in each demonstration, the validation and test demonstrations did not resemble the training set. To assess the accuracy of the prediction, the Euclidean distance was used as a similarity measure for position. As for orientation, the norm of the difference of the quaternions was utilized as the metric [52]. We report the mean of this metric over the trajectory as it is a simple and general reflection of the performance across the trajectory as a whole. Additionally, we report the similarity measure at key points such as the start, end, and middle. We repeated this process 100 times for all tasks and computed the average and standard deviation. We compared our proposed method with the original ProMPs model, the time-based TP-GMMs model, the dynamic-based TP-GMMs model, and the KMPs

model. When comparing with the KMPs and TP-GMMs models (time-based and dynamic-based), we trained several models with the number of components ranging from 2 to 10. The model with the best position prediction was chosen for comparison.

Our basic TP-ProMPs model models the 3D gripper position only, with $\boldsymbol{\omega}$ modeled as a Gaussian distribution and only object reference frames used. This basic model was used in the pick-and-place task. Depending on the specific task demands, we extended this basic TP-ProMPs model in several ways:

1. 7D Trajectories: Including 3D positions and 4D Quaternions (Pouring task).
2. Multi-Modal Trajectories: Modeling $\boldsymbol{\omega}$ as GMM to handle multi-modal trajectories effectively (Pouring task).
3. Paired-object parameterization: This was used for the shooting and sweeping tasks.

4.3 Results

4.3.1 Pick and place

During data collection, the table was divided into two areas, and demonstrations with both the teabag and cup on the right half were labeled as the training data pool, while those with at least one object on the left half were labeled as the test data pool. This separation ensured that a portion or the entirety of a trajectory from the test pool falls outside the distribution of the training pool. We then assigned local reference frames to the cup and the teabag to train a TP-ProMPs model.

The evaluation was conducted as described in Section 4.2.6. To adapt to new objects' positions, we conditioned the ProMPs model as described in [95]. Specifically, we conditioned the starting point of the trajectory to be 20 mm above the teabag and the endpoint of the trajectory to be 20 mm above the cup. All 5 models used in the experiment were trained on 6 demonstrations. However, the ProMPs model was trained only on trajectories in the global reference frame. On the other hand, TP-ProMPs, TP-GMMs, and KMPs models were trained on trajectories in the cup and teabag's reference frames.

Figure 4.4a demonstrates the performance comparison between the 5 models for one specific test case. The figure shows that the TP-ProMPs, KMPs, and TP-GMMs models outperform the ProMPs model. The trajectories predicted by TP-ProMPs, KMPs, and TP-GMMs models are closer to the ground truth in this extrapolation test, particularly

at the beginning and end of the trajectory. This observation is significant in this task, as accurately predicting the start and end of the trajectory is crucial to ensure the robot successfully grasps the teabag and places it into the cup. The average performance over 100 tests is reported in Table 4.1. As seen in Table 4.1, TP-ProMPs outperforms other models in general, except at the start of the trajectory where TP-ProMPs shows slightly worse results compared to the KMPs and the TP-GMMs models.

	Start (mm)	End (mm)	Average (mm)
ProMPs	107.78 ± 7.99	137.84 ± 18.10	250.68 ± 137.13
TP-GMMs (time-based)	28.22 ± 14.90	74.11 ± 48.06	58.76 ± 21.43
TP-GMMs (dynamic)	27.54 ± 15.56	129.42 ± 100.18	83.45 ± 43.01
KMPs	$\mathbf{25.80 \pm 11.39}$	73.99 ± 36.44	57.48 ± 21.62
TP-ProMPs	32.62 ± 8.54	$\mathbf{48.60 \pm 20.68}$	$\mathbf{48.51 \pm 12.21}$

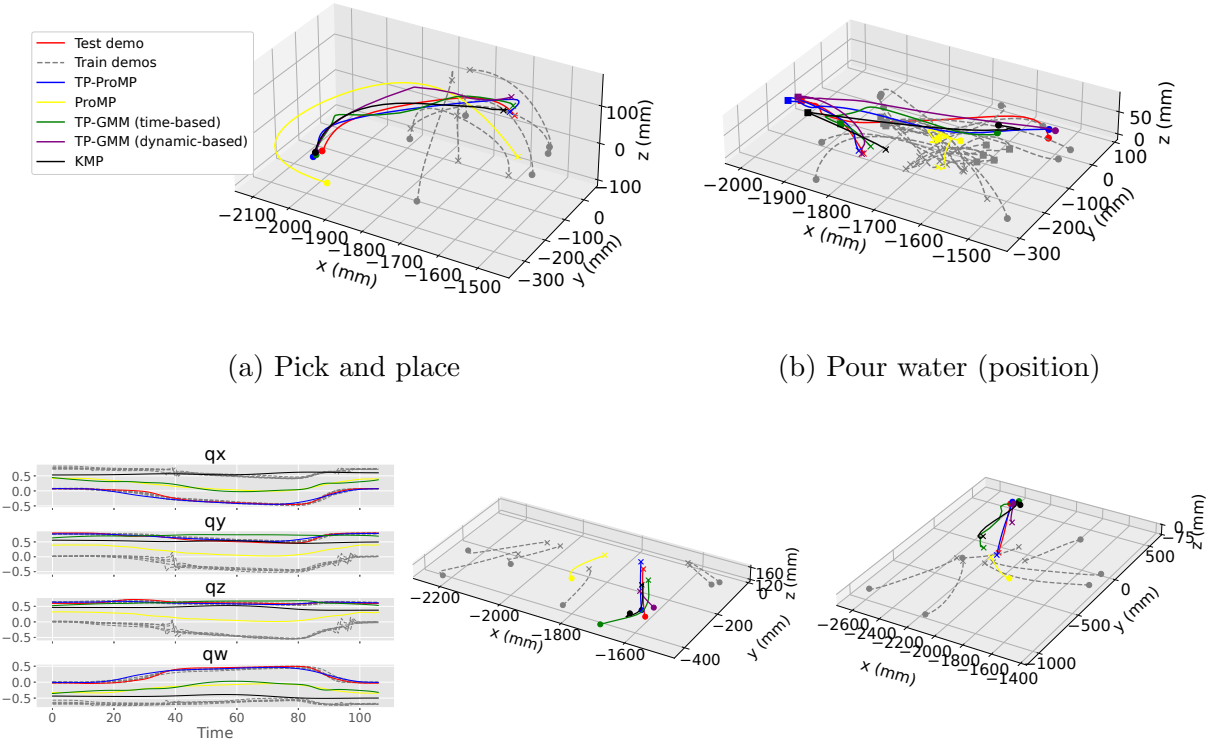
Table 4.1: **Performance comparison for the pick-and-place task with 6 training demonstrations.** Mean +/- standard deviation distance from ground truth for ProMPs, TP-GMMs, KMPs, and TP-ProMPs for the pick-and-place task.

ProMP's performance suffers due to two primary reasons. Firstly, it struggles when the conditioned point lies beyond the training distribution. Additionally, it heavily relies on heuristics, such as the gripper's assumed 20 mm elevation above objects at the trajectory's start and end points. These heuristic dependencies hinder ProMPs's adaptability and limit its application, particularly when precise timing at via points is uncertain, as seen in the water pouring task.

4.3.2 Water pouring

Demonstrations for this task used millet for safety, with executions in both left and right hands, resulting in two modes. Compared to pick-and-place, the pouring task is more challenging due to: 1) Gripper orientation changes significantly, requiring a 7D trajectory representation. 2) Involvement of two pouring modes, necessitating multi-modal trajectory modeling for accurate capture.

For position prediction, we compared the performance of TP-ProMPs with ProMPs, TP-GMMs (time-based), and TP-GMMs (dynamic-based), and KMPs models. All models were trained on a total of 10 demonstrations with equal numbers of left- and right-handed executions. The ProMPs model was not adapted, meaning it only captured the mean



(a) Pick and place

(b) Pour water (position)

(c) Pour water (orientation)

(d) Shoot Puck

(e) Sweep

Figure 4.4: **Trajectory predictions of the 5 models for 4 different tasks.** The grey trajectories are training demonstrations. The red trajectory represents the ground truth, while the blue, yellow, green, purple, and black trajectories represent the predictions by the TP-ProMPs model, the ProMPs model, the time-based TP-GMMs model, the dynamic-based TP-GMMs model, and the KMPs model, respectively. The circles denote the start of the trajectories and the crosses denote the end of the trajectories. (a) Position predictions for the pick-and-place task. (b) Position predictions for the pouring water task. A square symbol is used to indicate approximately when the pouring action occurs. (c) Orientation predictions for the pouring water task. (d) Position predictions for the shooting puck task. (e) Position predictions for the sweeping task.

behavior of the training demonstrations. As discussed in Section 4.3.1, ProMPs suffers from the limitation that adapting to new object positions is achieved through implicit conditioning on position and time. In this particular task, this implies conditioning on

unknown initial positions for grasping the pitcher at time $t = 0$ and an unknown via-point for pouring into the cup at an unknown time t. Conditioning the model on approximated position and time values is not only complex but also leads to poor performance. The TP-ProMPs model was trained following the procedure outlined in Section 4.2.5 with $C = 2$. For more generality, the Bayesian information criterion can be used for model selection. Both TP-ProMPs and dynamic-based TP-GMMs compared predictions from different modes to the ground truth, selecting the prediction closest to the ground truth for evaluation. Figure 4.4b illustrates the performance comparison of the 5 models in position prediction for the water-pouring task.

For orientation prediction, we conducted a comparison between TP-ProMPs, ProMPs, time-based TP-GMMs, and KMPs models. In [123], the quaternions predicted by a time-based TP-GMMs model were normalized to ensure they have unit norm. We followed the same procedure for all models with the exception of KMPs. This step was unnecessary for KMPs since it inherently respects the geometry of quaternions. Dynamic-based TP-GMMs was not included in the comparison because the literature does not describe its application to orientation, and the creators' public code does not support orientation. Figure 4.4c provides a comparison of the performance of the 4 models in orientation prediction for the water-pouring task.

The average performance over 100 tests is summarized in Table 4.2. TP-ProMPs outperformed the other models in both position and orientation modeling, highlighting its strength in handling trajectories with multiple modes. In contrast to [115], which models position and orientation separately, our approach simultaneously captures both, eliminating the need to match separate position and orientation models. The robustness of the clustering used in TP-ProMPs is a factor in its superior performance. As discussed in 4.2.5, using MAP estimate and estimating the uncertainty of $\boldsymbol{\omega}$ enables the proposed method to achieve a better estimate of $\boldsymbol{\omega}$. In this study, the proposed clustering algorithm achieved an accuracy of 93.5%, outperforming the algorithm presented [31], which achieved an accuracy of only 63.3%.

4.3.3 Shooting puck

Aiming is a more challenging task that has a low tolerance for variance. In order to enable our model to generalize to such tasks, we extended our method by assigning paired-object reference frames, as described in Section 4.2.5. The extended method was tested in a shooting puck task, where the demonstrator shoots a puck into a net on the table. In each demonstration, the position of the net remained fixed, while the position of the puck was

	Position				Orientation
	Start (mm)	Mid (mm)	End (mm)	Average (mm)	Average
ProMPs	249.6 ± 129.3	190.2 ± 88.96	140.6 ± 86.86	177.3 ± 74.83	1.028 ± 0.01667
TP-GMMs(time-based)	113.2 ± 53.41	132.8 ± 94.44	176.3 ± 129.8	142.4 ± 95.78	0.6518 ± 0.3171
TP-GMMs(dynamic)	78.78 ± 74.44	90.16 ± 68.36	121.9 ± 90.20	101.8 ± 69.68	N/A
KMPs	92.78 ± 31.78	105.9 ± 79.48	184.5 ± 132.9	139.3 ± 92.54	1.025 ± 0.01694
TP-ProMPs (GMM)	$\mathbf{18.83 \pm 8.181}$	$\mathbf{51.00 \pm 34.59}$	$\mathbf{83.97 \pm 52.92}$	$\mathbf{69.71 \pm 40.13}$	$\mathbf{0.1129 \pm 0.1112}$

Table 4.2: **Performance comparison for the pouring water task with** 10 **training demonstrations.** Mean ± standard deviation distance from ground truth for ProMPs, TP-GMMs, KMPs, and TP-ProMPs (GMM).

randomly selected. The TP-ProMPs was trained using trajectories in selected reference frames as discussed in Section 4.2.5, while TP-GMMs and KMPs models were trained on trajectories in object reference frames. The ProMPs model, on the other hand, was trained using trajectories in the global reference frame.

Figure 4.4d presents a visual comparison between the 5 models in the shooting puck task. The models were trained on 6 demonstrations. As shown in the figure, the ProMPs model performed poorly due to a lack of task parametrization. TP-GMMs and KMPs models, which incorporate object parameterization, demonstrate improved performance at the start and the end of the trajectory. However, the shooting action requires the gripper path to be straight in the puck-to-net direction for a successful outcome. The predictions generated by KMPs and TP-GMMs exhibit significant curvature and deviate considerably from the ground truth. In contrast, TP-ProMPs' predictions demonstrate great adaptation and exhibit minimal deviation from the ground truth, showing its superior performance in modeling the shooting action. Table 4.3 demonstrates the superior performance of TP-ProMPs in general, except at the end of the trajectory where TP-ProMPs shows slightly worse results compared to the KMPs.

4.3.4 Sweeping

To further evaluate the effectiveness of the TP-ProMPs method with paired-object reference frames, we conducted additional tests on another aiming task, specifically sweeping. In this task, a demonstrator cleaned rubbish into a dustpan. Due to physical constraints of the experimental setup, all demonstrations involved sweeping from the right to the left. To better mimic natural sweeping actions that can be executed in any direction, the demonstrations were augmented by rotating the trajectory and teabags' positions about

	Start (mm)	End (mm)	Average (mm)
ProMPs	269.8 ± 146.7	175.3 ± 92.73	237.0 ± 123.9
TP-GMMs (time-based)	76.39 ± 46.93	55.24 ± 29.56	67.96 ± 24.32
TP-GMMs (dynamic-based)	91.40 ± 79.73	79.72 ± 82.92	97.00 ± 86.21
KMPs	71.99 ± 42.16	$\mathbf{51.76 \pm 29.42}$	65.87 ± 24.80
TP-ProMPs	$\mathbf{45.28 \pm 25.08}$	52.92 ± 27.65	$\mathbf{57.57 \pm 24.38}$

Table 4.3: **Performance comparison for the shooting task with 6 training demonstrations.** Mean +/- SD distance from ground truth for ProMPs, TP-GMMs, KMPs, and TP-ProMPs for the shooting puck task.

the dustpan's vertical axis (z-axis) by a random angle. This augmentation produced a more diverse set of demonstrations.

The test results for the sweeping task are summarized in Table 4.4. Notably, TP-ProMPs with paired-object reference frames exhibited significantly improved performance over the trajectory and at the start of the trajectory. Poorer performance at the trajectory's end (compared with the start) is attributed to the size difference between the dustpan and teabag. The teabags, being smaller, may land in various parts of the dustpan. While this variability causes greater distances from the ground truth at the trajectory's end, these distances are relatively small compared to the dustpan's width.

Figure 4.4e shows a visual comparison between the 5 models. TP-ProMPs with paired-object reference frames provides a prediction that deviates the least from the ground truth.

	Start (mm)	End (mm)	Average (mm)
ProMPs	638.1 ± 324.2	240.4 ± 111.9	474.9 ± 216.3
TP-GMMs(time-based)	107.0 ± 58.87	189.2 ± 107.3	152.6 ± 64.19
TP-GMMs(dynamic-based)	131.3 ± 122.8	289.3 ± 235.3	211.5 ± 125.5
KMPs	97.62 ± 45.51	180.4 ± 95.67	139.3 ± 59.69
TP-ProMPs	$\mathbf{68.98 \pm 61.91}$	$\mathbf{110.51 \pm 85.00}$	$\mathbf{106.45 \pm 49.03}$

Table 4.4: **Performance comparison for the sweeping task with 6 training demonstrations.** Mean +/- SD distance from ground truth for ProMPs, TP-GMMs, KMPs, and TP-ProMPs for the sweeping task.

4.4 Conclusion

Task-parameterized probabilistic movement primitives (TP-ProMPs) offer a promising approach that combines the advantages of both ProMPs and TP-GMMs, making it well-suited for generalizing to different tasks. Its strengths include excellent extrapolation performance, simultaneous modeling of position and orientation, support for trajectories with multiple modes, and improved performance in aiming tasks. In future work we plan to study the method's efficiency (although it ran in $7 - 30$ms on a desktop computer), to compare this method to neural network (NN)-based methods such as those proposed in [121, 140, 104, 127, 96], and to integrate TP-ProMPs into multi-step tasks.

Chapter 5

Task-parameterized Transformer

Previous collaborative work on TP-TF [20] demonstrated that conditioning a transformer on task parameters describing the scene analogous to TP-ProMPs yields performance that matches or exceeds other PbD techniques on real-world manipulation tasks such as pick-and-place and pouring. Remarkably, TP-TF attained this level of performance with only 5–30 demonstrations, suggesting that the large data demands typical of deep-learning trajectory models stem primarily from perception, not control. To further evaluate the model's capacity for representing long-horizon, multi-stage manipulation and to move toward unifying the two-level framework into a single model, this study applies TP-TF to the bolt–nut assembly task on a real robotic platform.

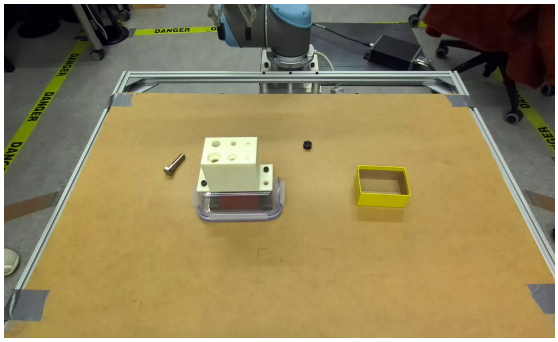
5.1 Methods

5.1.1 Physical task

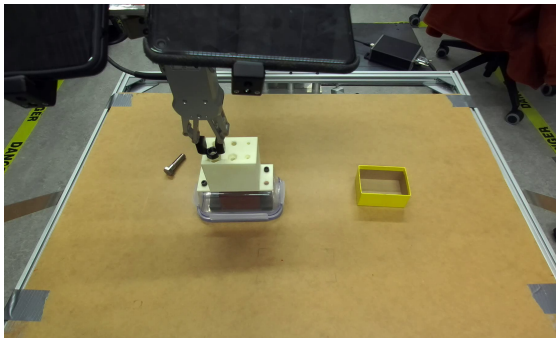
The bolt–and-nut assembly task comprises three sequential subtasks (Figure 5.1):

1. **Insert nut into jig. (Subtask 1)** The nut must be aligned with a jig recess whose diameter exceeds the nut's by only 1–3 mm; high positional accuracy is therefore required. Symmetry of the nut from a top-down view complicates pose estimation.
2. **Insert bolt into jig. (Subtask 2)** The bolt is grasped horizontally and re-oriented to a vertical pose before assembly, testing the model's ability to represent rotations.

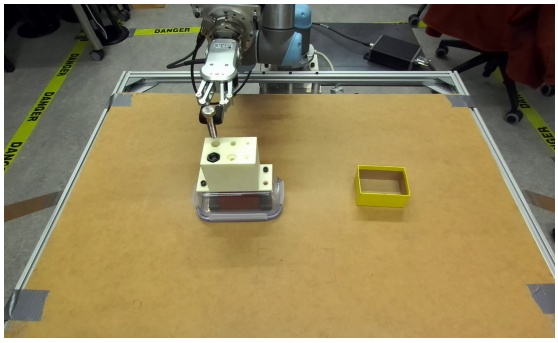
3. **Screw bolt into nut and place assembled pair in box. (Subtask 3)** A continuous screwing motion with tight tolerance is needed to thread the bolt, again demanding precise control of position and orientation. The completed assembly is transferred to a box, concluding the episode.



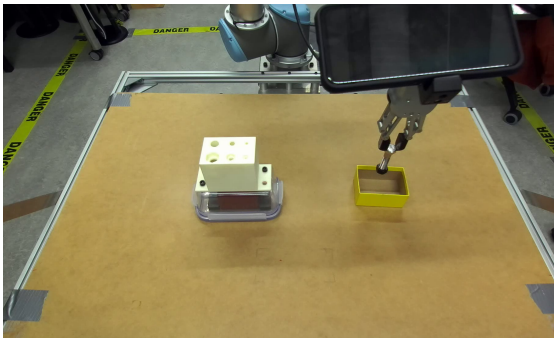
(a) Scene before action.



(b) Insert nut to jig.



(c) Insert bolt to jig.



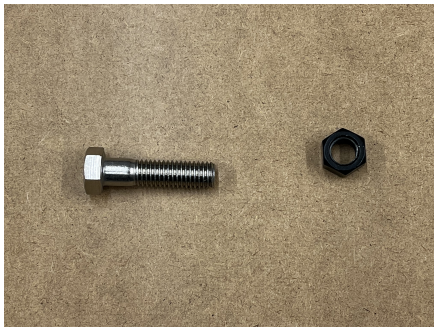
(d) Screw bolt into nut and place assembled pair in box

Figure 5.1: Subtasks of the bolt-and-nut assembly task.

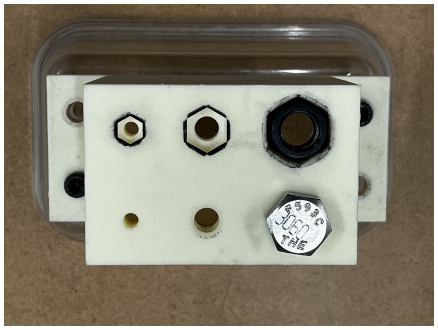
Each stage stresses different aspects of the task: positional precision, rotational adaptation, and smooth control of coupled translational–rotational motion. Figure 5.2 provides a close-up view of the assembly objects.



(a) Top view of the jig



(b) Bolt and nut separated



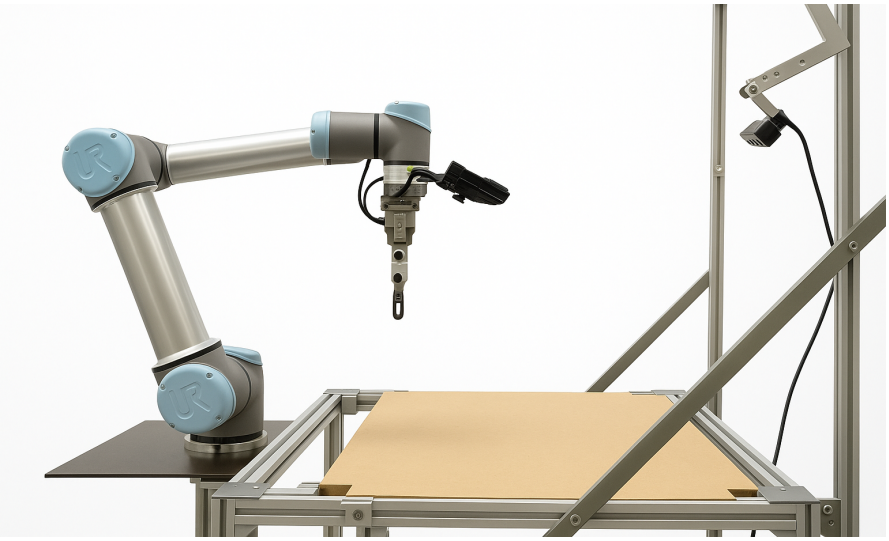
(c) Top view of the jig with bolt and nut inserted

Figure 5.2: Assembly objects close-up: (a) shows empty jig, (b) shows bolt and nut separated, and (c) shows the jig with bolt and nut inserted.

5.1.2 Experiment setup

Figure 5.3 depicts the experimental setup. Demonstrations and reproductions are executed on a UR5 robot arm equipped with an OnRobot RG2 parallel-jaw gripper. Perception is provided by two stereo systems:

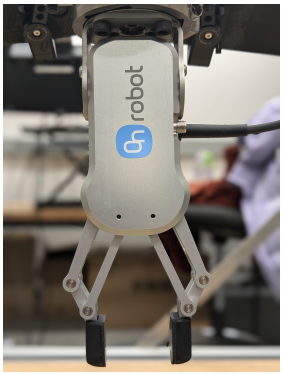
- A ZED2 stereo camera mounted above the workspace at about 45° angle offers a global view for coarse pose estimation.
- A wrist-mounted stereo pair consisting of two Google Pixel 4a smartphones supplies high-resolution, close-range RGB data during manipulation.



(a) Entire setup



(b) Wrist cameras



(c) Gripper close-up

Figure 5.3: **Robotic Setup for the Assembly Task. (a)** Side view showing the full setup, including a UR5 robot arm, a flat workspace, an OnRobot RG2 gripper, a wrist-mounted stereo camera pair, and a ZED2 stereo camera. **(b)** Wrist-mounted stereo camera pair, consisting of two Pixel 4a smartphones. **(c)** OnRobot RG2 gripper.

5.1.3 Data collection

Each demonstration is performed via keyboard-based teleoperation. A set of pre-defined *micro-programs*—move forward, lift, rotate the wrist, close/open the gripper, and so

forth—are bound to specific keys; pressing a key executes the corresponding motion primitive until the key is placed.

Prior to starting a demonstration, a short video clip is recorded with the overhead ZED2 camera; this clip is later processed to obtain coarse estimates of object poses. At keypoints of the task (e.g. immediately before grasping or placing), additional still images are captured from the wrist-mounted cameras, again triggered from the keyboard. Robot data are logged at 125 Hz, including the Cartesian pose of the end-effector and the binary state of the gripper.

5.1.4 Data preprocessing

Each demonstration is manually partitioned into the three subtasks introduced in Section 5.1.1. While various automatic segmentation methods exist (e.g., [84, 77, 109, 65]), they are typically tailored for shorter segments aimed at skill library construction or for datasets with noisy, repetitive behaviors. For instance, [77] integrates segmentation with trajectory learning by representing each segment as a ProMPs and modeling the overall skill library as a Gaussian mixture model. Both segmentation and trajectory refinement are performed iteratively through likelihood maximization. [65] models demonstrations as sequences of locally linear dynamical systems, detecting transition points via changes in the local dynamic matrices. This method is specifically designed for noisy surgical data characterized by complex and loopy behaviors. In contrast, this work employs manual segmentation into three subtasks, reflecting the way humans intuitively teach complex tasks—by breaking them down into high-level, meaningful components. This human-oriented approach promotes clearer instruction, interpretability, and consistency across demonstrations. Notably, this yields comparatively long segments, thereby placing greater demands on the model's capacity to learn longer trajectories.

The recorded orientation trajectory, originally stored in axis–angle form, is converted to a unit-quaternion representation. After preprocessing, the state at time step t is represented by the eight-dimensional vector

$$\boldsymbol{Y}_t = [\boldsymbol{p}, \boldsymbol{q}, g]^\top$$

where $\boldsymbol{p} = (p_x, p_y, p_z)$ is the Cartesian position of the end-effector, $\boldsymbol{q} = (q_x, q_y, q_z, q_w)$ is the associated unit quaternion, and $g \in \{0, 1\}$ encodes the gripper state (open = 0, closed = 1).

For every subtask, the recorded trajectory is first temporally down-sampled to 1 Hz and subsequently aligned across demonstrations by dynamic time warping (Dynamic Time

Warping (DTW)) [88]. The overhead video and wrist-camera images are processed with DeepLabCut, and object poses are then recovered as detailed in Section 4.2.2.

5.1.5 Data Augmentation

Transformer architectures generally benefit from extensive training data [14, 149, 113, 27]. To compensate for the limited number of demonstrations available in this study, a task-parameterised augmentation procedure inspired by TP-ProMPs is applied.

Identifying moving objects. Given N demonstrations, the pose of every task-relevant object is analyzed. For each object, the standard deviation of its Cartesian position and of its orientation (expressed as Euler angles) is computed across demonstrations. An object is classified as *moving* if the positional deviation exceeds 10 mm or the orientational deviation exceeds 10° about any axis. Position and orientation are treated separately because some objects vary only in one of these modalities.

Trajectory statistics in object frames. Let $\tau = \{\boldsymbol{\chi}_t\}_{t=1}^{T}$ denote a trajectory recorded in the global (robot-base) frame, where $\boldsymbol{\chi}_t = [p_x, p_y, p_z, q_x, q_y, q_z, q_w]^\top$. Note that the gripper state g is omitted here since it does not require augmentation. For each of the R objects whose poses constitute the task parameters, the trajectory is transformed into the object's local frame, yielding $\{\tau^{(j)}\}_{j=1}^{R}$.

A fully TP-ProMPs approach would involve training a TP-ProMPs model and then marginalizing its basis weights $\boldsymbol{\omega}$ to recover the mean $\boldsymbol{\mu}_{xt}^{(j)}$ and covariance $\boldsymbol{\Sigma}_{xt}^{(j)}$ (see Section 4.2.4 for details). However, in this assembly task, five reference frames (four local object frames plus the global base frame) are involved, which makes the associated covariance matrix $\boldsymbol{\Sigma}_w$ difficult to estimate reliably. This limitation of TP-ProMPs becomes more pronounced as the number of objects grows.

Instead, the simpler choice here is to compute a time-indexed mean $\boldsymbol{\mu}_t^{(j)}$ and covariance $\boldsymbol{\Sigma}_t^{(j)}$ directly from the ensemble of N demonstrations, in each object's local frame. Notice this does not model temporal correlations as in TP-ProMPs and could lead to jerky trajectories if the demonstrations are not perfectly temporally aligned.

Sampling new task parameters. For every moving object, either position, orientation, or both are perturbed:

- **Position.** New x and y coordinates are sampled uniformly within the permissible workspace; the z-coordinate is left unchanged because all objects rest on a common table surface.
- **Orientation.** A rotation angle uniformly drawn from $[-30°, 30°]$ is applied about the global z-axis.

Generating trajectories via Product-of-Gaussians Given the perturbed object poses, the frame-specific statistics $\boldsymbol{\mu}_t^{(j)}$ and $\boldsymbol{\Sigma}_t^{(j)}$ are transformed back to the global frame, producing $\boldsymbol{\mu}_{t,j}$ and $\boldsymbol{\Sigma}_{t,j}$. The fused mean and covariance in the global reference frame at time t are obtained through via the product-of-Gaussians formulation:

$$\boldsymbol{\Sigma}_t = \Big(\sum_{j=1}^{R} \boldsymbol{\Sigma}_{t,j}^{-1}\Big)^{-1}, \tag{5.1}$$

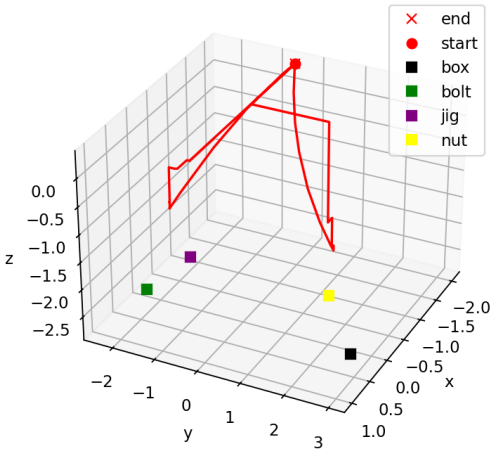
$$\boldsymbol{\mu}_t = \boldsymbol{\Sigma}_t \sum_{j=1}^{R} \boldsymbol{\Sigma}_{t,j}^{-1}\, \boldsymbol{\mu}_{t,j}. \tag{5.2}$$

Figure 5.4 (a) illustrates a demonstrated trajectory in the *insert-nut-into-jig* subtask, whereas Figure 5.4 (b) shows a trajectory generated from that demonstration by the procedure above. Owing to the independent treatment of each time step, some noise arises—particularly before grasping.

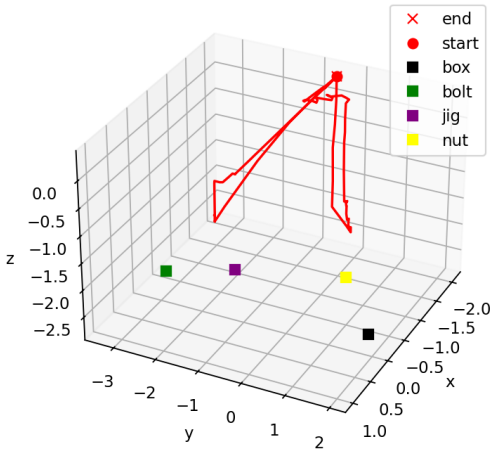
Refining trajectories through winner-take-all. Equation (5.2) expresses the global mean $\boldsymbol{\mu}_t$ as a covariance-weighted sum of the local means, analogous to an ensemble of experts in which frames with lower covariance exert greater influence. Since the covariance matrix is not properly modeled and yielding noisy global trajectory, a more robust winner-take-all (WTA) strategy is adopted. First, the covariance matrices for each local frame, $\boldsymbol{\Sigma}_{t,j}$, are approximated by diagonal forms $\hat{\boldsymbol{\Sigma}}_{t,j}$. The local frame with the smallest value $\min_j \max(\hat{\boldsymbol{\Sigma}}_{t,j})$ fully determines the global trajectory at time t, ignoring the other frames. Position and orientation are treated independently to account for their distinct scales. Figure 5.4 (c) shows the resulting WTA trajectory, which is smoother before the grasp but may exhibit abrupt transitions when dominance switches from one frame to another.

Combined method. A hybrid approach further mitigates sudden changes. Let $\mathrm{tr}(\boldsymbol{\Sigma}_t)$ denote the trace of $\boldsymbol{\Sigma}_t$ in (5.1). Define the threshold $\Lambda = \lambda \max_{t=1..T} \mathrm{tr}(\boldsymbol{\Sigma}_t)$ for some

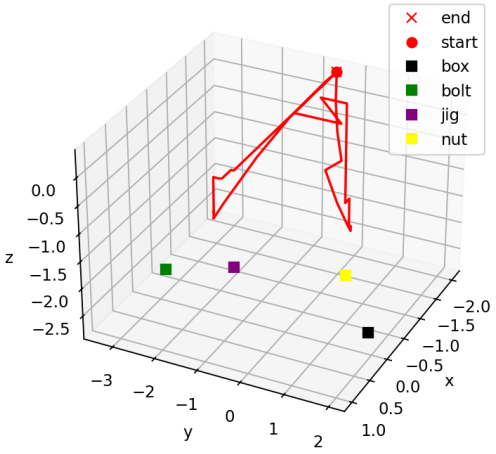
$0 < \lambda < 1$. At time steps t where $\mathrm{tr}(\boldsymbol{\Sigma}_t) < \Lambda$, the WTA strategy applies, while for higher-variance steps, the standard product-of-Gaussians approach is used. In this work, a value of $\lambda = 0.5$ is used. Larger values of λ make the fusion behavior more similar to WTA, while smaller values cause it to behave more like the product-of-Gaussians approach. A small Gaussian filter is then convolved with the resulting trajectory to enforce additional smoothness. Figure 5.4 (d) illustrates the final combination, which reduces noise while avoiding discontinuities introduced by a purely winner-take-all approach.



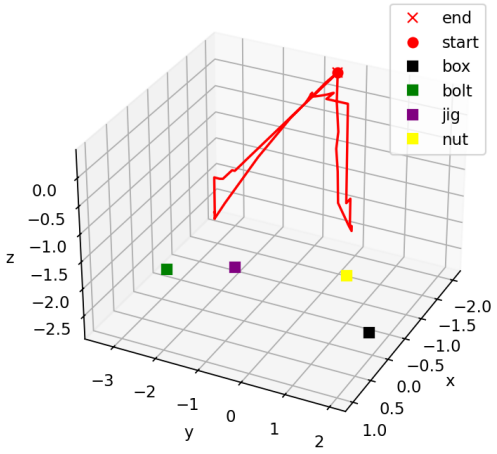
(a) Original trajectory



(b) Generated trajectory using Product-of-Gaussians



(c) Generated trajectory using WTA



(d) Generated trajectory using combined methods.

Figure 5.4: Generated trajectories with different methods.

5.1.6 Transformer

Architecture

Figure 5.5 shows the architecture of the transformer network, which adopts the encoder–decoder structure of the original Transformer [131]. The encoder encodes a set of object poses, infers the current subtask, and produces a latent representation that conditions the decoder. The decoder then predicts the eight–dimensional gripper trajectory associated with that subtask.

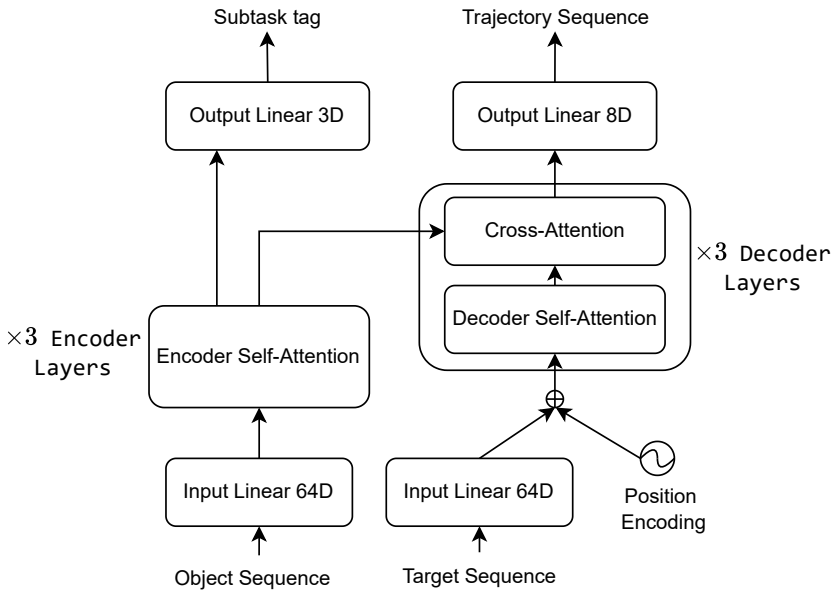


Figure 5.5: Architecture of the transformer network.

Object sequence. Consider the k-th object ($k \in \{1, \dots, n\}$) with Cartesian position $\boldsymbol{p} \in \mathbb{R}^3$ and unit-quaternion orientation $\boldsymbol{q} \in \mathbb{R}^4$. This object pose information is obtained by analyzing stereo video or images using DeepLabCut (DLC) (see Section 4.2.2 for details). A one-hot vector $\boldsymbol{o} \in \{0,1\}^n$ identifies the object type, where $o_k = 1$ and $\sum_{i=1}^{n} o_i = 1$. Concatenating pose and type yields the object token

$$\boldsymbol{e} = \left[\boldsymbol{p},\, \boldsymbol{q},\, \boldsymbol{o}\right] \in \mathbb{R}^{d_o+n},$$

with $d_o = 7$ for the pose components. Stacking the n tokens forms an object sequence of size $n \times (d_o + n)$. It is worth noting that this setup does not inherently require the presence

of only one object per type. However, in this work, all experiments are conducted with exactly one object of each type.

Target sequence. The target sequence consists of T consecutive gripper configurations. Because every element represents the gripper, the associated type vector is $\boldsymbol{o} = [0, \ldots, 0, 1]^\top$ (the final entry denotes the gripper class), while the position and orientation vectors are set to zero , serving as placeholders. Hence each token has dimension $d_o + n$, and the full target sequence has shape $T \times (d_o + n)$.

Dimensions in the assembly task. The workspace contains four physical objects—*box*, *bolt*, *jig*, and *nut*—together with the end-effector, so $n = 5$. Accordingly, the object sequence has dimension 5×12 ($d_o + n = 7 + 5$). The decoder must accommodate the longest subtask; a maximum horizon of $T = 200$ is therefore selected, producing a target sequence of shape 200×12. Both sequences pass through a shared linear projection before entering the encoder and decoder, respectively.

Positional encoding. Because the list of objects is unordered, no positional encoding is added to the encoder input. Temporal order, however, is essential for the decoder; hence sinusoidal position embeddings are summed with the decoder inputs prior to the multi-head attention layers, as indicated in Figure 5.5.

Loss function

The training objective comprises four terms—L_a, L_p, L_q, and L_g—which penalize errors in subtask classification, end-effector position, orientation, and gripper state, respectively.

Subtask loss. Whenever object poses are updated, a categorical cross-entropy loss L_a is computed between the predicted subtask probability $\hat{\boldsymbol{y}}_c$ and the one-hot ground truth $\boldsymbol{y}$:

$$L_a(\boldsymbol{y}, \hat{\boldsymbol{y}}_c) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c),$$

where C is the number of subtasks. The one-hot vector $\boldsymbol{y}$ identifies the ground-truth subtask ($y_c = 1$ for the true class and 0 otherwise), and $\hat{\boldsymbol{y}}_{\boldsymbol{c}}$ is the predicted probability obtained from a soft-max over the logits.

Trajectory losses. For position, orientation, and gripper state, a weighted mean-squared error is evaluated independently at every time step and then averaged over the trajectory:

$$L(Y, \hat{Y}) \;=\; \frac{1}{T} \sum_{t=1}^{T} w_t \, \|Y_t - \hat{Y}_t\|^2,$$

where $Y_t, \hat{Y}_t \in \mathbb{R}^D$ denote the ground truth and predicted vectors at time t; $D = 3$ for position, $D = 4$ for quaternions, and $D = 1$ for the binary gripper state. The scalar weight w_t emphasizes the keypoints of the trajectory: w_t initialized as 1 and is multiplied by 10^3 whenever the end-effector velocity falls below one quarter of its maximum, the gripper state changes, or a wrist-camera image is captured.

Total loss. The four components are combined linearly,

$$L_{total} \;=\; \alpha \, L_a + \beta \, L_p + \gamma \, L_q + \delta \, L_g,$$

with $\alpha = 100$, $\beta = 1$, $\gamma = 2$, and $\delta = 1000$, yielding initial loss terms of comparable magnitude across all channels.

Weighted Total Loss. The total loss is modulated by assigning higher weights to time steps identified as keypoints. A time step is defined as a keypoint if any of the following conditions hold:

1. the robot is moving at a relatively low speed,
2. the gripper state changes, or
3. wrist camera images are recorded.

Formally, a time step is considered slow if the instantaneous speed falls below 25% of the maximum observed speed. Keypoints are emphasized by assigning them a weight that is 100 times greater than that of non-keypoint time steps.

Training

Each trajectory is divided into segments, where each segment spans the interval between two successive updates of the object poses (e.g., between two consecutive wrist camera image captures). The loss is evaluated only on the corresponding segment whenever the object sequence is updated. In this work, there is one ZED2 video and three wrist images captured during the task, the object poses are updated four times, resulting in the entire trajectory being divided into four segments. The loss is then computed separately for each of these segments.

Training uses a batch size of 8. The learning rate starts at 1×10^{-5} and is halved whenever the validation loss fails to improve by at least 1% of the best value for 500 epochs (the patience period). Optimization stops when the learning rate falls below 1×10^{-9}.

5.1.7 Baselines

The proposed method is evaluated against two traditional PbD approaches—TP-GMMs[17] and TP-ProMPs[141]—as well as two neural network-based baselines: CNMP[121] and CNEP[142]. For the traditional PbD methods, optimal hyperparameters are selected via grid search on a validation set, separately for each subtask. Specifically, the number of Gaussian components is tuned for TP-GMMs to control model complexity. For TP-ProMPs, the model uses polynomial terms up to third order and 21 evenly spaced Gaussian radial basis functions. The parameter σ, which defines the width of the Gaussian basis functions, is optimized through grid search.

For CNMP and CNEP, the encoder and decoder hidden dimensions are set to 512, ensuring a comparable number of trainable parameters to the proposed transformer-based model. Batch sizes of 2 (as in the original implementation) and 8 (to match the transformer model) were tested. The model yielding better test performance was selected for comparison. All remaining hyperparameters are kept consistent with those used in the original implementations. The snapshot with the least validation error is selected for comparison.

Each method is trained on 15 demonstrations, with 4 additional demonstrations used for validation and 1 for testing. To account for variability, three trials with different training-validation-test splits are conducted, and mean and standard deviation metrics are reported.

Since the baseline methods do not natively support updated object poses, object pose information is extracted from a single image taken from the initial frame of the ZED2 video. This image is used either for estimating object poses (TP-GMMs, TP-ProMPs,

and TP-TF) or for extracting features (CNMP and CNEP). For all baseline methods, a separate model is trained for each subtask. In contrast, the proposed TP-TF is trained as a single model jointly across all three subtasks.

5.1.8 Metrics

Model comparison

To evaluate the discrepancy between predicted and ground-truth trajectories, we report the Average Displacement Error (ADE) for position and the Normalized Difference of Quaternions (NDQ) [52] for orientation. The ADE is computed as the mean of the Euclidean (L2) distances between corresponding positions along the predicted and reference trajectories. The NDQ measures orientation error as the mean norm of the difference between corresponding quaternions at each time step. Since all trajectories were temporally aligned using DTW to a fixed number of time steps, these comparisons were computed directly at corresponding indices.

Success rate

To evaluate task execution performance, the success rate is reported during autonomous reproduction of the task. A grasping attempt is considered successful if the object remains securely held in the gripper without being dropped prior to place. The placing phase is evaluated based on task-specific criteria: in Subtask 1, it is successful only if the nut is placed flat within the jig hole—tilted placements are considered failures; in Subtask 2, success requires the bolt to be fully inserted into the jig hole; and in Subtask 3, success is defined as placing the assembled bolt–nut pair entirely inside the box. A screwing phase is added to Subtask 3. The screwing phase is deemed successful if the bolt–nut assembly remains secure and does not fall prior to place.

If a subtask fails, human intervention is used to complete it, after which the system resumes autonomous execution of the remaining subtasks. If the grasping phase fails, the associated placing (and screwing, if applicable) phases are automatically marked as failures.

5.1.9 Stereo Image-Based Visual Servoing

Image-based visual servoing (IBVS) is employed during grasping and placing operations to refine the end-effector pose. The monocular formulation reviewed in Section 2.4.2 is extended here to a calibrated wrist-mounted *stereo* camera pair. Prior works have investigated stereo eye-in-hand IBVS in various contexts [86, 56]; however, they rely on specific assumptions regarding the relative configuration of the two cameras. For example, [56] assumes a parallel stereo setup, while [86] assumes the cameras have the same angle about the vertical (Y) axis. In contrast, the method proposed in this work makes no strict assumptions about the stereo geometry. By transforming the right camera's interaction matrix into the left camera frame using the extrinsic parameters obtained from the stereo calibration, image-space errors from both views are simultaneously minimized through a unified least-squares control law.

IBVS with two cameras. Let the left and right cameras be denoted by the superscripts l and r, respectively. For a set of 2-D feature coordinates $\mathbf{p}^{(\cdot)} \in \mathbb{R}^N$ with depths $Z^{(\cdot)}$, the classical IBVS kinematics for each camera reads

$$\dot{\mathbf{p}}^l = L^l(\mathbf{p}^l, Z^l)\,\mathbf{v}^l, \tag{5.3}$$

$$\dot{\mathbf{p}}^r = L^r(\mathbf{p}^r, Z^r)\,\mathbf{v}^r, \tag{5.4}$$

where $L^{(\cdot)} \in \mathbb{R}^{N\times 6}$ is the interaction (Jacobian) matrix and $\mathbf{v}^{(\cdot)} \in \mathbb{R}^6$ is the spatial twist of the corresponding camera frame. The image-plane velocity $\dot{\mathbf{p}}^{(\cdot)}$ is typically computed as the difference between the current and target feature positions in the image plane. This quantity represents the image-space error that the IBVS controller seeks to minimize by generating appropriate camera motions $\mathbf{v}^{(\cdot)}$ in the 3D workspace.

Relating the two twists. In this work, only the pose of the *left* camera with respect to the end-effector, T_l^E, is estimated; hence the control law must be expressed in that frame. Let the known transformation from the right camera to the left camera be denoted by

$$T_l^r \;=\; \begin{bmatrix} R & t \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathrm{SE}(3), \quad R \in \mathrm{SO}(3),\; t \in \mathbb{R}^3.$$

Its adjoint representation

$$\mathrm{Ad}_{T_l^r} \;=\; \begin{bmatrix} R & \mathbf{0} \\ [t]_\times & R \end{bmatrix} \in \mathbb{R}^{6\times 6}, \quad [t]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix},$$

maps twists between the two frames, i.e. $\mathbf{v}^r = \mathrm{Ad}_{T_l^r}\,\mathbf{v}^l$.

Equivalent right-camera interaction matrix. Substituting the above relation into the Eq. 5.4

$$\mathrm{Ad}_{T_l^r}\,\mathbf{v}^l \;=\; L^r(\mathbf{p}^r, Z^r)^+\,\dot{\mathbf{p}}^r.$$

Multiplying both sides by $L^r(\mathbf{p}^r, Z^r)$ yields

$$L^r(\mathbf{p}^r, Z^r)\,\mathrm{Ad}_{T_l^r}\,\mathbf{v}^l = \dot{\mathbf{p}}^r.$$

Define the *equivalent* right-camera interaction matrix, expressed in the left-camera frame, as

$$\hat{L}^r(\mathbf{p}^r, Z^r) \;=\; L^r(\mathbf{p}^r, Z^r)\;\mathrm{Ad}_{T_l^r}\,.$$

Then, the twist $\mathbf{v}^l$ corresponding to the right image can be computed by applying the pseudo-inverse of the equivalent interaction matrix:

$$\mathbf{v}^l \;=\; \hat{L}^r(\mathbf{p}^r, Z^r)^+\,\dot{\mathbf{p}}^r.$$

Stacked control law. Concatenating the constraints from both image planes over N feature points leads to a single least-squares problem:

$$\mathbf{v}^l \;=\; \begin{pmatrix} L^l(\mathbf{p}_1^l, Z_1^l) \\ \hat{L}^r(\mathbf{p}_1^r, Z_1^r) \\ \vdots \\ L^l(\mathbf{p}_N^l, Z_N^l) \\ \hat{L}^r(\mathbf{p}_N^r, Z_N^r) \end{pmatrix}^+ \begin{pmatrix} \dot{\mathbf{p}}_1^{\,l} \\ \dot{\mathbf{p}}_1^{\,r} \\ \vdots \\ \dot{\mathbf{p}}_N^{\,l} \\ \dot{\mathbf{p}}_N^{\,r} \end{pmatrix}. \tag{5.5}$$

The resulting twist $\mathbf{v}^l$ is finally converted to end-effector commands through the standard eye-in-hand kinematic chain.

This stereo formulation leverages all available visual information while maintaining a single control signal, thereby enhancing accuracy—particularly near contact points—without making assumptions about the relative geometry of the cameras.

Integrate with TP-Transformer To integrate visual servoing with the TP-TF model, the entire task is first demonstrated once. During this demonstration, stereo image pairs are captured from the wrist-mounted stereo camera immediately before each grasping and placing action. These images are then processed using DLC for feature points detection and stored as target references for subsequent visual servoing.

During execution, the TP-TF model predicts trajectories for position, orientation, gripper state, and subtask labels. The robot follows the predicted trajectory up to a detected gripper state change. At this point, the predicted subtask tag is used to select the corresponding target image pair for visual servoing, and the robot switches to visual control using the stereo visual servoing strategy described previously. Upon completion of the visual servoing phase, the gripper is actuated (opened or closed) according to the predicted gripper state trajectory, and the robot resumes control based on the TP-TF predicted motion trajectory. This cycle is repeated for each subtask until the entire assembly task is completed.

It is important to note that the computation of the image-plane velocity vector $\dot{\mathbf{p}}$ in Eq. 5.5 differs between the grasping and placing phases. For each subtask, the identity of the object to be grasped is inferred during demonstration using a simple heuristic: the object that moves in coordination with the gripper is identified as the target.

During visual servoing execution, the current and target wrist image pairs are compared. For grasping, $\dot{\mathbf{p}}$ is computed directly from the displacement of the feature points on the object to be grasped:

$$\dot{\mathbf{p}} = \mathbf{u}^c - \mathbf{u}^t,$$

where $\mathbf{u}^c$ and $\mathbf{u}^t$ denote the coordinates of a feature point in the current and target images, respectively.

In contrast, during the placing phase, a relative vector-based formulation is employed to compute $\dot{\mathbf{p}}$. Specifically, the image-plane velocity is calculated from the difference between vectors formed by feature points on the background and those on the grasped object

$$\delta\mathbf{p}^{(i,j)} = (\mathbf{u}_b^{c,(j)} - \mathbf{u}_g^{c,(i)}) - (\mathbf{u}_b^{t,(j)} - \mathbf{u}_g^{t,(i)}),$$

where $\mathbf{u}_g^{(\cdot)}$ and $\mathbf{u}_b^{(\cdot)}$ denote the image coordinates of feature points on the grasped object and background objects, respectively. This formulation yields one relative vector for each grasped–background pair (i, j). Given N_g feature points on the grasped object and N_b feature points in the background, the vectors for all pairs are stacked together to form the final error vector:

$$\dot{\mathbf{p}} = [\dots, \delta\mathbf{p}^{(i,j)}, \dots]^\top,$$

where the concatenation spans all $i = 1 \dots N_g$ and $j = 1 \dots N_b$. This formulation accounts for the fact that, during placing, the grasped object moves rigidly with the camera. Directly comparing its features between the current and target images can result in non-zero residual error in $\dot{\mathbf{p}}$ and preventing the convergence of the visual servoing controller.

5.2 Results

5.2.1 Model Comparison

Position Accuracy

Figure 5.6 compares the positional performance of TP-TF, TP-GMMs, TP-ProMPs, CNMP, and CNEP. It is important to note that the TP-TF results shown here exclude visual servoing, as that module is only employed during task reproduction. The proposed TP-TF model demonstrates superior performance in Subtasks 1 and 2. For Subtask 3, TP-ProMPs achieves the best accuracy, with TP-TF performing slightly worse. Across all subtasks, CNMP and CNEP consistently yield the lowest performance.

Sample trajectory predictions for each method are visualized in Figure 5.8. It is evident that TP-TF, TP-ProMPs, and TP-GMMs are task-parameterized, successfully adapting their predicted trajectories to the varying spatial configurations of the objects. For instance, in Subtask 1—where the nut must be picked up and inserted into the jig—these methods consistently generate trajectories approaching the nut across different scenes. In contrast, CNMP and CNEP fail to adapt effectively, suggesting that task parameterization based on high-dimensional image embeddings presents greater challenges and may require larger training datasets.

Moreover, TP-TF and TP-ProMPs produce notably smoother trajectories, with TP-TF remaining consistently closer to the ground truth—particularly near object interaction points. This proximity is critical for achieving successful task reproduction in physical settings.

Orientation Accuracy

Figure 5.7 presents the orientation prediction performance of all evaluated models. In Subtasks 1 and 2, all methods yield low NDQ values, as orientation remains largely unchanged

during these phases. However, in Subtask 3—where continuous orientation change is required during the screwing operation—TP-TF and TP-ProMPs significantly outperform the other models.

Figure 5.9 shows a representative example of quaternion predictions for Subtask 3. Both TP-TF and TP-ProMPs generate smooth orientation trajectories that closely follow the ground truth, whereas the remaining methods exhibit noisier and less accurate predictions.

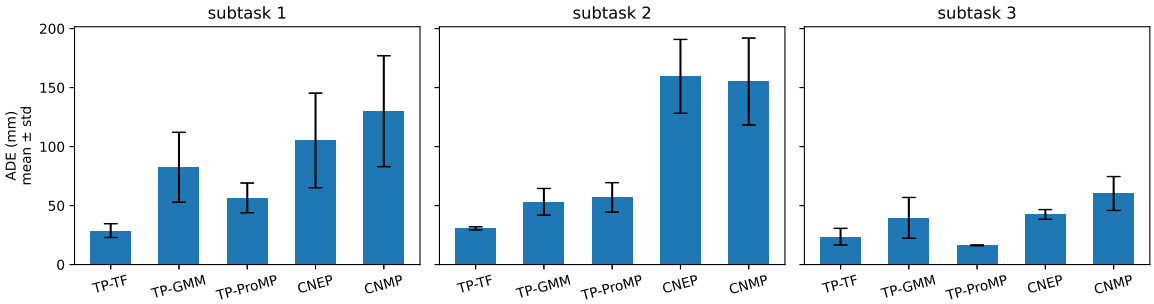


Figure 5.6: Position Comparison

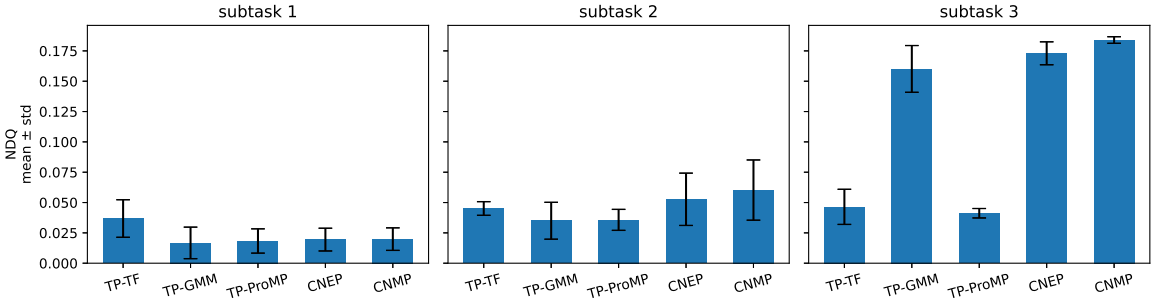


Figure 5.7: Orientation Comparison

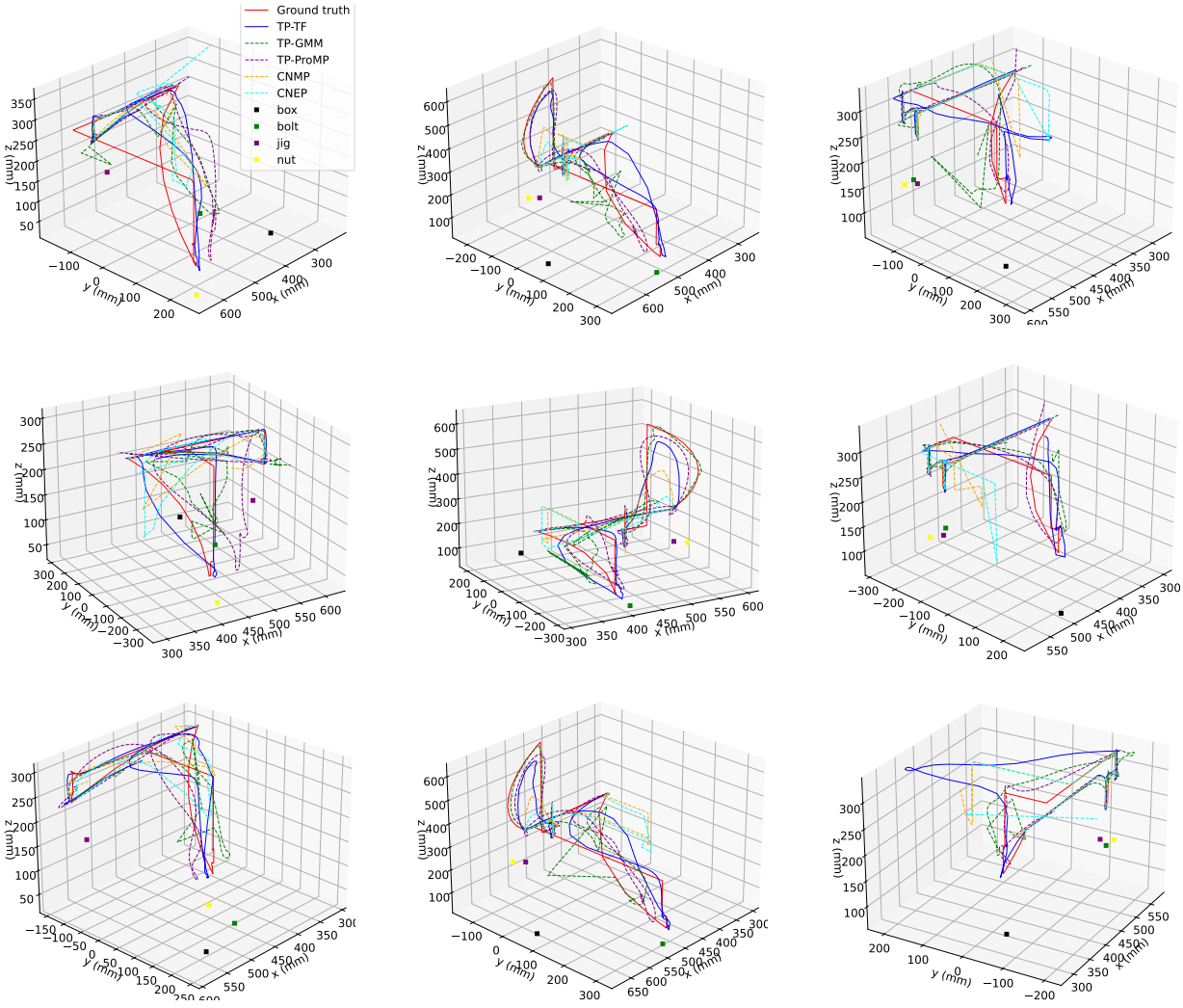


Figure 5.8: **Sample Prediction Trajectories for Each Subtask. First column:** Subtask 1. **Second column:** Subtask 2. **Third column:** Subtask 3. Human-demonstrated trajectories are shown as solid red lines. Squares indicate the positions of the objects in the scene.

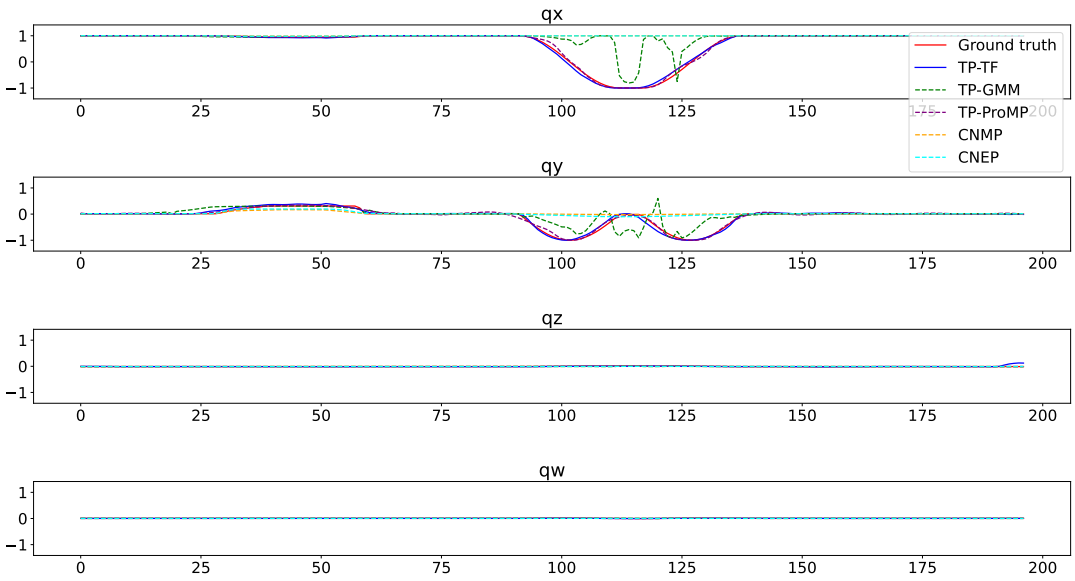


Figure 5.9: Sample quaternion predictions for Subtask 3.

5.2.2 Success rate

TP-Transformer only

A TP-TF model, as described in Section 5.1.6, is trained using 20 demonstration trajectories. The trained model is then used to autonomously reproduce the task over 20 trials, with the success rates for each subtask summarized in Table 5.1. In each trial, the bolt, nut, and box are randomly positioned within the workspace to ensure variability in object poses. During execution, the robot arm is controlled using the position and orientation trajectories predicted by the TP-TF model, while the gripper is actuated based on the predicted gripper state trajectory.

As shown in Table 5.1, the subtask label prediction achieved a success rate of 100%, indicating that the network reliably distinguishes between different subtasks based on the provided task parameters. However, the overall success rate for the complete assembly task was 0%, primarily due to the 0% success rate in the placing phase of Subtask 1. During testing, it was observed that the predicted trajectories were generally effective in adapting to updated object poses and approaching the correct target locations. However, Subtask 1 presented a significant challenge due to the tight tolerance between the nut and the jig

hole, which is only 2–3 mm wider than the nut itself. As a result, even small horizontal deviations of a few millimeters often lead to task failure.

Additionally, grasping the nut with rubber-tipped gripper fingers introduced further complications. The nut tended to adhere slightly to the gripper tip, which can cause failures during place—even when the horizontal positioning was accurate. Successful completion of this subtask therefore, requires the nut to be inserted into the jig hole with high positional precision both horizontally and vertically. Although similar issues may affect the other subtasks, the impact was most pronounced in Subtask 1 due to the tighter spatial constraints.

Subtasks 2 and 3 exhibited significantly higher success rates (70% and 60%, respectively), attributed to the larger size of the manipulated objects and the greater spatial tolerance of the corresponding placement targets, both of which contributed to increased robustness against trajectory prediction errors.

Table 5.1: Success rates using TP-Transformer only evaluated over 20 test trials.

Subtask	**Grasping**	**Screwing**	**Placing**
Subtask 1	65%	–	0%
Subtask 2	90%	–	70%
Subtask 3	100%	80%	60%
Subtask Label		100%	
Overall Task		0%	

TP-Transformer + Visual Servoing

To improve precision and overall task success, visual servoing (as described in Section 5.1.9) was integrated into the system alongside the TP-TF model. The resulting success rates are reported in Table 5.2. Visual servoing significantly enhanced performance across all subtasks. Notably, the success rate of the placing phase in Subtask 1 increased to 75%, and the overall assembly task success rate rised to 70%, demonstrating the effectiveness of visual servoing in improving fine-grained control during manipulation.

Table 5.2: Success rates using TP-Transformer + Visual Servoing evaluated over 20 test trials.

Subtask	**Grasping**	**Screwing**	**Placing**
Subtask 1	90%	–	75%
Subtask 2	100%	–	100%
Subtask 3	100%	90%	90%
Subtask Label		100%	
Overall Task		70%	

Discussion

The proposed TP-TF model demonstrates superior performance in Subtasks 1 and 2, while showing slightly lower accuracy compared to TP-ProMPs in Subtask 3. As illustrated in the bottom-right panel of Figure 5.8, the TP-TF prediction exhibits a noticeable deviation prior to placing the assembled bolt–nut pair into the box. This deviation may be attributed to the winner-takes-all strategy employed during data augmentation, which can induce dominance shifts between reference frames and result in movement patterns not present in the original demonstrations (see Section 5.1.5). While this issue could potentially be mitigated by selecting a smaller λ for Subtask 3, a uniform value was used across all subtasks for the sake of consistency and simplicity. Another contributing factor may be the relatively low weight assigned to this portion of the trajectory in the loss function (see Subsection 5.1.6), due to its temporal distance from the manipulation phase. Despite this deviation, the success rate test confirms that it does not negatively impact task execution.

CNMP and CNEP consistently yield the lowest performance across all subtasks. Notably, in their original evaluation [142], these models were applied to a simplified real-world task involving pick-and-place of wine glasses. In that setting, the initial positions of the wine glasses were fixed, and only the grasping point varied (e.g., rim or stem), with fixed placement locations among four discrete options. This constrained configuration significantly reduces task variability and complexity. In contrast, the bolt–nut assembly task used in this study involves continuously varying object positions and orientations, resulting in a much larger and more complex task space. This difference likely explains why CNMP and CNEP fail to generalize effectively in this context. While parameterizing tasks using high-dimensional image embeddings is more general in principle, it also introduces a more difficult learning problem, typically requiring substantially more demonstrations.

The success rate evaluation indicates that TP-TF alone is sufficient for subtasks with greater spatial tolerance, such as Subtasks 2 and 3 (see Table 5.1). When combined with visual servoing, performance improves markedly across all subtasks—most notably in Subtask 1—highlighting the effectiveness of vision-based feedback in precision-critical operations (see this video for an example of reproduction with and without visual servoing). To further enhance success rates, future work could incorporate a wrist-mounted force/torque sensor, enabling force-based control strategies during contact-rich phases such as placing and screwing. This would allow the system to dynamically respond to physical interaction forces, improving robustness in tasks requiring fine manipulation.

5.3 Conclusions

The proposed TP-TF model demonstrates superior performance in 6D trajectory modeling across three real-world subtasks when compared to traditional PbD methods—including TP-GMMs and TP-ProMPs—under limited-data conditions and identical task parameterization (i.e., object poses). It also significantly outperforms neural network-based methods such as CNMP and CNEP, which rely on high-dimensional image embeddings to encode task variation. These results lead to three key insights:

1. TP-TF provides a stronger alternative to traditional PbD approaches for 6D trajectory modeling.

2. Unlike prior transformer-based models, TP-TF directly predicts gripper trajectories from object poses supplied by an external vision system. This aligns it more closely with classical PbD frameworks and enables effective data augmentation, thereby reducing the number of required demonstrations.

3. The performance gap between TP-TF and image-based models such as CNMP/CNEP suggests that the large data requirements of deep learning-based trajectory models are primarily due to perceptual factors rather than control factors.

In addition, TP-TF was trained to handle the full multi-step assembly task, predicting 6D gripper poses, gripper states, and subtask labels in a unified framework. The success rates exceeding 60% for Subtasks 2 and 3 (Table 5.1) demonstrated that TP-TF alone is well-suited for tasks with relatively high spatial tolerance, such as bin picking. With the integration of visual servoing, the success rate of the entire task improved to 70%

(Table 5.2), highlighting the system's capability in addressing more demanding multi-stage manipulation tasks.

For context, GATO [113] achieves a 60% success rate in single-step stacking after being trained on 387k simulated trajectories, 15k real-world trajectories, and an additional 35k episodes of fine-tuning—alongside large-scale vision and language data. Similarly, RT-1 [14] reaches 67% success in long-horizon kitchen tasks after being trained with 130k demonstrations. In contrast, TP-TF achieves comparable performance with far fewer demonstrations and no language or simulation pretraining, underscoring its data efficiency.

Chapter 6

Conclusion

This thesis addressed the challenge of enabling robots to perform complex, multi-step manipulation tasks in unstructured environments with limited data. The core contribution lies in the development and evaluation of learning-based methods that integrate the strengths of classical PbD techniques with modern deep learning approaches.

The work began by proposing a two-level hierarchical framework. At the high level, VSA was used to abstract symbolic actions from sensory data, enabling interpretable and biologically plausible planning. At the low level, TP-ProMPs were used to model 6D trajectories conditioned on object-centric task parameters. This approach demonstrated superior performance in modeling position and orientation jointly compared to other established PbD baselines. Overall, the modular architecture demonstrated that meaningful high-level action sequences can be inferred and precise low-level motions can be learned, both from a limited number of demonstrations.

Inspired by the task parameterization approach in TP-GMMs and TP-ProMPs, this thesis introduced the TP-TF, a unified model capable of jointly predicting subtask labels, gripper pose trajectories, and gripper states from a small number of demonstrations, thereby integrating high-level symbolic action planning and low-level trajectory modeling within a single framework. By decoupling perception from trajectory modeling, TP-TF conditions directly on object poses, enabling data augmentation and preserving the data efficiency of TP-ProMPs while leveraging the modeling capacity, expressiveness, and flexibility of transformer architectures. Furthermore, by using explicit object pose and type information as input, the network offers greater interpretability compared to other neural alternatives that rely on end-to-end image-based inputs. The model significantly outperforms both traditional PbD methods and neural network baselines on a challenging

real-world bolt–nut assembly task. This task requires very low tolerance, as the jig hole is only 2–3 mm wider than the nut. When combined with a visual servoing module for precision-critical phases, the proposed system achieves a 70% reproduction success rate with only 20 training demonstrations.

These findings support three key conclusions. First, TP-TF serves as a superior replacement for traditional PbD methods such as TP-GMMs and TP-ProMPs, offering greater expressiveness and flexibility. Its primary drawback lies in training time: as shown in a previous collaborative study [20], traditional PbD methods typically require only tens of milliseconds to seconds to train, whereas TP-TF may take several hours. However, TP-TF remains competitive at inference time, operating within seconds. Unless training time is a critical constraint, TP-TF is the preferable choice due to its superior performance and capacity.

Second, control, when decoupled from perception, can be learned efficiently from a small number of demonstrations when task parameters are explicitly provided and data augmentation is employed. Many recent neural models [113, 14, 149] aim to learn end-to-end policies directly from raw inputs such as images (and sometimes text). While these approaches minimize manual engineering, they suffer from several limitations: (1) they exhibit poor sample efficiency, often requiring massive datasets (e.g., [14] reports a 67% success rate on kitchen tasks after 130k demonstrations); (2) they lack interpretability, complicating debugging and failure analysis; (3) they generalize poorly across tasks or environments, requiring full retraining when conditions change; and (4) they limit the use of data augmentation, since generating corresponding outputs for synthetic inputs is nontrivial. In contrast, TP-TF decouples perception and control, allowing each to be optimized independently. This reflects how humans learn—by acquiring reusable perceptual skills (e.g., object recognition or depth estimation) before learning downstream control tasks. These insights suggest a shift in focus toward developing modular, general-purpose perceptual models that can support a variety of downstream tasks, rather than relying solely on end-to-end solutions.

Third, symbolic and continuous aspects of robot behavior can be effectively integrated through hybrid or unified learning architectures. The TP-TF, in particular, demonstrates that it is possible to collapse the symbolic–trajectory hierarchy into a single model, simplifying deployment while maintaining generalization.

In summary, the main contributions of this work are:

1. The introduction of a two-level method that combines high-level action planning, using negative p-norm clustering and VSA, with low-level trajectory modeling through

a PbD-based approach capable of jointly representing position and orientation.

2. The proposal of a unified model, TP-TF, that integrates both symbolic action planning and trajectory modeling, and serves as a middle ground between Transformer-based and PbD-based methods by offering both sample efficiency and flexibility.

3. An empirical demonstration that the data demands of Transformer networks arise primarily from perception, rather than control.

Future work may extend this approach in several directions. First, incorporating force signals could further improve reproduction success rates, particularly in contact-sensitive phases such as insertion or screwing. Second, the visual servoing module could be replaced with Reinforcement Learning (RL). RL is suitable for two reasons: (1) grasping and placing are generic actions that transfer across many tasks, making the investment of large numbers of interactions worthwhile; and (2) refining gripper pose with RL requires only small adjustments, which keeps exploration safe. Finally, applying the framework to a broader range of contact-rich and structurally diverse tasks would provide a more comprehensive evaluation of its generality and robustness.

Ultimately, this research contributes toward the broader objective of developing data-efficient, interpretable, and scalable learning systems for autonomous robotic manipulation in real-world environments.

References

[1] Mongi A. Abidi and T Chandra. A new efficient and direct solution for pose estimation using quadrangular targets: Algorithm and evaluation. *IEEE transactions on pattern analysis and machine intelligence*, 17(5):534–538, 1995.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Miguel Arduengo, Adrià Colomé, Joan Lobo-Prat, Luis Sentis, and Carme Torras. Gaussian-process-based robot learning from demonstration. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–14, 2023.

[4] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[5] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[6] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[7] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning. *Lazy learning*, pages 11–73, 1997.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 404–417. Springer, 2006.

[9] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992.

[10] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

[11] Peter Blouw, Chris Eliasmith, and Bryan P Tripp. A scaleable spiking neural model of action planning. In *CogSci*, pages 1583–1588, 2016.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[13] B Boser. Handwritten digit recognition with a backpropagation network. *Advances in Neural Information Processing Systems II*, pages 396–404, 1990.

[14] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[15] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[16] Yannick Bukschat and Marcus Vetter. Efficientpose: An efficient, accurate and scalable end-to-end 6d multi object pose estimation approach. *arXiv preprint arXiv:2011.04307*, 2020.

[17] Sylvain Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent service robotics*, 9:1–29, 2016.

[18] Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.

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

[20] Yinghan Chen, Xueyang Yao, and Bryan Tripp. Deep networks for few-shot manipulation learning from scratch. *Robotics and Autonomous Systems*, page 105056, 2025.

[21] PETER. CORKE. *ROBOTICS, VISION AND CONTROL: Fundamental Algorithms in Python*. SPRINGER, 2022.

[22] Peter I Corke, Witold Jachimczyk, and Remo Pillat. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer, 2011.

[23] Neil Dantam, Irfan Essa, and Mike Stilman. Linguistic transfer of human assembly tasks to robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 237–242. IEEE, 2012.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[25] Rüdiger Dillmann. Teaching and learning of robot tasks via observation of human performance. *Robotics and Autonomous Systems*, 47(2-3):109–116, 2004.

[26] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[27] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-e: An embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, pages 8469–8488. PMLR, 2023.

[28] Staffan Ekvall and Danica Kragic. Learning task models from multiple human demonstrations. In *ROMAN 2006-The 15th IEEE International Symposium on Robot and Human Interactive Communication*, pages 358–363, 2006.

[29] Staffan Ekvall and Danica Kragic. Robot learning from demonstration: a task-level planning approach. *International Journal of Advanced Robotic Systems*, 5(3):33, 2008.

[30] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–5, nov 2012.

[31] Marco Ewerton, Gerhard Neumann, Rudolf Lioutikov, Heni Ben Amor, Jan Peters, and Guilherme Maeda. Learning multiple collaborative tasks with a mixture of interaction primitives. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1535–1542. IEEE, 2015.

[32] Yunis Fanger, Jonas Umlauft, and Sandra Hirche. Gaussian processes for dynamic movement primitives with application in knowledge-based cooperation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3913–3919. IEEE, 2016.

[33] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[34] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[35] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[36] Gui Fu, Linyi Fang, Liwen Liu, Xinyu Zhu, and You Wang. An image-based visual servoing control method for uavs based on fuzzy logic. *Advances in Mechanical Engineering*, 15(4):16878132231167238, 2023.

[37] Ross W. Gayler. Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. *Proceedings of the Joint International Conference on Cognitive Science*, pages 133–138, 2003.

[38] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.

[39] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[40] Sebastian Gomez-Gonzalez, Gerhard Neumann, Bernhard Schölkopf, and Jan Peters. Adaptation and robust learning of probabilistic movement primitives. *IEEE Transactions on Robotics*, 36(2):366–379, 2020.

[41] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[42] Daniel H Grollman and Odest Chadwicke Jenkins. Incremental learning of subtasks from unsegmented demonstration. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 261–266. IEEE, 2010.

[43] Siddhant Haldar, Jyothish Pari, Anant Rai, and Lerrel Pinto. Teach a robot to fish: Versatile imitation from one minute of demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[44] T. Harley. *Psychology of Language: From data to theory*. Psychology Press, 4th edition, 2014.

[45] Bradley Hayes and Brian Scassellati. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5469–5476. IEEE, 2016.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[47] Tomas Hodan, Martin Sundermeyer, Yann Labbe, Van Nguyen Nguyen, Gu Wang, Eric Brachmann, Bertram Drost, Vincent Lepetit, Carsten Rother, and Jiri Matas. Bop challenge 2023 on detection segmentation and pose estimation of seen and unseen rigid objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5610–5619, 2024.

[48] Radu Horaud, Bernard Conio, Olivier Leboulleux, and Bernard Lacolle. An analytic solution for the perspective 4-point problem. *Computer Vision, Graphics, and Image Processing*, 47(1):33–44, 1989.

[49] Yanlong Huang, Fares J Abu-Dakka, João Silvério, and Darwin G Caldwell. Generalized orientation learning in robot task space. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2531–2537. IEEE, 2019.

[50] Yanlong Huang, Leonel Rozo, Joao Silvério, and Darwin G Caldwell. Kernelized movement primitives. *The International Journal of Robotics Research*, 38(7):833–852, 2019.

[51] 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.

[52] Du Q Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35:155–164, 2009.

[53] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[54] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 1398–1403. IEEE, 2002.

[55] Noémie Jaquier, David Ginsbourger, and Sylvain Calinon. Learning from demonstration with model-based gaussian process. In *Conference on Robot Learning*, pages 247–257. PMLR, 2020.

[56] Mahmoud Jeddi and Ahmad Reza Khoogar. A modified eye-in-hand stereo visual control for grasping unknown objects via scara robot. *IET Image Processing*, 17(7):2015–2031, 2023.

[57] M. Jones and D. Mewhort. Representing word meaning and order information in a composite holographic lexicon. *Psychological Review*, 114(1):1–37, 2007.

[58] Khaled Kaaniche, Osama El-Hamrawy, Nasr Rashid, Mohammed Albekairi, and Hassen Mekki. Mobile robot control based on 3d visual servoing: A new approach combining pose estimation by neural network and differential flatness. *Applied Sciences*, 12(12):6167, 2022.

[59] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *Proceedings of the IEEE international conference on computer vision*, pages 1521–1529, 2017.

[60] Wadim Kehl, Fausto Milletari, Federico Tombari, Slobodan Ilic, and Nassir Navab. Deep learning of local rgb-d patches for 3d object detection and 6d pose estimation. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, pages 205–220. Springer, 2016.

[61] Azarakhsh Keipour, Guilherme AS Pereira, Rogerio Bonatti, Rohit Garg, Puru Rastogi, Geetesh Dubey, and Sebastian Scherer. Visual servoing approach to autonomous uav landing on a moving vehicle. *Sensors*, 22(17):6549, 2022.

[62] Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*, pages 2859–2867, 2013.

[63] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. *Advances in neural information processing systems*, 21:849–856, 2008.

[64] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012.

[65] Sanjay Krishnan, Animesh Garg, Sachin Patil, Colin Lea, Gregory Hager, Pieter Abbeel, and Ken Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. *The International journal of robotics research*, 36(13-14):1595–1618, 2017.

[66] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[67] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[68] Yann Labbé, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Cosypose: Consistent multi-view multi-object 6d pose estimation. In *Computer Vision–ECCV 2020:*

16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVII 16, pages 574–591. Springer, 2020.

[69] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

[70] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Ep n p: An accurate o (n) solution to the p n p problem. *International journal of computer vision*, 81:155–166, 2009.

[71] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018.

[72] Chenping Li, Baoquan Li, Runhua Wang, and Xuebo Zhang. A survey on visual servoing for wheeled mobile robots. *International Journal of Intelligent Robotics and Applications*, 5(2):203–218, 2021.

[73] Ge Li, Zeqi Jin, Michael Volpp, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. Prodmp: A unified perspective on dynamic and probabilistic movement primitives. *IEEE Robotics and Automation Letters*, 8(4):2325–2332, 2023.

[74] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. Deepim: Deep iterative matching for 6d pose estimation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 683–698, 2018.

[75] Ying Siu Liang, Damien Pellier, Humbert Fiorino, and Sylvie Pesty. A framework for robot programming in cobotic environments: First user experiments. In *Proceedings of the 3rd International Conference on Mechatronics and Robotics Engineering*, pages 30–35, 2017.

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

[77] Rudolf Lioutikov, Gerhard Neumann, Guilherme Maeda, and Jan Peters. Probabilistic segmentation applied to an assembly task. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 533–540. IEEE, 2015.

[78] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[79] David G Low. Distinctive image features from scale-invariant keypoints. *Journal of Computer Vision*, 60(2):91–110, 2004.

[80] Takashi Makiishi and Hiroshi Noborio. Sensor-based path-planning of multiple mobile robots to overcome large transmission delays in teleoperation. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, volume 4, pages 656–661. IEEE, 1999.

[81] Simon Manschitz, Jens Kober, Michael Gienger, and Jan Peters. Learning to sequence movement primitives from demonstrations. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4414–4421. IEEE, 2014.

[82] Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21(9):1281–1289, 2018.

[83] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[84] Franziska Meier, Evangelos Theodorou, Freek Stulp, and Stefan Schaal. Movement segmentation using a primitive library. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3407–3412. IEEE, 2011.

[85] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR), Workshop Track*, 2013.

[86] Abolfazl Mohebbi, Mohammad Keshmiri, and Wen Fang Xie. An eye-in-hand stereo visual servoing for tracking and catching moving objects. In *Proceedings of the 33rd Chinese Control Conference*, pages 8570–8577. IEEE, 2014.

[87] Yoan Mollard, Thibaut Munzer, Andrea Baisero, Marc Toussaint, and Manuel Lopes. Robot programming from demonstration, feedback, and transfer. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[88] Cory Myers, Lawrence Rabiner, and Aaron Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(6):623–635, 1980.

[89] Cory Myers, Lawrence R. Rabiner, and Aaron E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING*, 28(2):623–635, 1980.

[90] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[91] Chrystopher L Nehaniv, Kerstin Dautenhahn, and Kerstin Dautenhahn. *Imitation in animals and artifacts*. MIT press, 2002.

[92] Victor Reyes Osorio, Rajan Iyengar, Xueyang Yao, Presish Bhattachan, Adrian Ragobar, Nolan Dey, and Bryan Tripp. 37,000 human-planned robotic grasps with six degrees of freedom. *IEEE Robotics and Automation Letters*, 5(2):3346–3351, 2020.

[93] Victor Reyes Osorio, Rajan Iyengar, Xueyang Yao, Presish Bhattachan, Adrian Ragobar, Nolan Dey, and Bryan Tripp. 37,000 human-planned robotic grasps with six degrees of freedom. *IEEE Robotics and Automation Letters*, 5(2):3346–3351, 2020.

[94] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42:529–551, 2018.

[95] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. *Advances in neural information processing systems*, 26, 2013.

[96] Jyothish Pari, Nur Muhammad Shafiullah, Sridhar Pandian Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation. In *Proceedings of the Robotics: Science and Systems (RSS)*, 2022. Short talk, paper #010.

[97] Kiru Park, Timothy Patten, and Markus Vincze. Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7668–7677, 2019.

[98] 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.

[99] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4561–4570, 2019.

[100] Arul Selvam Periyasamy, Arash Amini, Vladimir Tsaturyan, and Sven Behnke. Yolopose v2: Understanding and improving transformer-based 6d pose estimation. *Robotics and Autonomous Systems*, 168:104490, 2023.

[101] Affan Pervez, Arslan Ali, Jee-Hwan Ryu, and Dongheui Lee. Novel learning from demonstration approach for repetitive teleoperation tasks. In *2017 IEEE World Haptics Conference (WHC)*, pages 60–65. IEEE, 2017.

[102] Richard Alan Peters, Christina L Campbell, William J Bluethmann, and Eric Huber. Robonaut task learning through teleoperation. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 2806–2811. IEEE, 2003.

[103] T A Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–41, jan 1995.

[104] Michael Przystupa, Faezeh Haghverd, Martin Jagersand, and Samuele Tosatto. Deep probabilistic movement primitives with a bayesian aggregator. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3704–3711. IEEE, 2023.

[105] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *IEEE Transactions on pattern analysis and machine intelligence*, 21(8):774–780, 1999.

[106] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[107] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[108] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[109] Karinne Ramirez-Amaro, Michael Beetz, and Gordon Cheng. Automatic segmentation and recognition of human activities from observation based on semantic reasoning. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5043–5048. IEEE, 2014.

[110] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3, 2020.

[111] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:297–330, 2020.

[112] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[113] Scott Reed, Konrad Zolna, et al. A generalist agent. *Transactions on Machine Learning Research*, 2022. Featured Certification, Outstanding Certification.

[114] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[115] Leonel Rozo and Vedant Dave. Orientation probabilistic movement primitives on riemannian manifolds. In *Conference on Robot Learning*, pages 373–383. PMLR, 2022.

[116] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[117] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE, 2001.

[118] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach (global 3rd edition). *Essex: Pearson*, pages 122–125, 2016.

[119] Thiusius Rajeeth Savarimuthu, Anders Glent Buch, Christian Schlette, Nils Wantia, Jürgen Roßmann, David Martínez, Guillem Alenyà, Carme Torras, Ales Ude, Bojan Nemec, et al. Teaching a robot the semantics of assembly tasks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(5):670–692, 2017.

[120] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

[121] Muhammet Yunus Seker, Mert Imre, Justus H Piater, and Emre Ugur. Conditional neural movement primitives. In *Robotics: Science and Systems*, volume 10, 2019.

[122] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1715–1725, 2016.

[123] Joao Silvério, Leonel Rozo, Sylvain Calinon, and Darwin G Caldwell. Learning bimanual end-effector poses from demonstrations using task-parameterized dynamical systems. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 464–470. IEEE, 2015.

[124] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[125] P. Smolensky and G. Legendre. *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*, volume 1. MIT Press, 2006.

[126] Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624, 2011.

[127] Ahmet Tekden, Marc Peter Deisenroth, and Yasemin Bekiroglu. Neural field movement primitives for joint modelling of scenes and motions. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3648–3655. IEEE, 2023.

[128] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 292–301, 2018.

[129] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *Conference on Robot Learning (CoRL)*, 2018.

[130] Aleš Ude, Bojan Nemec, Tadej Petrić, and Jun Morimoto. Orientation in cartesian space dynamic movement primitives. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2997–3004. IEEE, 2014.

[131] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[132] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[133] David Vogt, Simon Stepputtis, Steve Grehl, Bernhard Jung, and Heni Ben Amor. A system for learning continuous human-robot interactions from human-human demonstrations. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2882–2889. IEEE, 2017.

[134] Gu Wang, Fabian Manhardt, Federico Tombari, and Xiangyang Ji. Gdr-net: Geometry-guided direct regression network for monocular 6d object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16611–16621, 2021.

[135] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

[136] David Whitney, Eric Rosen, Elizabeth Phillips, George Konidaris, and Stefanie Tellex. Comparing robot grasping teleoperation across desktop and virtual reality with ros reality. In *Robotics Research*, pages 335–350. Springer, 2020.

[137] David E Wilkins. *Practical planning: extending the classical AI planning paradigm*. Elsevier, 2014.

[138] Qian Xiang, Xiaodan Wang, Yafei Song, Lei Lei, Rui Li, and Jie Lai. One-dimensional convolutional neural networks for high-resolution range profile recognition via adaptively feature recalibrating and automatically channel pruning. *International Journal of Intelligent Systems*, 36(1):332–361, 2021.

[139] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[140] Fan Xie, Alexander Chowdhury, M De Paolis Kaluza, Linfeng Zhao, Lawson Wong, and Rose Yu. Deep imitation learning for bimanual robotic manipulation. *Advances in neural information processing systems*, 33:2327–2337, 2020.

[141] Xueyang Yao, Yinghan Chen, and Bryan Tripp. Improved generalization of probabilistic movement primitives for manipulation trajectories. *IEEE Robotics and Automation Letters*, 9(1):287–294, 2023.

[142] Yigit Yildirim and Emre Ugur. Conditional neural expert processes for learning movement primitives from demonstration. *IEEE Robotics and Automation Letters*, 2024.

[143] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. Dpod: 6d pose object detector and refiner. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1941–1950, 2019.

[144] Martijn JA Zeestraten, Ioannis Havoutis, Joao Silvério, Sylvain Calinon, and Darwin G Caldwell. An approach for imitation learning on riemannian manifolds. *IEEE Robotics and Automation Letters*, 2(3):1240–1247, 2017.

[145] Juyong Zhang, Yuxin Yao, and Bailin Deng. Fast and robust iterative closest point. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3450–3466, 2021.

[146] Zhou and Chellappa. Computation of optical flow using a neural network. In *IEEE 1988 international conference on neural networks*, pages 71–78. IEEE, 1988.

[147] Zuyuan Zhu and Huosheng Hu. Robot learning from demonstration in robotic assembly: A survey. *Robotics*, 7(2):17, 2018.

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

[149] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

APPENDICES

Appendix A

Pose in 3D

A.1 Position in $\mathbb{R}^3$

Throughout this thesis, a right-handed Cartesian coordinate system is used to describe the position of a point P in three-dimensional space. Let the basis vectors of the coordinate system be $\hat{x}$, $\hat{y}$, $\hat{z}$. These vectors satisfy

$$\hat{z} = \hat{x} \times \hat{y}, \quad \hat{x} = \hat{y} \times \hat{z}, \quad \hat{y} = \hat{z} \times \hat{x},$$

where $\times$ denotes the cross product. The position of P is expressed by the column vector $\mathbf{p} = [x, y, z]^\mathsf{T}$, where x, y, z are the signed distances along the respective axes.

A.2 Orientation in $\mathbb{R}^3$

Figure A.1 shows an example of a reference frame is rotated in 3D space. It could be seen that different orders of rotation yields different final orientations. Rotation in 3D space has several different forms of representation, including: Rotation matrix, Euler angles, rotation vector (or axis-angle representation), and unit quaternions.

A.2.1 Rotation Matrix

A rotation in 3D is represented by a 3×3 orthogonal matrix. Consider two reference frames A and B with different orientations and $\mathbf{R}_B^A$ is the rotation matrix that represent

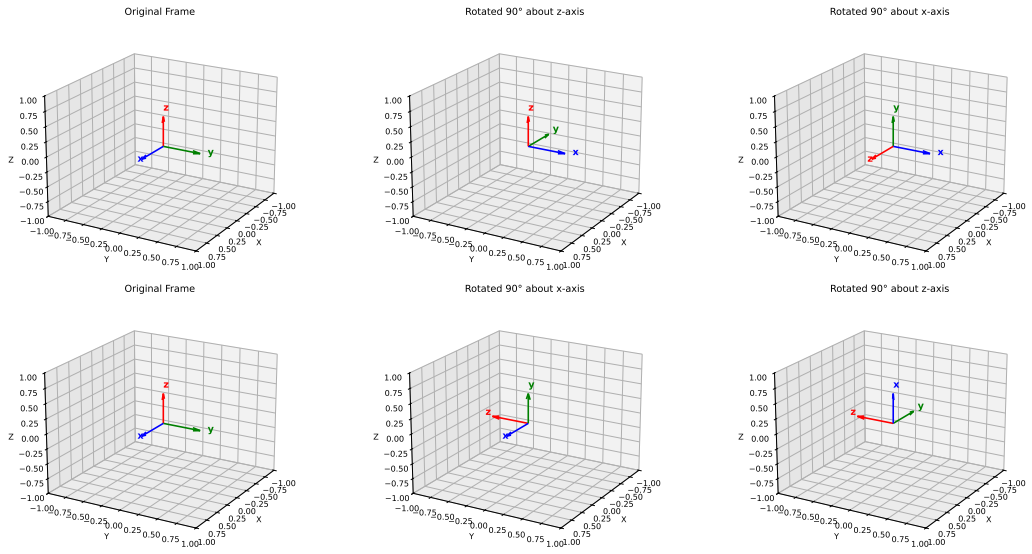


Figure A.1: **An Example of Rotation Order in 3D. Top row:** The reference frame is first rotated 90° about the z-axis, followed by a 90° rotation about the x-axis. **Bottom row:** The reference frame is first rotated 90° about the x-axis, followed by a 90° rotation about the z-axis.

the orientation of B in the frame of A. The matrix $\mathbf{R}_B^A$ then converts a point $\mathbf{p}^B$ in frame B to frame A

$$\mathbf{p}^A = \mathbf{R}_B^A \mathbf{p}^B.$$

Since a rotation matrix is orthogonal, its inverse equals its transpose, i.e. $\mathbf{R}^{-1} = \mathbf{R}^\top$. Converting a point from frame A to frame B can be achieved by

$$\begin{aligned} \mathbf{p}^B &= (\mathbf{R}_B^A)^{-1} \mathbf{p}^A \\ &= (\mathbf{R}_B^A)^\top \mathbf{p}^A \\ &= \mathbf{R}_A^B \mathbf{p}^A \end{aligned}$$

Elementary rotations about the principal axes by an angle θ are given by

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$
$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$
$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Since SO(3) is closed under multiplication, composite rotations are obtained by matrix multiplication. Take the top row of Figure A.1 as an example; the net rotation is

$$\begin{aligned} \mathbf{R} &= \mathbf{R}_z(\frac{\pi}{2})\mathbf{R}_x(\frac{\pi}{2}) \\ &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{aligned}$$

,

whose columns correspond to the axes of the rotated frame expressed in the original coordinates. Direct linear interpolation between rotation matrices can leave SO(3).

A.2.2 Euler angles

Any rigid-body orientation can be decomposed into three sequential rotations about specified axes. Conventions vary: the Z-Y-Z scheme is popular in rigid–body dynamics, whereas roll–pitch–yaw (X-Y-Z) is prevalent in aerospace. For a fixed convention, the orientation is captured by the triple of angles. The price of this compactness is the singularity problem: when two of the rotation axes become collinear in the global frame the representation becomes locally ambiguous. Figure A.2 shows that 2 different sets of Euler angles could express the same rotation at the singularity. Moreover, linear interpolation of Euler angles rarely traces the shortest path on SO(3), often producing unexpected intermediate orientations.

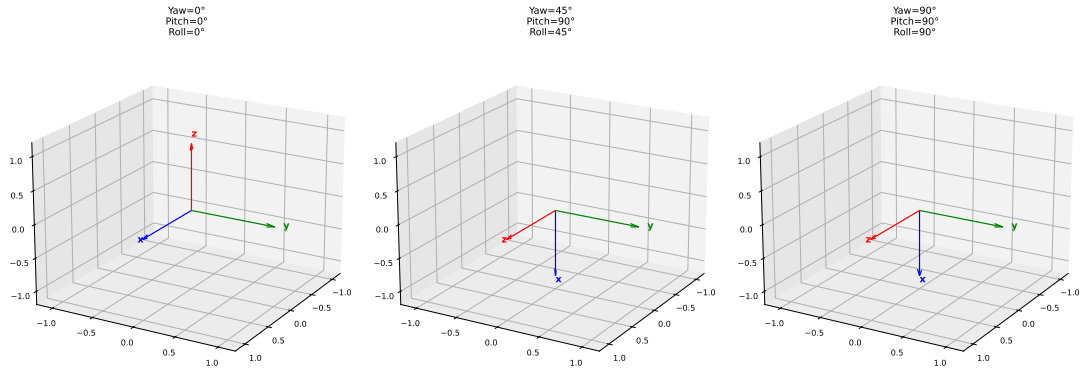


Figure A.2: Ambiguity of Euler angle representation at a singularity

A.2.3 Axis-angle representation (rotation vector)

An alternative is to specify the unit vector $\mathbf{u} = (u_x, u_y, u_z)^\mathsf{T}$ along the rotation axis together with the rotation magnitude θ. A compact 3-vector $\boldsymbol{\theta} = \theta\mathbf{u}$ can be used to represent a rotation, whose corresponding rotation matrix is obtained via Rodrigues' formula:

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)\mathbf{u}\mathbf{u}^T + \sin\theta[\mathbf{u}]_\times,$$

where $\mathbf{I}$ is the identity matrix and $[\mathbf{u}]_\times$ denotes the skew–symmetric matrix of $\mathbf{u}$. As with Euler angles, naive linear interpolation of rotation vectors does not follow the shortest path on SO(3).

A.2.4 Unit Quaternions

Unit quaternions provide a non–singular, compact representation well suited to interpolation. A quaternion is written

$$\mathbf{q} = q_w + q_i\mathbf{i} + q_j\mathbf{j} + q_k\mathbf{k}, \quad ||\mathbf{q}|| = 1,$$

with the imaginary units obeying

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1.$$

A rotation vector $\theta\mathbf{u}$ can be converted to unit quaternion as:

$$\mathbf{q} = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})(u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}).$$

and the corresponding rotation matrix is

$$R = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_zq_w) & 2(q_xq_z + q_yq_w) \\ 2(q_xq_y + q_zq_w) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_xq_w) \\ 2(q_xq_z - q_yq_w) & 2(q_yq_z + q_xq_w) & 1-2(q_x^2+q_y^2) \end{bmatrix}$$

Because quaternions lie on the unit three–sphere S^3, spherical linear interpolation (*slerp*) yields constant–velocity motion along the shortest arc. In practice, simple linear interpolation followed by renormalisation often suffices as a computationally cheaper approximation.

A.3 Pose in $\mathbb{R}^3$

Describing the relative pose of frames requires both the translation between their origins and the rotation that aligns their axes. Throughout this thesis the pose of frame B expressed in frame A is represented by the 4×4 homogeneous transformation$\mathbf{T}_B^A = \begin{bmatrix} \mathbf{R}_B^A & \mathbf{t}_B^A \\ \mathbf{0} & 1 \end{bmatrix}$, where $\mathbf{R}_B^A \in \mathrm{SO}(3)$ is the rotation matrix that orients B relative to A, and $\mathbf{t}_B^A \in \mathbb{R}^3$ is the position of the origin of B written in the coordinates of A.

Given a point whose homogeneous coordinates in frame B are $\tilde{\mathbf{p}}^B = [x^B,\, y^B,\, z^B,\, 1]^\top$, its coordinates in frame A follow from the passive (change–of–coordinates) transformation

$$\tilde{\mathbf{p}}^A = \mathbf{T}_B^A \tilde{\mathbf{p}}^B \tag{A.1}$$

$$\begin{bmatrix} x^A \\ y^A \\ z^A \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_B^A & \mathbf{t}_B^A \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x^B \\ y^B \\ z^B \\ 1 \end{bmatrix}. \tag{A.2}$$

Successive frame changes compose by right multiplication, e.g. $\tilde{\mathbf{p}}^A = \mathbf{T}_B^A\,\mathbf{T}_C^B\,\tilde{\mathbf{p}}^C$. Because $\mathbf{R}_B^A$ is orthonormal, the inverse of a homogeneous transformation is

$$(\mathbf{T}_B^A)^{-1} = \begin{bmatrix} (\mathbf{R}_B^A)^\top & -(\mathbf{R}_B^A)^\top \mathbf{t}_B^A \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}.$$

A.3.1 Passive transformation vs. Active transformation

Equation (A.1) is *passive*: the physical point is fixed in space, while its coordinates change because we describe it in a different frame. In contrast, an *active* transformation rigidly moves the point itself within a single frame. An example shown in Figure A.3 is used to illustrate the distinction. Frames A and B are expressed in a global frame G via

$$\mathbf{T}_A^G = \begin{bmatrix} 0 & -1 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{T}_B^G = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 2 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

This is the passive view. On the other hand, Multiplying each transform by the homogeneous vector $[0,\, 0,\, 0,\, 1]^\top$ yields the Cartesian positions $(2, 0, 1)$ and $(-1, 2, 1)$, respectively—the origins of A and B viewed as active displacements in G.

The rigid motion that carries frame A onto frame B in the global frame is obtained by combining a passive change of coordinates with an active displacement:

$$\mathbf{T}_{A\to B}^G\, \mathbf{T}_A^G = \mathbf{T}_B^G \quad \Longrightarrow \quad \mathbf{T}_{A\to B}^G = \mathbf{T}_B^G\, (\mathbf{T}_A^G)^{-1}.$$

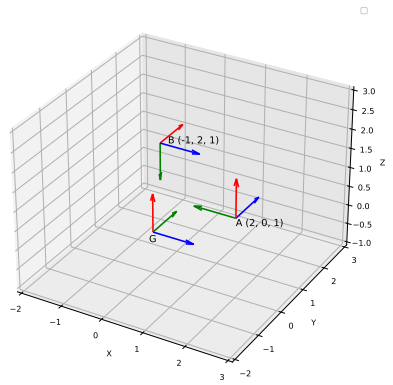


Figure A.3: **Global and Local Coordinate Frames.** Global frame G and two local frames, A and B. The origins of frames A and B are located at $(2, 0, 1)$ and $(-1, 2, 1)$, respectively, with respect to frame G.