EFFICIENT LEARNING AND OPTIMIZATION FOR ROBOTIC MANIPULATOR MOTION GENERATION

DISSERTATION

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

DMYTRO PAVLICHENKO

aus

Odesa, Ukraine

Bonn, Dezember 2024

Angefertigt mit Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

Gutachter / Betreuer	Prof. Dr. Sven Behnke
Gutachter	Prof. Dr. Gerhard Neumann
Tag der Promotion	07.07.2025
Erscheinungsjahr	2025

ABSTRACT

A central goal of robotics research is to develop autonomous systems capable of achieving and ultimately surpassing human-level task performance in unstructured environments. Robotic manipulation plays a critical role in this aspiration. In this thesis, we address the problem of robotic manipulator motion generation with data-driven and optimization-based approaches. The high-dimensional state spaces with complex underlying dynamics and real-time operational constraints pose major challenges. Our methods address these problems and provide solutions to a sequence of interconnected tasks. These include planning and tracking manipulator trajectories followed by dexterous object manipulation.

First, a feed-forward open-loop reference correction policy improves the joint trajectory tracking accuracy. The policy is learned offline in a supervised manner on a small real-world dataset. We propose to incorporate a hardwired one-step future prediction into the model to facilitate planning behavior. Next, we introduce a methodology for learning a closed-loop policy with deep reinforcement learning directly on the real robot. Our policy leverages the advantages of online feedback to significantly improve trajectory tracking accuracy.

Second, we address dual-arm trajectory optimization with multiple constraints. We propose an obstacle cost function based on the estimation of the worst-case overlap volume. Additionally, we handle the closed kinematic chain constraint by subdividing the system into active and passive sub-chains, with an implicit redundancy resolution for the passive sub-chain. These components significantly decrease the method's runtime when optimizing high-dimensional dual-arm trajectories.

Third, we propose a method for learning dexterous pre-grasp manipulation for functional grasping using a human-like hand. The policy is trained with deep reinforcement learning. Our dense multi-component reward function and curriculum avoid the need for expert demonstrations and other costly data collection processes. We propose two target grasp representations and analyze their effects on the behavior of the policy. The policy quickly learns to dexterously manipulate novel object instances of known categories and achieve provided functional grasps that enable object use, such as operating a drill.

We showcase the effectiveness of our methods in simulation and real-world experiments. Our approaches significantly improve trajectory tracking accuracy, quickly generate high-dimensional trajectories that satisfy multiple constraints, and dexterously manipulate complex objects using a human-like hand.

ZUSAMMENFASSUNG

Ein zentrales Ziel der Robotikforschung besteht darin, autonome Systeme zu entwickeln, die in der Lage sind, menschliches Leistungsniveau in unstrukturierten Umgebungen zu erreichen und letztendlich zu übertreffen. Die robotische Manipulation spielt eine entscheidende Rolle in diesem Bestreben. In dieser Dissertation wird das Problem der Bewegungserzeugung von Roboterarmen mit einer Kombination aus datengetriebenen und optimierungsbasierten Ansätzen behandelt. Hochdimensionale Zustandsräume mit komplexen zugrunde liegenden Dynamiken sowie Echtzeitanforderungen stellen wesentliche Herausforderungen dar. Unsere Ansätze adressieren diese Probleme und bieten Lösungen für eine Reihe miteinander verbundener Aufgaben. Dazu gehören die Planung und Ausführung der Trajektorie von Roboterarmen, sowie die geschickte Manipulation von Objekten.

Zunächst verbessert eine Feedforward-Referenzkorrekturstrategie mit offenem Regelkreis die Genauigkeit der Gelenktrajektorie. Die Regelstrategie wird offline in einer überwachten Weise auf wenigen realen Daten gelernt. Wir schlagen vor, eine Vorhersage für den nächsten Zeitschritt fest in das Modell zu integrieren, um ein planungsorientiertes Verhalten zu erleichtern. Anschließend stellen wir eine Methodik vor, um eine geschlossene Regelstrategie mithilfe von Deep Reinforcement Learning direkt auf dem realen Roboter zu erlernen. Unsere Regelstrategie nutzt die Vorteile von Online-Feedback, um die Genauigkeit bei der Abarbeitung der Trajektorien erheblich zu verbessern.

Zweitens behandeln wir die Optimierung von Trajektorien für zweiarmige Roboter unter mehreren Nebenbedingungen. Wir stellen eine Hinderniskostenfunktion vor, die auf der Abschätzung des Überlappungsvolumens im schlimmstmöglichen Fall basiert. Zusätzlich werden die Nebenbedingungen der geschlossenen kinematischen Kette durch die Unterteilung in aktive und passive Teilketten behandelt, wobei eine implizite Redundanzauflösung für die passive Teilkette erfolgt. Diese Komponenten reduzieren die Laufzeit bei der Optimierung hochdimensionaler zwei-armiger Trajektorien erheblich.

Drittens wird eine Methode zur Erlernung geschickter Vorgriffmanipulation für funktionales Greifen mit einer menschenähnlichen Hand durch Deep Reinforcement Learning vorgestellt. Wir schlagen eine mehrkomponentige Belohnungsfunktion mit zugehöriger Trainingsstrategie vor, die den Bedarf an Expertendemonstrationen oder anderen kostspieligen Datenerfassungsprozessen eliminiert. Wir schlagen zwei Repräsentationen für gezielte Griffe vor und analysieren deren Auswirkungen auf das Verhalten der Regelstrategie. Die Re-

gelstrategie lernt effizient, neuartige Objekte bekannter Kategorien geschickt zu manipulieren und funktionale Griffe zu erreichen, die den praktischen Gebrauch von Objekten ermöglichen, wie beispielsweise die Bedienung einer Bohrmaschine.

Wir demonstrieren die Wirksamkeit unserer Methoden in simulierten Experimenten wie auch Versuchen in der realen Welt. Unsere Ansätze verbessern die Genauigkeit bei der Trajektorienumsetzung erheblich, generieren schnell hochdimensionale Trajektorien, unter Beachtung mehrerer Nebenbedingungen, und ermöglichen eine geschickte Manipulation komplexer Objekte mit einer menschenähnlichen Hand.

ACKNOWLEDGMENTS

First, I would like to express my gratitude to Prof. Dr. Sven Behnke for sharing his invaluable experience and for his guidance throughout my research. His profound advice, insightful discussions, and inspiring ideas have played a key role in shaping this thesis.

I would like to thank my colleagues Grzegorz Ficht, Dr. Diego Rodriguez, Dr. Philipp Allgeuer, Dr. Hafez Farazi, Angel Villar-Corrales, Malte Mosbach, Jan Quenzel, Max Schwarz, Christian Lenz, Dr. Simon Bultmann, Dr. Marius Beul, Arul Selvam Periyasamy, Michael Schreiber, Mojtaba Hosseini, André Brandenburger, and Luis Denninger for our memorable performances and achievements at numerous robotic competitions as well as for our collaboration on multiple projects. I also extend my gratitude to all members of the Autonomous Intelligent Systems group for their mutual support and fostering a collaborative and productive environment.

My deepest gratitude goes to my parents Valeriy and Olga, and older brother Evgeniy, for nurturing my enduring curiosity and passion for science. Their unwavering support helped me throughout this challenging path.

This work was supported by the European Union's Horizon 2020 Programme under Grant Agreement 644839 (CENTAURO) and the German Research Foundation (DFG) under grant BE 2556/12 ALROMA in priority programme SPP 1527 Autonomous Learning, and grant BE 2556/16-2 (Research Unit FOR 2535 Anticipating Human Behavior), German Ministry of Education and Research (BMBF) under grant No. 01IS21080, project "Learn2Grasp: Learning Human-like Interactive Grasping based on Visual and Haptic Feedback".

CONTENTS

1 INTRODUCTION

Robots are being employed for an increasingly broad range of tasks due to their advanced automation capabilities. In particular, robotic manipulators, either independently or in combination with mobile platforms, are used to perform tasks ranging from simple pick-and-place operations to complex assembly processes. While automation in highly structured factory settings is largely a solved problem, replicating such performance in unstructured environments remains a significant challenge. Tasks such as assisting in households, facilitating search-and-rescue operations in hazardous areas, or handling various tools continue to pose considerable difficulties. These domains share a high degree of uncertainty. Efficiently acting in such scenarios requires accurate and dexterous robotic manipulator motions that are generated quickly and can generalize across variability in hardware and environments.

In this thesis, we study the three fundamental capabilities of robotic manipulators: trajectory tracking, trajectory generation, and object manipulation. These methods allow robotic manipulators to traverse their workspace with precision, avoiding obstacles and adhering to kinematic constraints. Upon reaching the target location, the manipulator can then interact directly with the environment. A major challenge in motion generation for robotic manipulators is the high-dimensional state spaces. Combined with complex underlying dynamics and computational time constraints, planning and control become significantly more difficult. The methods presented in this thesis leverage recent advancements in the field of artificial intelligence and classical optimization techniques to address these challenges.

Trajectory tracking control of robotic manipulators requires highly accurate dynamics models. When robots are deployed outside of controlled factory settings, particularly in shared workspaces with humans, additional safety measures are often necessary. One such measure is the use of series-elastic actuators, which provide enhanced compliance, ensuring safer interaction between robots and humans. This further complicates system identification. Even when accurate models are available, the use of low-cost hardware introduces limitations. Over time, wear and tear accumulate, affecting the performance negatively. As a result, frequent adjustments to the models are required. Additionally, parameters of classical controllers must be regularly fine-tuned to maintain optimal operation. This process is time-consuming and, to be executed effectively, requires the expertise

of a skilled engineer. Such procedures necessitate stopping the normal operation of the robot, reducing its efficiency.

Many tasks demand not only accurate trajectory tracking but also an ability to avoid obstacles in unstructured environments while adhering to kinematic and dynamic constraints. Manipulators often have a high number of degrees of freedom (DoF), introducing redundancy to expand their effective workspace. Moreover, dual-arm systems are commonly utilized to significantly expand the range of tasks that can be performed. These applications include handling large, heavy objects that exceed the payload capacity of a single arm, as well as scenarios where one arm supports the object while the other conducts complex manipulations. Generating trajectories for efficient motion of multiple high-DoF manipulators under multiple constraints while maintaining low computation times is a complex problem.

Finally, once the robotic manipulator moves through its workspace safely and efficiently, one of the primary objectives is manipulating an object. Interacting with the environment involves making sustained contacts over time, governed by complex dynamics. In particular, grasping objects for functional use, as opposed to simple pick-and-place tasks, is difficult. Many objects have complex shapes, and achieving a functional grasp often requires very specific grasp configurations. A direct functional grasp may not always be feasible depending on the object's position. That necessitates intricate pre-grasp manipulations, including repositioning and reorienting the object. To enable this level of versatility, human-like multi-fingered hands are frequently employed. Achieving dexterous manipulation with a high-DoF multi-finger hand, however, is non-trivial.

In this thesis, we propose methods to tackle the aforementioned challenges. We do so by employing learning-based and classical optimization-based approaches. We aim to achieve efficient motion generation for robotic manipulators. For learning-based approaches, efficiency is achieved through the use of compact models with low runtimes, short training times, and modest hardware requirements. For the optimization-based methods, efficiency is characterized by low computation times, enabling fluid, on-demand robot operation. We accomplish this by thoughtfully incorporating prior knowledge of the problems into the model architectures, the design of cost functions, and the formulation of learning pipelines.

We improve the trajectory tracking control accuracy by introducing outer-control-loop reference correction policies. First, we explore offline supervised learning for open-loop feed-forward reference correction policy. Second, we propose a methodology to learn a closed-loop policy online directly on the real robot with deep reinforcement learning (DRL). The fast generation of feasible trajectories for dual-arm systems, satisfying multiple constraints, is achieved with our optimization-based method through a multi-component cost function.

Finally, to achieve dexterous manipulation, we train a policy with DRL in highly parallelized simulation. We avoid the need for expert demonstrations by utilizing the proposed dense multi-component reward function and curriculum.

1.1 KEY CONTRIBUTIONS

The main contributions of this thesis are summarized as follows:

TRAJECTORY TRACKING WITH SUPERVISED LEARNING. We propose to learn open-loop reference correction policy in a supervised manner. We incorporate a one-step future prediction module within the model to move away from pure reactive behavior towards a more planning-oriented strategy. The policy is trained offline on a small real-world dataset. Since the ground-truth optimal reference trajectories are unknown, the policy is trained with inverted data, leveraging hindsight experience replay (HER). At the same time, the introduced one-step prediction module is trained on the original data. This allows for reusing the same dataset while learning two different modalities, improving the data efficiency of the method. The linear time-invariant (LTI)-based layers are used in the model to facilitate approximating complex dynamics.

TRAJECTORY TRACKING WITH DEEP REINFORCEMENT LEARNING. We employ DRL to train a closed-loop reference correction policy. The policy is trained online directly on the real robot. To ensure that the actions of the policy stay within safe margins, the stochastic policy is represented with a beta distribution. State, action, and reward formulations enable quick and stable convergence on a single robot, using an ordinary laptop for computations. Additionally, we propose to learn a coarse simulation from a small real-world dataset to pre-train the policy before starting learning on the real robot. This mitigates the negative effects of the initial exploration of the action space.

DUAL-ARM TRAJECTORY OPTIMIZATION. We introduce a multi-component cost function, facilitating optimization of dual-arm trajectories with respect to several costs and constraints simultaneously. To better account for collisions in the obstacle cost term, an estimation of the worst-case overlap volume is calculated. To address the closed kinematic chain constraint, we subdivide the kinematic chain into active and passive sub-chains. Given the active chain configurations, the passive sub-chain configurations are projected to satisfy the constraint. We propose an implicit redundancy resolution for the passive sub-chain through the optimization of the initial configurations for the inverse kinematics (IK) solver.

DEXTEROUS PRE-GRASP MANIPULATION WITH DEEP REINFORCEMENT LEARNING. DRL is utilized for learning a dexterous pre-grasp manipulation policy for functional grasping. We propose two distinctive target functional grasp representations and analyze their influence on the learned behaviors. First, explicit representation is defined with six-dimensional (6D) pose of the end-effector and finger positions. Second, constraint-based representation is defined with a three-dimensional (3D) index fingertip position and end-effector orientation. The constraint-based target grasp representation enables the policy to freely learn a way to grasp each object category, as opposed to achieving strictly defined grasps in the case of the explicit grasp representation. We propose compact state and action representations, as well as a dense multi-component reward function. They are agnostic of the arm, hand, and object types. That facilitates learning of intuitive dexterous behaviors from scratch, without the need for costly expert demonstrations.

1.2 PUBLICATIONS

Parts of this thesis have been published in peer-reviewed journals and conference proceedings. The most relevant publications are presented below in chronological order:

D. Pavlichenko, D. Rodriguez, M. Schwarz, C. Lenz, A. S. Periyasamy, and S. Behnke (2018). "Autonomous dual-arm manipulation of familiar objects." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. DOI: 10.1109/HUMANOIDS.2018.8624922.

D. Pavlichenko and S. Behnke (2021). "Flexible-joint manipulator trajectory tracking with learned two-stage model employing one-step future prediction." In: *IEEE International Conference on Robotic Computing (IRC)*. DOI: 10.1109/IRC52146.2021.00008.

D. Pavlichenko and S. Behnke (2022a). "Flexible-joint manipulator trajectory tracking with two-stage learned model utilizing a hardwired forward dynamics prediction." In: *International Journal of Semantic Computing (IJSC)* 16.03, pp. 403–423. DOI: 10.1142/S1793351X22430036.

D. Pavlichenko and S. Behnke (2022b). "Real-robot deep reinforcement learning: improving trajectory tracking of flexible-joint manipulator with reference correction." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2671–2677. DOI: 10.1109/ICRA46639.2022.9812023.

D. Pavlichenko and S. Behnke (2023). "Deep reinforcement learning of dexterous pre-grasp manipulation for human-like functional categorical grasping." In: *IEEE International Conference on Automation Science and Engineering (CASE)*. DOI: 10.1109/CASE56687.2023.10260385.

D. Pavlichenko and S. Behnke (2025). "Dexterous pre-grasp manipulation for human-like functional categorical grasping: Deep reinforcement learning and grasp representations." In: *IEEE Transactions on Automation Science and Engineering (T-ASE)*. DOI: 10.1109/TASE.2025.3541768.

The following publication is closely related to the topics presented in this thesis and was written during the time in which the presented research has been conducted.

D. Pavlichenko, D. Rodriguez, C. Lenz, M. Schwarz, and S. Behnke (2019). "Autonomous bimanual functional regrasping of novel object class instances." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 351–358. DOI: 10.1109/HUMANOIDS43949.2019.9035030.

1.3 OUTLINE

This thesis is composed of six chapters. The scientific contributions are presented in Chapters 2 to 5. Each of the chapters starts with an introduction of the problem and the proposed solution, followed by a review of related work. After that, the method is presented, and the conducted experiments together with their analysis are described. Finally, the discussion section summarizes the findings of a chapter. Each chapter is written to be self-contained and can be read individually. The methods are ordered by their position in the software hierarchy, from the low-level to the high-level.

Chapter 2 presents a method for improving trajectory tracking accuracy based on reference correction with supervised learning. The policy is learned offline from a small real-world dataset in a supervised fashion. A one-step future prediction is hardwired within the model

to facilitate planning behavior. The policy is serving as an open-loop feed-forward controller on top of the underlying classical controller. The method is evaluated in the real world on a 7 DoF arm of the Baxter robot and is compared to model architectures without the prediction step as well as against the vendor-provided controller.

Chapter 3 presents a DRL-based method for improving trajectory tracking accuracy. The policy is learned online directly on the real robot from scratch. The policy is serving as a closed-loop reference correction controller on top of the underlying classical controller. The stochastic actions of the policy are drawn from the beta distribution to ensure that the action magnitude stays within safe margins. In addition, a one-step future prediction model from Chapter 2 is used as a simulator to pre-train the policy. This approach offers an alternative to starting learning on the real robot with a random policy. The method is evaluated on the real 7 DoF arm of the Baxter robot and is compared to the approach presented in Chapter 2 as well as against the vendor-provided controller.

Chapter 4 presents an optimization-based method for dual-arm trajectory planning. The method leverages a multi-component cost function, enabling optimization with respect to multiple costs and constraints simultaneously. In the cost function, the obstacle cost term is improved by defining the cost through the estimation of the worst-case overlap volume. Closed kinematic chain constraint is addressed by splitting the chain into active and passive sub-chains. The configurations of the passive sub-chain are obtained by projecting the sampled active sub-chain configurations. We introduce an implicit redundancy resolution for the passive sub-chain. The method is evaluated in simulation and on the real robot and is compared to several well-established planners.

Chapter 5 presents a DRL-based method for learning policy for dexterous pre-grasp manipulation for functional grasping. The approach leverages a dense multi-component reward function that is agnostic of the arm, hand, and object types. This enables learning of dexterous manipulation behaviors without expensive expert demonstrations. A compact state representation together with highly-parallelized simulation enables quick learning on a single computer. Two possible target grasp representations are studied. The method is evaluated in simulation on previously unseen instances of objects of three categories.

Finally, Chapter 6 concludes this thesis. The scientific findings and developed approaches are summarized and discussed. The possible directions for future work are proposed.

2

TRAJECTORY TRACKING WITH REFERENCE CORRECTION: SUPERVISED LEARNING

PREFACE

This chapter is adapted from Pavlichenko and Behnke, 2021, previously published by IEEE and presented at the 5th IEEE International Conference on Robotic Computing (IRC 2021), and Pavlichenko and Behnke, 2022a, previously published by World Scientific Publishing in the International Journal of Semantic Computing (IJSC).

Statement of Personal Contribution

The author of this thesis substantially contributed to all aspects of the publication (Pavlichenko and Behnke, 2021), including the literature survey, conception, design, and implementation of the proposed method, the preparation and conduct of experiments and evaluation of the proposed approach, the analysis and interpretation of the experimental results, drafting the manuscript, as well as the revision and final editing of the version to be published.

The author of this thesis substantially contributed to the following aspects of the publication (Pavlichenko and Behnke, 2022a), including the literature survey, conception, design, and implementation of the proposed method, the preparation and conduct of experiments and evaluation of the proposed approach, the analysis and interpretation of the experimental results, drafting the manuscript, as well as the revision and final editing of the version to be published.

The content presented in this chapter, unless otherwise stated, is the contribution of the author of this thesis.

ABSTRACT

In this chapter, we present an approach for improving trajectory tracking accuracy with an open-loop feed-forward reference correction policy. It acts as an outer-loop controller on top of the underlying classical controller. We propose a two-stage model that is composed of a one-step future prediction and a reference correction module. The future prediction module is designated to move towards planning-oriented behavior. We train the model on a small real-world dataset in a supervised manner and evaluate it on the Baxter robot. We compare our method to several popular model architectures and to the vendor-provided classical controller. The conducted real-world experiments

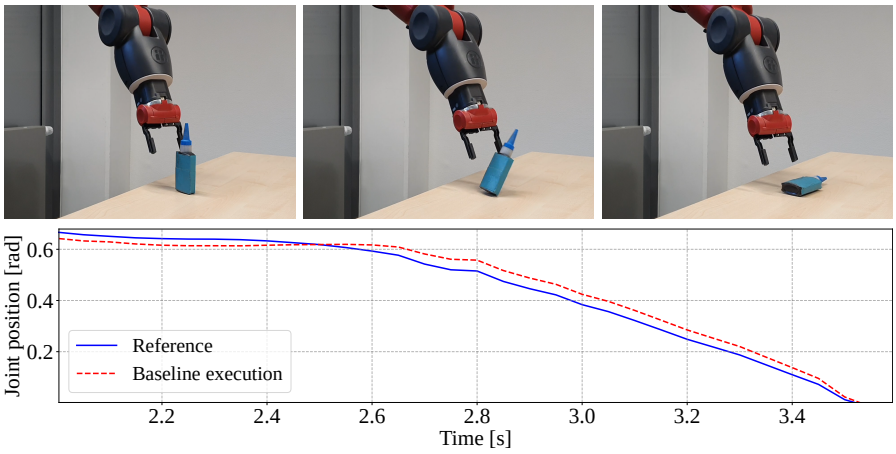


Figure 2.1: Typical challenge of using a flexible-joint manipulator to reach a pre-grasp pose. *Top:* Baxter arm tips over an object due to inaccurate reference trajectory tracking. *Bottom:* Shoulder yaw joint position vs. time during execution with the baseline vendor-provided classical controller. Large portions of the trajectory are tracked with significant errors.

indicate that the hard-wired one-step future prediction substantially improves the trajectory tracking accuracy compared to other models and to the proposed model without the prediction step.

2.1 INTRODUCTION

The ability to accurately follow a planned trajectory is a fundamental prerequisite for the majority of robotic manipulator applications. For traditional industrial manipulators, methods such as iterative learning control (ILC) (Arimoto, 1990) have proven effective in achieving this objective. ILC is predicated on the assumption that identical trajectories are repeated within a highly structured environment.

Recently, however, the application of robotic manipulators has expanded beyond these contexts: direct human-robot collaboration in shared workspaces necessitates significantly higher safety requirements. Often, these requirements are initially met at the hardware level through the use of compliant series-elastic actuators. Nevertheless, flexible manipulators tend to output less accurate motions, and their complex underlying dynamic models are frequently unknown. Designing a classical controller for such manipulators is a challenging task. Fig. 2.1 demonstrates a shortcoming of poorly tracked trajectory in practice. The simplest approach to mitigate this issue is to operate at low velocities, but this compromises the system's efficiency. In this chapter we present an approach to attain accurate trajectory execution at high velocities for inexpensive flexible-joint manipulators by complementing a classical controller with an open-loop reference correction model obtained through offline supervised learning.

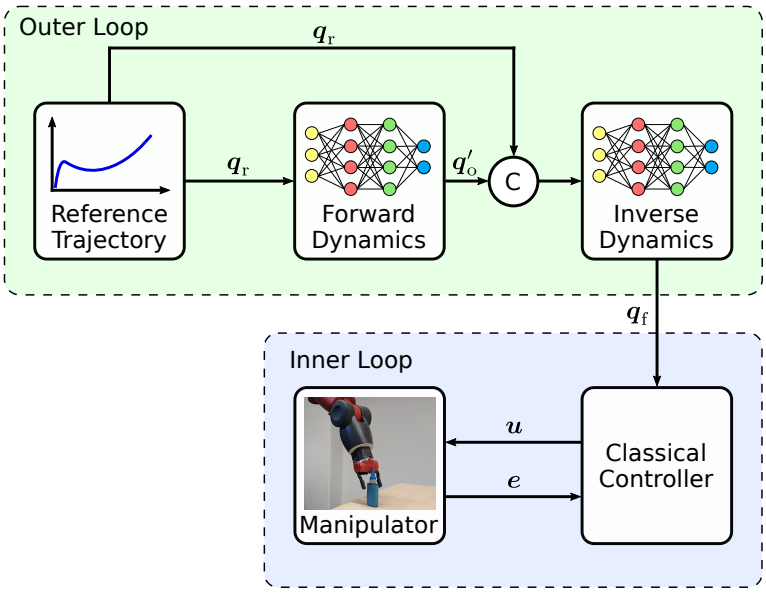


Figure 2.2: Open-loop reference correction control architecture with one-step future prediction step. Given reference trajectory q_r, the learned one-step future prediction model produces a prediction q'_o. The prediction is combined with the reference q_r, forming an input for the learned reference correction model, which outputs a modified reference q_f. The classical controller produces the control signal u for the actuators and receives feedback e. (C): concatenation.

Artificial neural networks (NNs) are known for their ability to generalize and model complex non-linear relations. We present a methodology for neural-learned feed-forward outer-loop control based on LTI dynamical operators. In particular, as a part of our model, we use the dynoNet (Forgione and Piga, 2021), which resembles the features of a recurrent neural network (RNN) (Greff et al., 2017) and one-dimensional (1D) convolution (Z. Wang, Yan, and Oates, 2017). The LTI layers are specifically designed for sequence modeling and system identification. They successfully approximate complex non-linear causal dynamics while being differentiable and suitable for backpropagation. Thus, we utilize them for learning reference correction of a flexible-joint manipulator.

We propose a two-stage model. The first part predicts the manipulator state one step into the future. The output of this model is used to augment the input to the LTI-based model, which produces the feed-forward joint position and velocity commands, representing a modified reference trajectory, which is then fed to the inner-loop classical feedback controller. The control architecture is shown in Fig. 2.2. The motivation for such model architecture is to hardwire the *"Infer what will happen in the future, then think what would be the best action now to prevent the foreseen inaccuracies"* structure within the network. We elaborate that such an architecture moves away from the pure reactive

policy towards a more intelligent planning-ahead behavior. This can be viewed as a simplistic model-predictive framework.

Such model architecture helps to maximize the extraction of information from the scarce real-world data. That is achieved by using the same data twice to learn two different modalities. First, to learn the one-step future prediction. Second, to learn the reference correction from the inverted data. The models are trained on the real-robot data in a supervised manner using plain backpropagation. We evaluate the performance of our method on the real Baxter robot against a vendor-provided baseline classical controller, a multi-layer perceptron (MLP) and RNN. Our approach significantly improves trajectory tracking accuracy compared to the baseline controller and outperforms other models. The method allows executing fast trajectories with higher accuracy.

In summary, our main contributions are:

- Two-stage model architecture, explicitly utilizing one-step future prediction to maximize knowledge extraction from a small dataset, trained with backpropagation,
- hardwired one-step future prediction step, reinforcing planning behavior within the network, and,
- investigation of the effectiveness of LTI-based models for robotic manipulator reference correction.

2.2 RELATED WORK

Trajectory tracking has been extensively researched for decades. A notable classical method is ILC (Arimoto, 1990) which incrementally refines the control input using tracking errors. After a few iterations, it achieves nearly perfect tracking of a reference trajectory. However, a key drawback of ILC is its lack of transferability, as the optimization must be repeated from the beginning for each new trajectory. Alternatively, differential dynamic programming (DDP) (Mayne, 1966) employs a linear quadratic regulator (LQR) to iteratively refine the control inputs. This method is computationally intensive, making it impractical for real-time applications.

Trajectory tracking is often addressed by means of NNs (Q. Guo et al., 2019; He et al., 2018; Jin et al., 2018; Yiming et al., 2017). Such increased attention is due to their comprehensive ability to generalize and model complex nonlinear dynamics. Radial basis function (RBF) NNs are a popular model choice (Q. Chen et al., 2016; H. Han, X. Wu, et al., 2019; H. Han, L. Zhang, et al., 2016; Qiao, Meng, and W. Li, 2018; F. Wang et al., 2017; C. Yang et al., 2016). Xia, L. Wang, and Chai, 2014 use RBF-NN to mitigate the effects of friction in swing-up control of a two-joint manipulator. However, such models contain a large number

of parameters, and it is challenging to tune the hyperparameters, such as the number of Gaussian kernels, their centers, and their shapes.

A wavelet fuzzy NN is proposed for predictive control by Lu, 2011. The model is based on a set of fuzzy rules. Each rule is linked to the wavelet function from the consequent rules. The model is trained with backpropagation. The main disadvantage of this architecture lies in its complexity and high computational cost. Nevertheless, it is successfully applied for control of nonlinear systems (F.-J. Lin, S.-G. Chen, and I.-F. Sun, 2017; Sheng, Xiaojie, and Lanyong, 2017; Zhao, C.-M. Lin, and Chao, 2019). Rueckert et al., 2017 use Gaussian process regression (GPR) to perform kinematic control of a surgical cable-driven manipulator. Saveriano et al., 2017 model the residual dynamics using a Gaussian process (GP)-based model together with reinforcement learning. The main drawback of GP-based models, in comparison to NNs, is that they grow together with the data, thus requiring a lot of resources for evaluation and execution.

Mahler et al., 2014 demonstrate that long short-term memory (LSTM) NNs can outperform the GP-based method for modeling inverse dynamics. While many works focus on the model architecture, Morse et al., 2020 apply meta-learning to obtain state-dependent loss functions, which demonstrates another viable approach. M. Wang, Ye, and Z. Chen, 2017 propose a method for neural learning from adaptive neural control (ANC). The authors introduce an adaptive neural dynamic surface control scheme, which reduces the dimension of neural inputs and the number of neural approximators. The ANC scheme is shown to be capable of storing knowledge of unknown system dynamics through experiments on a single-link robot in simulation and on the real Baxter robot.

Much research is built around ILC (K. Patan, M. Patan, and Kowalów, 2017; Y. Yang, D. Huang, and X. Dong, 2019; G. Zhang et al., 2021; Zuo and Cai, 2010). For instance, in work by Schwarz and Behnke, 2014 the compliant position control is achieved through learning the parameters of direct current motors and friction models with the help of ILC. The method is tested in the real world with humanoid robot gait. This approach avoids performing a separate run for each parameter but instead identifies all parameters at once. A combination of $\mathcal{L}_1$ adaptive control and ILC is used for transfer learning between systems with different dynamics by Pereida et al., 2018. An extended $\mathcal{L}_1$ controller runs in the inner closed-loop control level and achieves robust and repeatable behavior. ILC is used as an outer-loop control, where the transfer of learned experience is realized. The system is tested on two quadrotors with different dynamics and outperforms systems composed of ILC and proportional-derivative (PD) or proportional-integral-derivative (PID) controllers.

Another use of ILC involves the production of the ground truth input trajectories to train NNs that approximate the inverse dynamics

of the system by D. Chen et al., 2021. The authors apply this approach to an industrial manipulator, training the model in simulation. The model is then applied to the real robot with transfer learning. A separate NN is trained for each joint using backpropagation. The approach demonstrates a significant improvement of the trajectory tracking accuracy both in simulation and in the real world. However, this method cannot be applied to systems that do not have good dynamics approximation for simulation, which is frequently the case for compliant systems. In addition, the assumption of decoupled joints does not hold when dealing with flexible joints.

A relevant area of research lies in the area of motion planning. For instance, Qureshi, Bency, and Yip, 2019 and Qureshi, Miao, et al., 2021 introduce computationally efficient motion planning network (MPN). The architecture consists of an encoder and planning network, which bidirectionally generate connectable paths. Moreover, the model can be merged with classical planning algorithms to provide the worst-case guarantees.

L. Li et al., 2021 use imitation learning to train a scalable model-predictive motion planning networks framework. The method quickly finds near-optimal paths with worst-case guarantees. NN-based approaches are applied to motion planning in various ways (Ha, J. Xu, and Song, 2020; Johnson et al., 2020; Qureshi, J. Dong, et al., 2020; Qureshi and Yip, 2018). In this work we address the problem of trajectory tracking, which does not require an extensive search in high-dimensional spaces. Nevertheless, we encourage a simplistic planning behavior within the proposed model by hard-wiring the future prediction step, which is related to the works mentioned above. Moreover, the proposed two-stage architecture has certain similarities with an encoder-decoder structure. We discuss it in more detail in Section 2.7.

S. Chen and Wen, 2019 present two approaches for performing feed-forward trajectory tracking with series-elastic actuators. The first approach uses RNN. It approximates forward dynamics in combination with ILC. It in turn utilizes this model to find an optimal command sequence. The main drawback of this approach is the significant runtime required for ILC to converge, which makes this method impractical to be applied online. The second approach utilizes bidirectional recurrent neural network (BRNN) (Schuster and Paliwal, 1997) to approximate the inverse dynamics directly, such as by Talebi, Patel, and Khorasani, 1998 and Q. Li et al., 2017. This allows directly obtaining the required control inputs in a short time. Both approaches are trained with backpropagation on three hours of sinusoidal and random trajectories recorded on the real Baxter robot. Resulting models improve the trajectory tracking over the baseline PD controller.

Callar and Böttger, 2022 present an approach for learning inverse dynamics models in robotic systems, particularly focusing on locally isotropic robot motion. The authors propose a hybrid learning frame-

work that combines data-driven learning with model-based methods. This allows for more accurate inverse dynamics modeling by leveraging both physical insights from traditional models and the flexibility of machine learning techniques. In particular, LSTM and Transformer network topologies. The method is evaluated on the KUKA iiwa 14 manipulator.

W. Huang et al., 2024 propose a LSTM-based model that is enhanced with velocity-awareness, allowing the system to capture the temporal dependencies in the manipulator's motion and improve predictions by explicitly incorporating velocity information into the learning process. In contrast, we propose a model that utilizes causal LTI dynamical operators. In addition, we also encapsulate a pre-trained model for one-step future prediction to augment the input to the reference correction model. This two-stage approach, inspired by Zeng et al., 2020, allows the model to take the prediction of the future into account. This increases the effectiveness of the produced feed-forward command that helps minimize future inaccuracies. The proposed model uses a more abstract input representation. It considers two timesteps of joint positions, velocities, and accelerations, instead of multiple timesteps of joint positions.

In combination, this allows the model to learn and generalize effectively from a smaller amount of real-world data. In addition, our model produces not only feed-forward joint position commands but also velocities, which allows us to further improve the trajectory tracking accuracy. Finally, we train the model on a smaller dataset of functional trajectories, imitating learning from a regular operation, as opposed to learning from trajectories of specific artificial shapes, such as sine waves.

2.3 BACKGROUND

In this chapter we use LTI dynamical operators as building blocks for the proposed model. They are introduced as a part of the dynoNet architecture (Forgione and Piga, 2021) and can be trained end-to-end by backpropagation. In this section we briefly review the concept behind such building blocks.

LTI layers are parametrized using rational transfer functions, enabling them to perform infinite impulse response (IIR) filtering on their input sequences. In the dynoNet architecture, the input-output relationship for an individual single-input single-output (SISO) LTI layer is defined by the dynamical rational operator $G(s)$, given the input signal $u(t) \in \mathbb{R}$ and output $y(t) \in \mathbb{R}$ at time t:

$$y(t) = G(s)u(t) = \frac{B(s)}{A(s)}u(t), \tag{2.1}$$

where $A(s)$ and $B(s)$ are polynomials in the time delay operator $s^{-1} : s^{-1}u(t) = u(t-1)$:

$$A(s) = 1 + a_1 s^{-1} + \ldots + a_{n_a} s^{-n_a}, \tag{2.2a}$$

$$B(s) = b_0 + b_1 s^{-1} + \ldots + b_{n_b} s^{-n_b}. \tag{2.2b}$$

The filtering operation through $G(s)$ in Eq. 2.1 is equivalent to the input/output equation:

$$A(s)y(t) = B(s)u(t), \tag{2.3}$$

which is equivalent to the recurrence equation:

$$\begin{aligned} y(t) = b_0 u(t) + b_1 u(t-1) + \ldots + b_{n_b} u(t-n_b) \\ - a_1 y(t-1) \ldots - a_{n_a} y(t-n_a). \end{aligned} \tag{2.4}$$

The coefficients of the polynomials $A(s)$ and $B(s)$ are the configurable parameters of $G(s)$. For simplicity, these coefficients are assembled in vectors $\boldsymbol{a} = [a_1, a_2, \ldots, a_{n_a}] \in \mathbb{R}^{n_a}$ and $\boldsymbol{b} = [b_0, b_1, \ldots, b_{n_b}] \in \mathbb{R}^{n_b+1}$.

To integrate the linear dynamical operator into a deep learning framework, it is essential to define both the forward and backward operations. During the forward pass, an input sequence $\boldsymbol{u} \in \mathbb{R}^T$ is filtered through a dynamical system $G(s)$ characterized by the structure specified in Eq. 2.1 and parameters $\boldsymbol{a}$ and $\boldsymbol{b}$. The resulting block output is a vector $\boldsymbol{y} \in \mathbb{R}^T$ that contains the filtered sequence:

$$\boldsymbol{y} = G.\text{forward}(\boldsymbol{u}, \boldsymbol{b}, \boldsymbol{a}) = G(s)\boldsymbol{u}. \tag{2.5}$$

In the backward pass, G receives the vector $\boldsymbol{y} \in \mathbb{R}^T$ containing the partial derivatives of the loss $\mathcal{L}$ w.r.t. $\boldsymbol{y}$, specifically:

$$\overline{\boldsymbol{y}}_t = \frac{\delta \mathcal{L}}{\delta \boldsymbol{y}_t}, t = 0, \ldots, T-1. \tag{2.6}$$

Given $\overline{\boldsymbol{y}}$, it is necessary to compute the derivatives of the loss $\mathcal{L}$ with respect to its differentiable inputs $\boldsymbol{b}$, $\boldsymbol{a}$, and $\boldsymbol{u}$. The structure of the backward operation is as follows:

$$\overline{\boldsymbol{b}}, \overline{\boldsymbol{a}}, \overline{\boldsymbol{u}} = G.\text{backward}(\boldsymbol{b}, \boldsymbol{a}, \boldsymbol{u}, \overline{\boldsymbol{y}}). \tag{2.7}$$

The detailed derivation of each respective partial derivative is available in the original dynoNet paper (Forgione and Piga, 2021). With the forward and backward operations defined, integrating the linear dynamical operator into a NN alongside classical building blocks for inference and training becomes straightforward.

2.4 TWO-STAGE MODEL WITH ONE-STEP FUTURE PREDICTION

Given a reference trajectory $\boldsymbol{q}_\mathrm{r}$, our objective is to produce a modified reference trajectory $\boldsymbol{q}_\mathrm{f}$ that would lead the manipulator to follow $\boldsymbol{q}_\mathrm{r}$ more accurately using an underlying classical feedback controller. Each trajectory $\boldsymbol{q}$ consists of N equally spaced in time keyframes $\boldsymbol{q}(t_i) : \boldsymbol{q} = [\boldsymbol{q}(t_1) \dots \boldsymbol{q}(t_N)]$. Each keyframe $\boldsymbol{q}(t_i) \in \mathbb{R}^M$ represents a manipulator configuration in joint space with M joints: $\boldsymbol{q}(t_i) = [q(t_1, 1) \dots q(t_N, M)]$.

We propose a two-stage model that consists of two main parts. The first part is the forward-inference network (FIN): a MLP that produces one-step future prediction. Provided with two consequent reference trajectory points, it outputs a predicted state of the manipulator at the following timestep. The second part of the model takes the original input, augmented with the FIN prediction, to produce the corrected reference.

The second part of the model is based on LTI dynamical operators, as implemented in the dynoNet[1]. LTI operators were shown to be efficient when learning the complex causal non-linear dynamics while being suitable for an end-to-end backpropagation (Forgione and Piga, 2021). These properties are advantageous for learning the dynamics of the flexible-joint manipulator. That is why we choose LTI-based blocks to be the core of our model. The model resembles a multiple-input multiple-output (MIMO) Wiener-Hammerstein structure, according to the block-oriented modeling framework (Giri and Bai, 2010). Thus, it is referred to as the dynoNet Wiener-Hammerstein (DWH) model. Finally, we refer to the whole model as FIN-DWH. The two-stage architecture with one-step future prediction aims to push the model from purely reactive policy behavior towards more intelligent planning ahead. That allows achieving a higher accuracy of the trajectory tracking.

In this chapter we apply the proposed method to the arm of the Baxter robot. Thus, below we describe the approach within the context of that robot. However, it is straightforward to apply the method to an arbitrary robot manipulator. The Baxter arm has $M = 7$ joints, thus $\boldsymbol{q}(t) \in \mathbb{R}^7$. Since the joints of Baxter are coupled (S. Chen and Wen, 2019), a common approach of training a separate model for approximating the dynamics of each joint is not feasible. Instead, we approximate the underlying dynamics by considering all joints simultaneously. We represent each point of the manipulator trajectory as a tuple $\langle \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t) \rangle$. Explicit inclusion of velocity and acceleration provides information about the dynamics to the NN directly, as op-

1 https://github.com/forgi86/dynonet

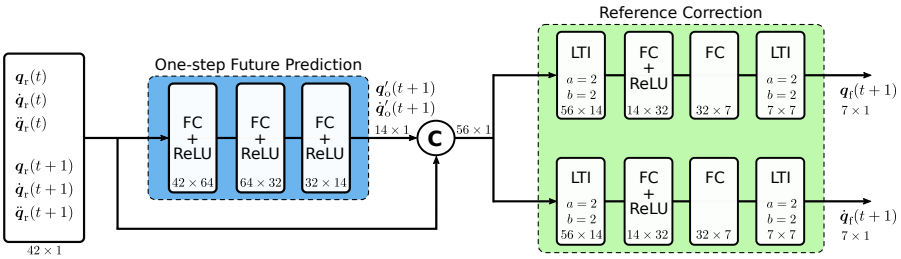


Figure 2.3: Two-stage reference correction model. Input is the current reference state of the manipulator, expressed with joint positions $\boldsymbol{q}_r(t)$, velocities $\dot{\boldsymbol{q}}_r(t)$, accelerations $\ddot{\boldsymbol{q}}_r(t)$, and the reference state at the next keyframe at $t+1$. Blue: FIN, one-step future prediction block. Green: DWH model, that takes the same input, augmented with the future state estimation from FIN. It outputs the feed-forward joint positions and velocities, forming a corrected reference. (c): concatenation. FC: fully-connected layer. LTI: linear time-invariant dynamical operator-based block.

posed to forcing the NN to infer it from the series of joint positions. The input to the FIN-DWH model is a 42-element vector:

$$[\boldsymbol{q}_r(t), \dot{\boldsymbol{q}}_r(t), \ddot{\boldsymbol{q}}_r(t), \boldsymbol{q}_r(t+1), \dot{\boldsymbol{q}}_r(t+1), \ddot{\boldsymbol{q}}_r(t+1)], \tag{2.8}$$

where $\boldsymbol{q}_r(t)$ is the reference keypoint of the manipulator at time t and $\boldsymbol{q}_r(t+1)$ is the next reference keypoint, correspondingly. The output of the network is then a 14-element vector $[\boldsymbol{q}_f(t+1), \dot{\boldsymbol{q}}_f(t+1)]$ where $\boldsymbol{q}_f(t+1)$ is the corrected reference point which should lead the manipulator to the state $\boldsymbol{q}_r(t+1)$, after being supplied to the underlying classical controller. The diagram of the model is shown in Fig. 2.3.

The proposed method is open-loop and does not include live feedback from the robot. This imposes an assumption that the consequent execution of the corrective feed-forward commands should result in the manipulator following the reference trajectory perfectly. This assumption allows training the model offline using the collected data as is, without the need to introduce an additional dynamics model to produce the ground-truth control inputs. This significantly simplifies the data collection process, making the integration and execution of the approach more transparent.

We analyze the practical shortcomings of the aforementioned assumption by performing the experiments with a previously unseen payload. Since the complete feed-forward command sequence from our model is available before it is executed, we apply a zero-phase Savitzky-Golay filter to each individual joint position and joint velocity trajectory with window 21 and polynomial order 2 to alleviate any potential non-smooth fragments in the control signal.

The best-performing Baxter vendor-provided controller is an Inverse Dynamics Feed Forward Position Controller. We use it as a baseline in this work. This controller calculates the necessary torque

from the supplied positions, velocities, and accelerations using the internal dynamics model. Thus, we train our model to produce velocities $\dot{\boldsymbol{q}}_{\mathrm{f}}(t+1)$ as well. We do not train the model to output joint accelerations $\ddot{\boldsymbol{q}}_{\mathrm{f}}(t+1)$ because they can not be measured by the robot hardware.

To give an intuition about how the position vendor-provided controller and the inverse dynamics feed-forward vendor-provided controller compare, the average cumulative joint position tracking error per point is 0.157 ± 0.082 rad in the first case against 0.069 ± 0.032 rad (mean $\pm$ standard deviation) in the second case. The latter is more than two times accurate. Thus, we train the model to provide feed-forward velocity input and compare it against this more accurate vendor-provided controller. Note that it is straightforward to use only the position or velocity vector from the output in case when position or velocity control is used. The existing error in trajectory tracking accuracy shows that the internal dynamics model does not represent the complex, coupled-joints real-robot dynamics accurately enough. The proposed data-driven method does not replace the classical baseline controller but forms an outer-loop, complementing the existing dynamics model and learning to compensate for the observed inaccuracies.

By explicitly including velocity and acceleration in the input, we provide enriched information about the manipulator state without forcing the network to infer derivatives from the time series of joint positions. In addition, since a tuple of $\langle \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}} \rangle$ contains certain information about the dynamic state of the manipulator, we can significantly reduce the number of timesteps needed as an input.

In this work, we use only two timesteps, as described above. Moreover, velocity and acceleration represent certain patterns in robot dynamics in a more general way, as opposed to sequences of joint positions alone. The trajectories used for training contain points separated by $\Delta t = \frac{1}{F}\,\mathrm{s}$ where $F = 20\,\mathrm{Hz}$. In case of any encoder inaccuracies of consistent magnitude, a larger time span between sample points allows decreasing their influence. In addition, larger Δt also reduces the influence of latency. The inner-loop feedback controller of the Baxter joints operates at a much higher frequency.

To perform a one-step future prediction of the manipulator, we define the FIN model: a MLP with three fully connected layers. It takes a 42-element vector as an input and produces a 14-element vector $[\boldsymbol{q}'_{\mathrm{o}}(t+1), \dot{\boldsymbol{q}}'_{\mathrm{o}}(t+1)]$, where $\boldsymbol{q}'_{\mathrm{o}}(t+1)$ is a predicted state of the manipulator after executing the command $\boldsymbol{q}_{\mathrm{r}}(t+1)$ as it is. We use the rectified linear unit (ReLU) activation function as non-linearity. FIN model has 5,294 weights in total. The following DWH model performs the reference correction and consists of two identical independent branches.

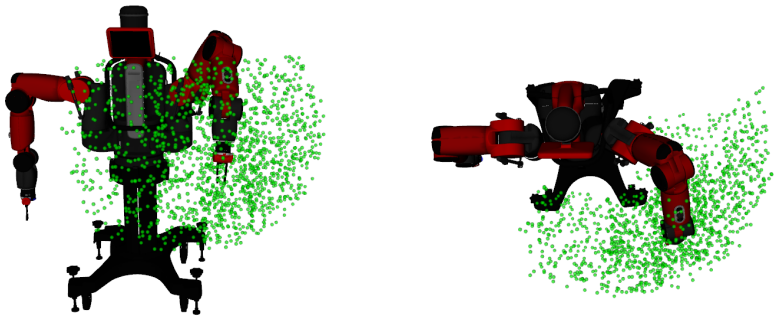


Figure 2.4: Region of workspace for data collection for the left arm of the Baxter robot. Keypoints of the sampled trajectories are shown as green points, representing the corresponding end-effector positions. We focus on the most used portion of the workspace in the front and to the side of the robot.

Each branch takes in the original 42-element input concatenated with the 14-element output of the FIN model, resulting in a 56-element input. Each branch then outputs a 7-element vector. One branch produces positions, the other – velocities. The architecture of a branch is as follows. First, the LTI block with $a = 2$ and $b = 2$, which outputs 14 features, motivated to represent a rough position + velocity approximation. a and b define the polynomial order for the denominator and nominator of a rational transfer function (Forgione and Piga, 2021). The LTI block is followed by two fully connected layers. The result of these layers is then fed to the last LTI block, which also has $a = 2$ and $b = 2$ and produces the final 7-element vector.

The architecture of the model is shown in Fig. 2.3. Each branch has 4,043 parameters, which results in a total of $4{,}043 \times 2 = 8{,}086$ parameters for the DWH model. The LTI layers are parameterized in terms of rational transfer functions and thus apply IIR filtering to the input. Stacking this hardwired linear structure in multiple layers together with fully connected layers was shown to approximate the complex non-linear dynamics (Forgione and Piga, 2021). By providing the one-step future prediction obtained from the FIN, we allow the model to take into account the predicted future in which we would execute the next command as it is. This can be interpreted as a simplistic version of model-predictive control with only one step of looking ahead. As we show in our evaluation, this additional input improves the performance of the model.

2.5 MODEL TRAINING

In order to learn the reference correction model, we generate 45 minutes of *functional* reference trajectories $\boldsymbol{q}_r$ for the left arm of the Baxter robot. Random sinusoidal trajectories are often used to form the base of the training set (S. Chen and Wen, 2019, 2021). However, recording

such a dataset takes the valuable robot hours away from the user. Thus, we imitate learning from the data that is collected while performing actual tasks. That is why we refer to such trajectories as *functional*. This approach is advantageous as it facilitates data collection seamlessly while the robot executes relevant tasks. Subsequently, the learned model reduces execution time by achieving more accurate reference trajectory tracking at higher speeds, extending the overall capabilities of the flexible-joint manipulator.

The training set consists of pick-and-place trajectories with equal portions executed with different joint speeds: 0.6, 0.8, and 1.0 rad/s maximum speeds, respectively. The sampled trajectories cover the major part of the workspace in front and to the side of the robot with approximate dimensions of $1.4 \times 0.7 \times 1.0$ m, as shown in Fig. 2.4. When executing a reference trajectory $\boldsymbol{q}_\mathrm{r}$, the actual observed trajectory $\boldsymbol{q}_\mathrm{o}$ and $\dot{\boldsymbol{q}}_\mathrm{o}$ are recorded. Since Baxter has no way to measure the acceleration $\ddot{\boldsymbol{q}}_\mathrm{o}$, we approximate it by a cubic spline interpolation. Approximately 60% of the trajectories contain 1-2 additional waypoints between the start and the goal. All waypoints are sampled from a uniform distribution and are constrained to be at least 15 cm apart from each other to prevent executing extremely short motions. This increases the variety of the movements and simulates maneuvers such as avoiding an obstacle.

Given the set of reference trajectories $\boldsymbol{q}_\mathrm{r}$, $\dot{\boldsymbol{q}}_\mathrm{r}$ and $\ddot{\boldsymbol{q}}_\mathrm{r}$, as well as the set of the observed robot responses $\boldsymbol{q}_\mathrm{o}$, $\dot{\boldsymbol{q}}_\mathrm{o}$ and $\ddot{\boldsymbol{q}}_\mathrm{o}$, we train the two-stage model in two steps.

First, we train the FIN model. Since we have the reference and observed trajectories, the composition of the training input-output tuples is straightforward. We train the network using stochastic gradient descent (SGD) with a minibatch of 32 data points in a fully supervised manner. We use the Adam optimizer with a learning rate of 10^{-4} to minimize the mean square error (MSE) loss and employ L^2 regularization.

The training of the full FIN-DWH model is not as straightforward, because the ground-truth modified reference trajectories $\boldsymbol{q}_\mathrm{f}$ that would lead to $\boldsymbol{q}_\mathrm{o}$ following $\boldsymbol{q}_\mathrm{r}$ perfectly are unknown. It would be possible to employ ILC to obtain them, but since we do not have a good model for simulation, this would have to be done on the real robot, exponentially increasing the number of robot-hours needed to produce such a dataset. Instead, we use the same data as for FIN training and apply the HER technique (M. Andrychowicz et al., 2017). This method can be described in short as pretending that what we achieved was what we actually wanted. It is commonly used in reinforcement learning (RL) to mitigate the negative effects of sparse delayed rewards. We use $\boldsymbol{q}_\mathrm{o}$ as the goal reference trajectory, and then $\boldsymbol{q}_\mathrm{r}$ becomes the corresponding ground-truth input.

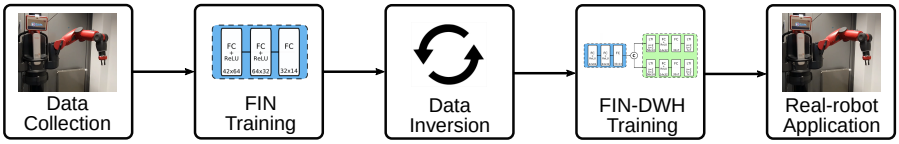


Figure 2.5: Two-stage training process. First, the FIN model, which performs a one-step future prediction, is learned. Second, data is inverted, and the complete FIN-DWH model is trained to perform reference correction. Reusing the same data to learn two different modalities improves data-efficiency.

After inverting the training examples, we perform the same training procedure as above: 32 data points per minibatch, Adam optimizer with a learning rate of 10^{-4} with the MSE loss and L^2 regularization. Note that we keep the weights of the FIN frozen during the whole training of the FIN-DWH model. This ensures that the DWH model is supplied with one-step future prediction. We have observed that allowing the network to update the FIN weights during the training leads to inferior performance. This supports our idea that when training on a limited dataset, careful hard-wiring of NN structure is of high importance. To demonstrate that in our experiments, we also evaluate the performance of the FIN-DWH version, where FIN weights are updated freely. The diagram of the training process is shown in Fig. 2.5.

Although we train the model directly with trajectories with different velocity profiles, it is also possible to train the model by gradually increasing the difficulty. Such a training curriculum would fit transparently into the real-world application scenario when allocating robot hours solely for the purpose of training is infeasible and learning from the real tasks is required instead. In this setting, arm movements are initially executed at lower velocities to minimize trajectory tracking inaccuracies and ensure successful task completion. Then, as the model improves the tracking accuracy, the maximum velocity of the arm is gradually increased.

2.6 EVALUATION

To evaluate the presented approach, we conduct experiments on the real Baxter robot. We compare the performance to the baseline vendor-provided Baxter controller: Inverse Dynamics Feed Forward Position Control. In addition, we also compare our method against the three other models. The first model is a MLP, consisting of three fully connected layers: $42 \times 64 \rightarrow 64 \times 32 \rightarrow 32 \times 14$. It has 5,294 parameters. Note that we also experimented with a larger MLP with four layers and 16,302 parameters. However, it did not demonstrate a superior performance. The second model is a three-layer RNN, which has 7,772 parameters and is referred to as RNN. It has the structure analogous to

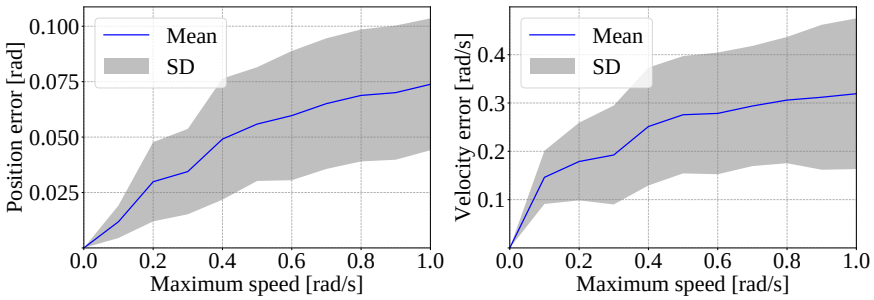


Figure 2.6: Tracking errors vs. maximum joint speed in a trajectory. *Left:* Joint position tracking error. *Right:* Joint velocity tracking error. The errors are averaged per trajectory point. The trajectories are executed with a vendor-provided controller.

MLP: $(42 + h) \times 64 \rightarrow 64 \times 32 \rightarrow 32 \times 14$, with two additional layers to produce a 14-element hidden state h: $(42 + h) \times 32 \rightarrow 32 \times h$. The third model is the DWH model with 6,518 parameters. It has the same structure as the second part of the network described in Section 2.4, omitting the FIN model. In addition, we perform an ablation study of the proposed model, introducing two model variants: FIN_u-DWH and FIN_f-DWH. In FIN_u-DWH, we let the weights of FIN model be updated during the second stage of training. In FIN_f-DWH, we keep the weights of FIN model fixed during the second stage of the training instead. Such comparison allows investigating the effectiveness of the one-step future prediction, since in FIN_u-DWH the network is given the freedom to update the weights of the FIN model.

All models take 42-element vectors as input and produce a 14-element vector of feed-forward joint positions and velocities. All models use ReLU non-linearity and are trained on the same dataset until convergence, minimizing MSE loss with the Adam optimizer. The models are trained on a regular laptop, with each training procedure taking less than 2 hours. We do our best to find the best set of hyperparameters for each model using the grid search. For the MLP, they are a learning rate of 2.0×10^{-4} and a minibatch size of 24 data points. For the RNN model, they are a learning rate of 1.5×10^{-4} and a minibatch size of 48 data points. For the DWH and both versions of the FIN-DWH, we use the same hyperparameters: a learning rate of 10^{-4} and a minibatch size of 32 data points, as the core of these models is the same. This allows us to better observe how the proposed two-stage architecture influences the performance of the model compared to the same model without the one-step future prediction.

2.6.1 *Quantitative Evaluation*

To quantitatively evaluate the proposed approach, we generate 100 unseen trajectories for the left arm of Baxter. Although our model can be used for trajectories with arbitrary speed profiles, we find it

especially interesting to evaluate the methods on trajectories with high speeds. In that case the compliance of the manipulator causes the most inaccuracies due to complex inertia effects on flexible joints, as shown in Fig. 2.6. Data shown in these plots was collected by executing random trajectories with the baseline controller. Clearly, the higher the maximum allowed speed is, the less accurate the movements of the manipulator are. The error grows slower with maximum speed increase because not all joints are able to reach this speed during the trajectory execution. Thus, in our experiments, the maximum joint speed was set to 1.0 rad/s.

We measure the average cumulative error per point in a trajectory for joint position, joint velocity, and end-effector position. We also measure the extra time needed to converge to the final point in a trajectory, as well as the runtime of each model. Given an observed trajectory $\boldsymbol{q}_\mathrm{o}$ and a reference trajectory $\boldsymbol{q}_\mathrm{r}$ with N keyframes and M joints, we calculate the average cumulative joint position error e_q per point as follows:

$$e_q = \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}|q_\mathrm{r}(t_n, m) - q_\mathrm{o}(t_n, m)|. \tag{2.9}$$

The procedure is analogous for the average joint velocity error:

$$e_{\dot{q}} = \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}|\dot{q}_\mathrm{r}(t_n, m) - \dot{q}_\mathrm{o}(t_n, m)|. \tag{2.10}$$

Finally, the average end-effector position error is computed given a forward kinematics (FK) function $\mathcal{F}$:

$$e_\mathrm{eef} = \frac{1}{N}\sum_{n=1}^{N}||\mathcal{F}(\boldsymbol{q}_\mathrm{r}(t_n)) - \mathcal{F}(\boldsymbol{q}_\mathrm{o}(t_n))||_2. \tag{2.11}$$

In Table 2.1, the average cumulative joint position and velocity errors are shown. In all tables we provide 95% confidence intervals of the mean. One can see that all the models make an improvement over the sole baseline controller. However, MLP clearly shows the worst performance. This is the case because fully-connected layers do not have an underlying structure to capture the unknown dynamics. RNN and DWH models show similar performance, although DWH has slightly better results. Finally, both versions of FIN-DWH improve over the plain DWH, demonstrating that the one-step prediction of the future allows producing a more accurate control input. Notably, $\mathrm{FIN_f}$-DWH outperforms $\mathrm{FIN_u}$-DWH. Our intuition is that it is very challenging for the internal FIN model to implicitly learn one-step future prediction during an end-to-end training procedure on a limited dataset. On the other hand, having explicit targets during the first step of the training allows to learn the desired behavior. Consequently,

Table 2.1: Comparison of average cumulative joint position and velocity errors per point.

Method	Joint position error [rad $\times 10^{-2}$]	Joint velocity error [rad/s $\times 10^{-2}$]
Baseline	6.90±0.62 (—)	32.55±2.83 (—)
MLP	4.68±0.61 (32%)	24.32±2.13 (25%)
RNN	4.20±0.42 (39%)	22.71±1.74 (30%)
DWH	4.02±0.37 (42%)	21.84±1.68 (33%)
FIN_u-DWH	3.83±0.36 (44%)	21.68±1.64 (33%)
FIN_f-DWH	**3.64±0.33 (47%)**	**21.26±1.51 (35%)**

95% confidence interval is provided after "±". Improvement over the baseline is in brackets.

this results in a superior performance of the network with fixed FIN weights, preserving the learned one-step future prediction. This signifies the importance of hard-wiring a structured framework within the network architecture to facilitate planning behavior, particularly when learning from the limited real-world data.

In Table 2.2, the average end-effector position error per point is shown. We also report the extra time to converge to the final point of the trajectory. The extra time is defined as a difference between the actual trajectory execution time and the desired execution time. This difference arises when the state of the manipulator in the final trajectory point differs significantly from the desired state and extra time is needed to reach it. A similar tendency in the performance of the models can be observed.

On average, the baseline controller has a 3 cm deviation from the desired path, while our method achieves an improvement of three times, reducing it to 1 cm on average. The extra time to arrive at the destination point is reduced by a significant 92%. That happens because

Table 2.2: Comparison of average end-effector position error per point and average extra time to reach the endpoint.

Method	EEF position error [cm]	Extra time [s]
Baseline	2.98±0.35 (—)	0.75±0.091 (—)
MLP	1.40±0.24 (53%)	0.28±0.029 (62%)
RNN	1.15±0.14 (61%)	0.12±0.013 (84%)
DWH	1.11±0.13 (62%)	0.09±0.010 (88%)
FIN_u-DWH	1.08±0.13 (63%)	0.09±0.010 (88%)
FIN_f-DWH	**1.01±0.12 (66%)**	**0.06±0.008 (92%)**

95% confidence interval is provided after "±". Improvement over the baseline is in brackets.

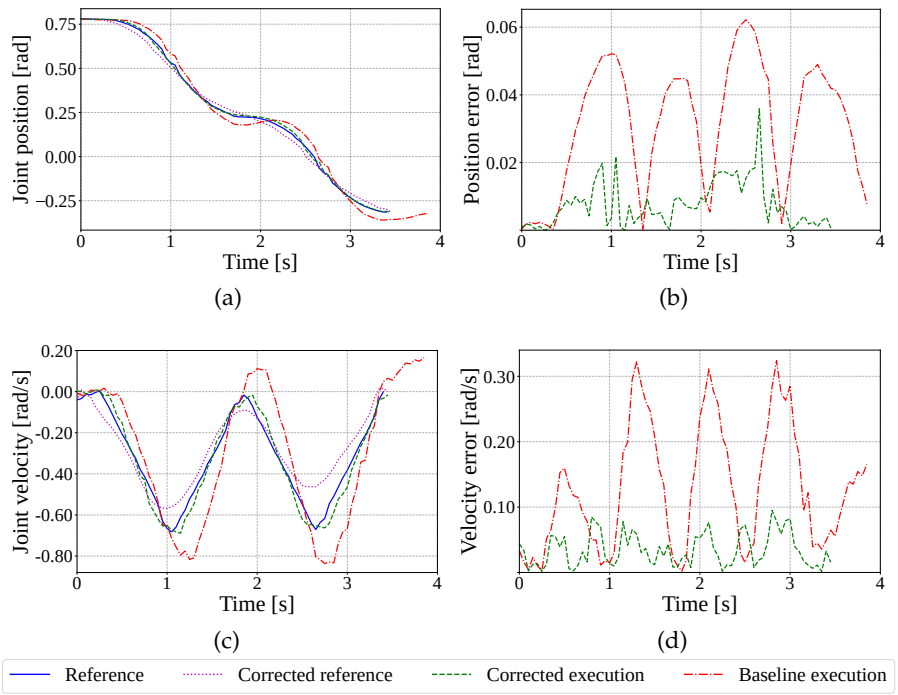


Figure 2.7: Shoulder yaw joint trajectory. (a) Position vs. time. (b) Position error vs. time. (c) Velocity vs. time. (d) Velocity error vs. time. The baseline controller alone leads to several major deviations, while our method compensates for the inertia affecting the elastic joints and follows the trajectory with higher accuracy.

the motion is much smoother overall, making the immediate, accurate arrival at the final point possible. The average runtimes per trajectory (usually consisting of 60-90 timesteps) are as follows. MLP: 0.031 s, RNN: 0.062 s, DWH: 0.067 s, FIN-DWH: 0.085 s. All computations are executed on an Intel i7-6700HQ 2.6 GHz central processing unit (CPU). Further improvements of the runtimes are possible. By listing them here, we give an intuition about the relative computational load of the compared models.

In Fig. 2.7, we show an example test trajectory of the shoulder yaw. One can see that the trajectory achieved solely by the baseline controller deviates from the target much more than the trajectory obtained with FIN-DWH. There are four big peaks of the position error and three big peaks of the velocity error. It can be noticed that the minima of the position error correspond to the maxima of the velocity error and vice versa. This can be explained by the controller trying to compensate for the position error by increasing velocity to "catch up". However, this consequently leads to the overshooting of the velocity. The whole cycle is then repeated. We attribute this effect to the compliance of the manipulator. It can be seen that our model learns to compensate for this effect, modifying the joint commands

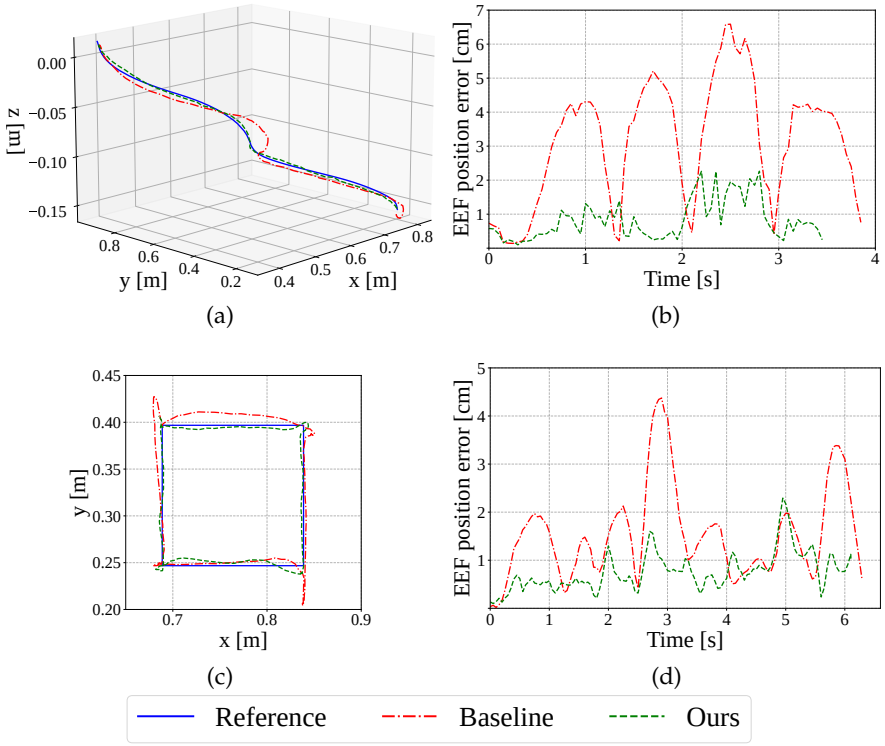


Figure 2.8: Example end-effector trajectories. (a) Path of the end-effector in 3D in one of the test trajectories. (b) Corresponding end-effector position error. (c) Path of the end-effector for a square-shaped trajectory, projected onto the XY plane. (d) Corresponding end-effector position error. Execution with the baseline controller leads to several major deviations, while our approach tracks the trajectories with higher accuracy.

when needed. In this way, our method achieves a much more accurate trajectory tracking of both joint position and velocity.

In Fig. 2.8a, we show the path of the end-effector in one of the test trajectories. One can see that with the baseline controller, the end-effector makes several swings, deviating significantly from the reference trajectory. The peaks of the deviations (Fig. 2.8b) correspond to the peaks of the shoulder yaw deviations. That is because shoulder joints are affected by inertia the most, bearing the weight of the whole arm. It is also worth noticing that the baseline trajectory takes around half a second of extra time to reach the final keyframe. Our method requires almost no extra time. In Fig. 2.8c we show the path of the end-effector for a square-shaped trajectory, projected onto the XY plane. This trajectory has a shape that differs drastically from the trajectories in the training and test sets. Nevertheless, our method significantly improves the accuracy of its execution. Overall, our approach shows better performance in comparison with the other three models and significantly improves over the baseline controller, leading to smoother

Table 2.3: Comparison of average cumulative joint position and velocity errors per point while carrying a payload.

Payload [kg]	Method	Joint position error [rad $\times 10^{-2}$]	Joint velocity error [rad/s $\times 10^{-2}$]
0.25	Baseline	7.37±1.47 (—)	34.14±6.417 (—)
	MLP	5.06±1.36 (31%)	25.94±4.78 (25%)
	RNN	4.51±1.17 (39%)	23.71±3.93 (30%)
	DWH	4.34±1.13 (41%)	23.16±3.74 (32%)
	FIN_u-DWH	4.12±1.08 (44%)	22.53±3.61 (34%)
	FIN_f-DWH	**3.92±0.87 (47%)**	**21.77±3.56 (36%)**
0.5	Baseline	8.01±1.59 (—)	34.69±6.77 (—)
	MLP	5.60±1.43 (30%)	26.47±4.91 (24%)
	RNN	5.11±1.24 (36%)	24.66±4.17 (29%)
	DWH	4.75±1.01 (41%)	23.83±3.94 (31%)
	FIN_u-DWH	4.54±0.98 (43%)	23.21±3.98 (33%)
	FIN_f-DWH	**4.21±0.93 (47%)**	**22.66±3.81 (34%)**
1.1	Baseline	10.83±1.97 (—)	37.80±8.27 (—)
	MLP	8.53±1.71 (21%)	29.16±6.92 (23%)
	RNN	7.78±1.68 (28%)	27.32±5.78 (27%)
	DWH	7.45±1.56 (31%)	26.01±5.14 (31%)
	FIN_u-DWH	7.23±1.39 (33%)	**25.53±4.65 (32%)**
	FIN_f-DWH	**7.13±1.32 (34%)**	25.60±4.63 (32%)

95% confidence interval is provided after "±". Improvement over the baseline is in brackets.

trajectories at high speeds. Achieving higher accuracy over the raw DWH model, the addition of the FIN module is shown to be beneficial.

To further evaluate the efficiency of the proposed approach, we execute 20 random trajectories from the previous experiment with payloads of 0.25, 0.5, and 1.1 kg. Note, that the maximum payload of Baxter is 2.3 kg. Same as in the previous experiment, we measure the average cumulative joint position and velocity errors. These metrics are shown in Table 2.3.

It is possible to see that while carrying the smallest payload of 0.25 kg, all methods perform similarly to the run without the payload (Table 2.1). Increasing the payload to 0.5 kg causes a more noticeable decrease of the trajectory tracking accuracy. Finally, with a significant load of 1.1 kg, the trajectory tracking accuracy deteriorates substantially, influenced by the inertia forces of larger magnitude. The largest deviations are observed in parts of the trajectories with rapid changes in acceleration. Overall, the model ranking is the same as for the experiment without the payload. FIN_f-DWH demonstrates the best results.

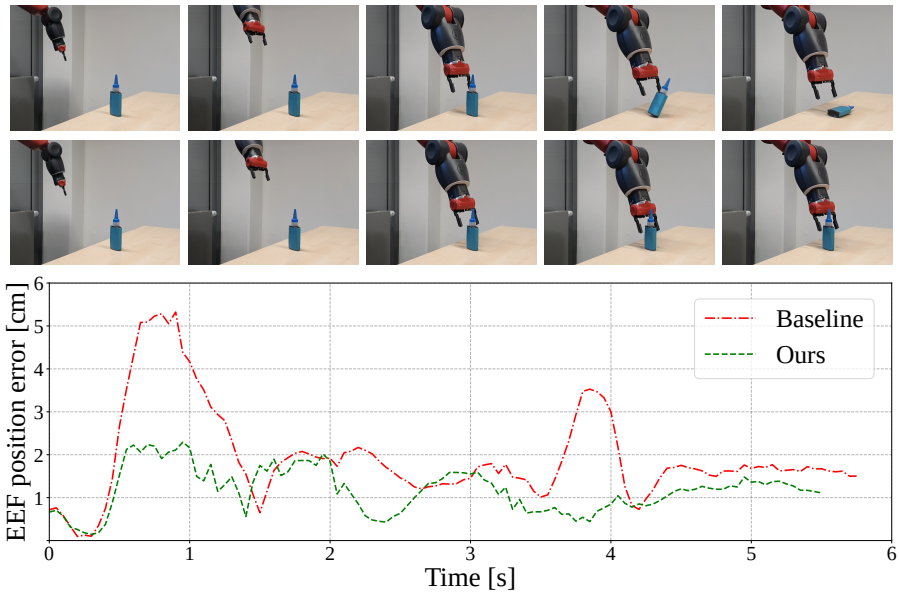


Figure 2.9: Pre-grasp trajectory execution. *Top row:* Using only the baseline controller. *Middle row:* Using our model as an outer-loop controller. *Bottom:* Corresponding end-effector position errors vs. time. In the case of the baseline controller, inaccurate motion leads to a collision with the object and a failed grasping attempt. With our approach, the arm successfully reaches the goal position without colliding with the object.

The velocity is tracked slightly better by FIN_u-DWH in the case of the payload of 1.1 kg. The proposed model is trained on the trajectories without the payload. Nevertheless, this experiment demonstrates that the learned dynamics model of the manipulator helps to reduce the negative impact of the previously unseen payload, achieving more accurate trajectory execution.

2.6.2 *Practical Example*

In Fig. 2.8a, one can see that under the control of the baseline controller, the end-effector arrives at the goal pose in a curve, overshooting the desired path. Such behavior makes it very difficult to perform picking tasks with high speeds because the end-effector often collides with objects at the pre-grasp pose. This often results in failed grasping attempts and can potentially damage the robot and its surroundings. A typical mitigation approach is to define another pre-grasp pose and, upon arrival there, continue at a very low speed to the final pre-grasp pose. However, this slows down the execution of the task.

In this example, we demonstrate the effectiveness of our approach by reaching a pre-grasp pose at high speed. We execute the same trajectory first with the baseline controller and then with the proposed model as an outer loop controller. Two sequences of pictures in Fig. 2.9 show the execution of these trajectories. The baseline controller de-

viates from the path, leading to the collision with the object. This results in tipping over the object. In contrast, our method tracks the trajectory more accurately and has a smoother velocity profile, avoiding the typical overshooting problem. This results in the successfully reached pre-grasp pose without colliding with the object. A video of the experiment is available online[2].

2.7 DISCUSSION

In this chapter, we presented a two-stage model based on LTI dynamical operators for feed-forward outer loop control of a manipulator with flexible joints and unknown complex dynamics. The first part of the model estimates the future state of the system one step ahead, assuming an unchanged control command. This estimation is used to augment the input to the second part of the model, which produces feed-forward joint position and velocity commands. The aim of this two-stage architecture is to push the model from reactive policy behavior towards more intelligent planning. Additionally, such an approach facilitates data-efficiency of the method, because the same dataset can be utilized to learn two different modalities: one-step future prediction and reference correction.

The approach was evaluated on the Baxter robot. The model was trained with backpropagation on a small 45 min real-robot dataset using an ordinary laptop for computations. Data collection and training combined took less than three hours, signifying time-efficiency of the proposed method.

An ablation study showed that the hardwired one-step future prediction improved the trajectory tracking accuracy. In particular, the variant of the model that had the weights of the future prediction model frozen during the final stage of the training outperformed the variant where these weights were updated freely. This signifies the importance of hard-wiring the planning behavior when training on small real-world datasets.

Our approach improved the trajectory tracking accuracy over the baseline controller both without and with previously unseen payloads. Without the payloads, the improvements were 47% and 35% for the joint position and velocity tracking accuracy, respectively. That resulted in a 66% improvement of the end-effector position tracking accuracy compared to the baseline. This contributed to smooth trajectory executions at high velocities that required 92% less extra time to reach the endpoint, allowing performing the tasks faster. The proposed model architecture with a hard-wired one-step future prediction outperformed several other models.

2 https://www.ais.uni-bonn.de/videos/IRC_2021_Pavlichenko

3

TRAJECTORY TRACKING WITH REFERENCE CORRECTION: DEEP REINFORCEMENT LEARNING

PREFACE

This chapter is adapted from Pavlichenko and Behnke, 2022b, previously published by IEEE and presented at the 39th IEEE International Conference on Robotics and Automation (ICRA 2022).

Statement of Personal Contribution

The author of this thesis substantially contributed to all aspects of the publication (Pavlichenko and Behnke, 2022b), including the literature survey, conception, design, and implementation of the proposed method, the preparation and conduct of experiments and evaluation of the proposed approach, the analysis and interpretation of the experimental results, drafting the manuscript, as well as the revision and final editing of the version to be published.

The content presented in this chapter, unless otherwise stated, is the contribution of the author of this thesis.

ABSTRACT

In this chapter, we present an approach for improving trajectory tracking accuracy with a closed-loop stochastic reference correction policy. The policy acts as an outer-loop controller on top of the underlying classical controller. We propose a pipeline to train the policy directly on the real robot with DRL. The reference correction actions are bounded by using beta distribution. This, together with a compact state formulation and a dense reward function, enables starting the learning directly on the real robot. In addition, we propose an informed policy initialization, where the agent is pre-trained in a learned simulation. We demonstrate that the proposed method learns consistently across multiple runs when applied directly on the real robot. The informed initialization significantly reduces the inaccuracies at the start of the learning. In under two hours of training, our method yields a policy that significantly improves the trajectory tracking accuracy in comparison to the vendor-provided controller and the open-loop approach from the previous chapter.

3.1 INTRODUCTION

In Chapter 2 the trajectory tracking of a flexible-joint manipulator is improved through an open-loop reference correction. This correction is provided by a policy represented by a NN. The training of the model is conducted offline using a small real-world dataset in a supervised manner. The proposed method functions as an open-loop feed-forward controller, complementing the underlying classical controller.

The open-loop nature of the method from Chapter 2 restricts the improvement in trajectory tracking accuracy, as the observed tracking error is not accounted for during the reference correction phase. Additionally, the entire pipeline is divided into three stages: data collection, model training, and model deployment for trajectory tracking. This segmented pipeline makes method integration more challenging. In this chapter, we address the trajectory tracking problem with a closed-loop reference correction.

Similarly, the correction is performed by a learned policy, but the learning is conducted online, directly on the real robot. That unifies data collection, training, and model application into one coherent block. To achieve this, we transition from supervised learning to DRL, which is well-suited for the problem where ground-truth reference corrections are unknown and only the observed deviations from the reference are available. Finally, the policy is transparently integrated on top of the classical inner-loop controller.

DRL methods have generated effective policies for a wide range of control tasks (Hwangbo et al., 2019; Z. Li et al., 2021; Rodriguez and Behnke, 2021). Most DRL approaches depend on learning in simulation and the consequent sim-to-real transfer. However, accurate simulations of robots are often unavailable. In this chapter, we present an approach capable of learning an outer-loop control policy with DRL online, directly on the real robot. The policy operates at a lower frequency than the underlying classical controller and provides bounded reference correction actions, as shown in Fig. 3.1. These corrections are applied to the reference trajectory before it is fed to the classical controller.

This formulation makes the method agnostic of the underlying classical controller type. Our approach can also be interpreted as an online closed-loop trajectory optimization. The corrective actions of bounded magnitude alleviate safety concerns while training the model online on the real robot. To shorten the real-robot training time, we use an off-policy soft actor-critic (SAC) method, which has been shown to have a lower sample complexity and more stable convergence compared to other DRL approaches (Haarnoja, A. Zhou, Hartikainen, et al., 2018). To further accelerate learning, we propose an *informed initialization*: policy pretraining in a *learned* simulation.

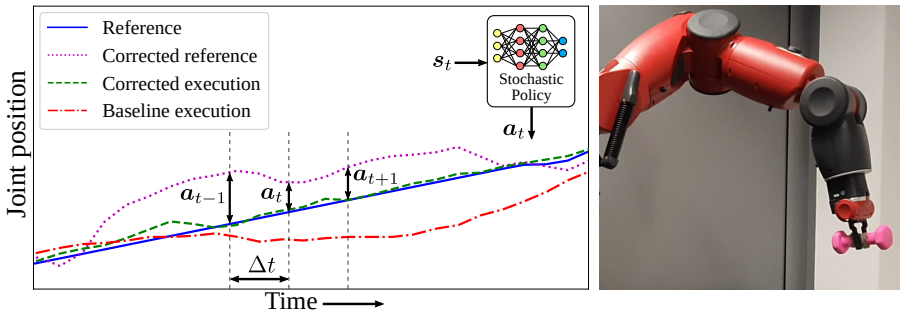


Figure 3.1: Stochastic reference correction policy learned directly on the real robot. *Left:* The policy learns to produce actions $\boldsymbol{a}_t$ from the states $\boldsymbol{s}_t$, which correct an arbitrary reference trajectory (blue solid line) with frequency Δt, resulting in a corrected reference (magenta dotted line), which is fed to the vendor-provided controller and leads to an improved trajectory tracking accuracy (green dashed line) as opposed to the sole vendor-provided controller (red dash-dotted line). *Right:* Baxter executes a trajectory while carrying a payload.

The evaluation is conducted on the 7 DoF arm of a Baxter robot. The policy is learned in less than two hours. Our experiments show that incorporating the learned high-level control policy significantly enhances trajectory tracking accuracy compared to the vendor-provided classical controller. We also assess the policy's performance under changes in the dynamics by conducting experiments with a previously unseen payload. The results indicate a sustained improvement in trajectory tracking accuracy. Finally, the proposed method outperforms the open-loop approach presented in Chapter 2, underscoring the importance of integrating live feedback for reference trajectory correction.

In summary, our main contributions are:

- Action, state, and reward formulation to learn a reference correction policy directly on the real robot with DRL,
- informed initialization of the policy through a coarse dynamics model *learned* from data, which is used as a simulator.

3.2 RELATED WORK

Trajectory tracking control methods can generally be categorized into model-based (An, Atkeson, and Hollerbach, 1988) and model-free approaches (Longman, 2000). Model-free methods are particularly effective for addressing the limitations posed by imperfect models. Classical examples of such approaches include ILC (Bristow, Tharayil, and Alleyne, 2006) and repetitive control (RC) (Cuiyan, Dongchun, and Xianyi, 2004). Nevertheless, these techniques are specifically designed for the repetitive execution of predefined trajectories. In contrast, policies modeled by deep neural network (DNN) and trained using

DRL exhibit a strong capacity to generalize to new trajectories while effectively approximating complex, nonlinear dynamics (Haarnoja, A. Zhou, Abbeel, et al., 2018).

First, we make a brief overview of the widely used DRL algorithms. On-policy methods, such as trust region policy optimization (TRPO) (Schulman, Levine, et al., 2015), proximal policy optimization (PPO)(Schulman, Wolski, et al., 2017), and asynchronous advantage actor-critic (A3C) (Mnih et al., 2016), do not utilize past experience. Consequently, their high sample complexity poses significant challenges for learning directly on a real robot.

In contrast, off-policy algorithms leverage past experiences, substantially reducing sample complexity. A well-known example is deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), which, however, is very challenging to tune. DreamerV2 (Hafner et al., 2021) learns a latent dynamics model of the environment, enabling it to plan and simulate trajectories in the latent space. However, that introduces a significant computational overhead. SAC (Haarnoja, A. Zhou, Abbeel, et al., 2018) is much more computationally efficient. It modifies the RL objective by introducing a maximum entropy term (Haarnoja, Tang, et al., 2017). This feature enhances both sample efficiency and learning stability. Because of that, we utilize this approach in our work.

There are numerous RL-based approaches to robotic manipulator control (Franceschetti et al., 2022). Pradhan and Subudhi, 2012 use actor-critic RL to adjust the model parameters in response to the payload variations on a two-link manipulator. Z. Xu et al., 2021 propose to learn a policy with DDPG to control a two-link manipulator in simulation. An RL agent acting as a nonlinear input compensator over the traditional controller is presented by Bayiz and Babuska, 2014. It is applied to a 1 DoF robot in simulation. This idea is further explored (Pane et al., 2019), and applied to a 5 DoF UR5 manipulator, introducing a notion of RL-based reference compensation. A corrective controller is trained for each joint. Our action definition resembles the formulation of reference compensation. However, we train a single policy for all joints at once, which is advantageous when dealing with coupled flexible joints. Finally, we train the policy with DRL online on the real robot, promoting generalization to a broad range of trajectories.

In some works, the approaches operate in task-space (Shao, Migimatsu, and Bohg, 2020; Y. Zhu et al., 2018). While this may be beneficial in certain cases, we focus on the joint space since it avoids redundancy. Cao et al., 2023 present a RL-based fixed-time trajectory tracking control method for uncertain robotic manipulators with input saturation. The proposed approach aims to ensure accurate trajectory tracking within a fixed time, regardless of the system's initial conditions or uncertainties, such as unknown dynamics and external disturbances. By incorporating input saturation constraints, the method enhances

real-world applicability where actuators have limited control input. X. Li, Shang, and Cong, 2024 utilize physics-inspired deep models to learn both the kinematics and the dynamics of a robotic manipulator. The model-based learning is done offline and is combined with a traditional computed-torque controller. The method is evaluated both in simulation and on a real 7 DoF arm. The advantage of our method is that it is transparently applied to the real robot online, on top of the classical controller, avoiding the need for an accurate simulation model.

Hu and Si, 2018 control a two-link robotic manipulator by means of a NN-based model in simulation. An approach to mobile manipulation pick-and-place tasks is demonstrated by Iriondo et al., 2019. The authors train a policy to control a mobile base with PPO and DDPG. The policy input is provided directly from the arm manipulation planner and is used to determine positions where the arm can reach an object for picking.

Kumar et al., 2021 present a joint-level controller for robotic manipulators, trained with PPO. The policy maps reference trajectory from joint to task space and is trained in simulation. It is also applied to the real robots via sim-to-real transfer and is shown to achieve similar accuracy as traditional controllers. The method is extended to facilitate obstacle-avoiding behavior by augmenting the input to the network with obstacle proximity information. Calderon-Cordova and Sarango, 2023 propose an approach to learn reaching policies in simulation with advantage actor-critic (A2C). The method is applied to a 7 DoF Franka Emika Panda arm.

Y. Li et al., 2024 introduce Cluster-SAC framework. There, a given trajectory is divided into segments, and a separate SAC agent is assigned to track the segment. Such an approach can be beneficial for very complex trajectories. The learning and evaluation are done in simulation.

These methods train policies in simulation that are then transferred to a real robot. This makes the availability of an accurate simulation necessary. The resulting controller is used as a final product, which discards advantages of learning while interacting with the real system. Moreover, in this case, retraining is needed if the dynamics change, for instance, due to wear and tear over time. Such retraining requires the aforementioned changes to be transferred back to the simulation first.

In contrast, we present an approach to learn a policy directly on the real robot. This eliminates the need for simulation and enables continuous learning from new experiences, allowing the system to adapt to changes such as wear and tear. Furthermore, instead of fully replacing the classical controller, we enhance it with a reference correction policy as an outer-loop controller. This integration allows the two methods to complement each other.

3.3 BACKGROUND

The essence of RL is learning through interaction with the environment. The agent observes its current state, performs an action, and receives the reward, quantifying how good the action was. By doing this repeatedly, the policy is updated so that the taken actions yield higher rewards. Thus, the objective of RL is to find a policy π that maximizes the expected discounted sum of rewards. In the context of DRL, a policy π_θ is represented by a DNN, parameterized by learnable weights θ. The problem is modeled as a Markov decision process (MDP): $\{S, A, P, r\}$ with state space $S \in \mathbb{R}^n$, action space $A \in \mathbb{R}^m$, state transition function $P\colon S \times A \mapsto S$, and a reward function $r\colon S \times A \mapsto \mathbb{R}$. Thus, the objective is formulated as:

$$J(\pi_\theta) = \sum_{t=0}^{T} \gamma \mathbb{E}[r(\boldsymbol{s}_t, \boldsymbol{a}_t)], \tag{3.1}$$

where $\gamma \in [0, 1]$ is a discounting factor.

In this work, we address the problem with continuous state and action space. Thus, we define a stochastic policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{s})$, representing an action probability distribution when observing a state $\boldsymbol{s}_t$ at timestep t. We use the off-policy SAC algorithm (Haarnoja, A. Zhou, Abbeel, et al., 2018) to train the policy. SAC is based on an actor-critic approach, where an actor provides actions and a critic represents the value function. SAC follows a maximum entropy RL formulation, which optimizes both expected reward and the entropy of the policy:

$$J(\pi_\theta) = \sum_{t=0}^{T} \mathbb{E}[\gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha\, \mathcal{H}(\pi_\theta(\cdot|\boldsymbol{s}_t))], \tag{3.2}$$

where $\mathcal{H}$ is the entropy of the stochastic policy and $\alpha \in [0, 1]$ is the temperature parameter. Note that with $\alpha = 0$ the above equation reduces to the conventional RL objective (Eq. 3.1). SAC is defined through a soft Q-function $Q_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and a policy $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$, parameterized by parameters ϕ and θ respectively. The soft Q-function parameters can be trained to minimize the soft Bellman residual:

$$J(Q_\phi) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t)\sim\mathcal{D}}\Big[\frac{1}{2}(Q_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t) - (r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1}\sim p}[V_{\bar{\phi}}(\boldsymbol{s}_{t+1})]))^2\Big], \tag{3.3}$$

where $\bar{\phi}$ are the parameters of the target soft Q-function and are obtained as an exponentially moving average of the soft Q-function weights. The value function V is implicitly parameterized through the soft Q-function parameters:

$$V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t\sim\pi}[Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)]. \tag{3.4}$$

The soft Q-function can be optimized with stochastic gradients:

$$\hat{\nabla}_\phi J(Q_\phi) = \nabla_\phi Q_\phi(\boldsymbol{a}_t, \boldsymbol{s}_t)(Q_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t) - (r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma Q_{\overline{\phi}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) - \alpha log(\pi_\theta(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1})))). \quad (3.5)$$

Finally, the policy parameters can be learned:

$$J(\pi_\theta) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}}[\mathbb{E}_{\boldsymbol{a}_t \sim \pi_\theta}[\alpha log(\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)) - Q_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t)]]. \quad (3.6)$$

The policy is reparameterized using the neural network transformation:

$$\boldsymbol{a}_t = f_\theta(\boldsymbol{\epsilon}_t; \boldsymbol{s}_t), \quad (3.7)$$

where $\boldsymbol{\epsilon}_t$ is an input noise vector, sampled from some fixed distribution, such as a spherical Gaussian. Now Eq. 3.6 can be rewritten as:

$$J(\pi_\theta) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}, \boldsymbol{\epsilon}_t \sim \mathcal{N}}[\alpha log \pi_\theta(f_\theta(\boldsymbol{\epsilon}_t; \boldsymbol{s}_t)) - Q_\phi(\boldsymbol{s}_t, f_\theta(\boldsymbol{\epsilon}_t; \boldsymbol{s}_t))]. \quad (3.8)$$

The maximum entropy formulation incentivizes the policy exploration and was shown to produce robust policies and achieve stable learning. In this work, we use a SAC version with automatic tuning of the temperature parameter α. A detailed description of the algorithm can be found in the original paper by Haarnoja, A. Zhou, Abbeel, et al., 2018.

3.4 METHOD

We propose an approach to improve joint trajectory tracking by employing a stochastic reference correction policy, learned with DRL directly on the real robot. The policy serves as an outer-loop controller over the underlying classical controller. Given an arbitrary reference trajectory $\boldsymbol{q}_\mathrm{r}$ of N equally spaced in time joint configurations $\boldsymbol{q}_\mathrm{r}(t_1) \dots \boldsymbol{q}_\mathrm{r}(t_N), i \in [1 \dots N]$ with $\Delta t = t_i - t_{i-1}$ and duration $T = t_N - t_1$, our goal is to minimize the trajectory tracking error. The notation for trajectory, keyframes, and joint angles is identical to the previous Chapter 2. Given a baseline with a subpar tracking accuracy, we propose to improve its performance by augmenting the control loop with an outer-loop learned policy.

The underperformance of the baseline occurs when the available model is not accurate enough, the baseline was not tuned well enough, or the robot hardware accumulated significant wear and tear. Thus, achieving higher accuracy with the baseline requires developing a better baseline or performing tedious, instance- or task-specific tuning. These solutions are time-consuming and require a highly skilled professional. We propose to utilize recent developments of DRL in order

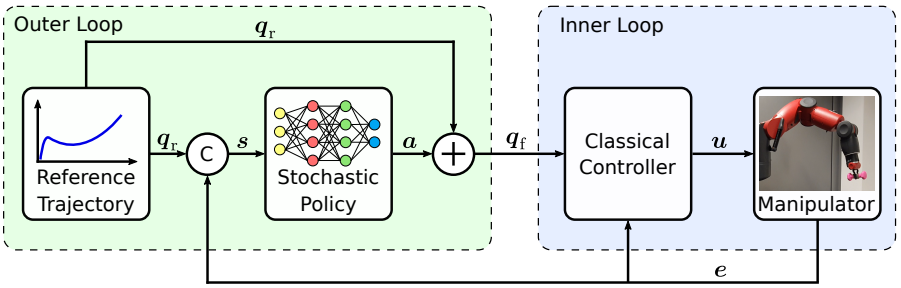


Figure 3.2: Closed-loop reference correction control architecture. Given reference trajectory q_r and feedback e, which form a state s, the learned stochastic policy corrects the reference q_r with action a, resulting in a corrected reference q_f. The classical controller produces the control signal u for the actuators. (C): concatenation.

to learn a policy that compensates for the inaccuracies of the baseline. The learning is done online on the real robot, which avoids the need for an accurate simulation and consequent sim-to-real transfer.

In the absence of an accurate model, DRL allows for efficient learning of the control policy, guided by the reward function. The trajectory tracking problem has *dense rewards*, which facilitates the learning process. Since we aim to perform the learning directly on the real robot, the DRL system should have high *robot persistence* (Ibarz et al., 2021), consisting of two components.

The first component is *self-persistence*. The robot must not damage itself while training. We use strictly bounded actions, which represent a corrective term for the reference trajectory. Thus, at any given time, the potential deviation from the reference trajectory is bounded. Selection of the maximum allowed magnitude of the corrective action provides a necessary flexibility when setting up the learning system. In addition, the corrective actions are filtered with a low-pass filter. Note that we assume that the reference trajectories are collision-free. Hence the safety concerns raised here are related solely to the influence of models' actions on the movement of the arm but not on the interactions with the environment.

The second component is *task-persistence*. The robot must learn and collect data with minimal human assistance. The state of the manipulator at any time step is determined from the actuator encoders. Although there is noise, its magnitude is negligible since the learned policy operates at a relatively low frequency. Thus, there is no need for any additional software or hardware, which makes the learning setup and process coherent: the manipulator learns online from its own trial and error.

We choose a reference correction instead of input correction because, first, input correction has to be done with a much higher frequency, which decreases the influence of a single action, obstructing the learning. Moreover, higher frequencies mean higher impact of latencies and noise, which makes the learning even more challenging. Second,

reference correction is agnostic of the underlying baseline control law, which makes the approach more flexible. The control diagram is shown in Fig. 3.2.

3.4.1 *Action Space*

The action $\boldsymbol{a}(t) \in \mathbb{R}^M$ for a robot with M joints, produced by a learned stochastic continuous control policy, represents a reference correction term, such that $\boldsymbol{q}_\mathrm{f}(t+1) = \boldsymbol{q}_\mathrm{r}(t+1) + \boldsymbol{a}(t)$, where $\boldsymbol{q}_\mathrm{r}(t+1)$ is a reference trajectory point in joint space and $\boldsymbol{q}_\mathrm{f}(t+1)$ is a resulting corrected reference. The corrective action $\boldsymbol{a}(t)$ produced at timestep t is designated to change the next reference trajectory point at timestep $t+1$ such that commanding the inner-loop classical controller to reach $\boldsymbol{q}_\mathrm{f}(t+1)$ would result in the manipulator reaching $\boldsymbol{q}_\mathrm{r}(t+1)$. We strictly bound the actions $\boldsymbol{a}(t) \in [-\boldsymbol{a}_\mathrm{max}, \boldsymbol{a}_\mathrm{max}]$ by a predefined constant vector $\boldsymbol{a}_\mathrm{max}$. The stochastic policy runs at a frequency of 20 Hz, so $\Delta t = t_i - t_{i-1} = 0.05\,\mathrm{s}$. The produced actions $\boldsymbol{a}$ are filtered by a low-pass filter with a cutoff frequency of 4 Hz to smooth out any inconsistent signal. The predefined constant $\boldsymbol{a}_\mathrm{max}$ regulates the amount of influence the learned policy has over the reference trajectory. In this work, we choose to define $\boldsymbol{a}_\mathrm{max}$ as:

$$\boldsymbol{a}_\mathrm{max} = \frac{0.95 \cdot \dot{\boldsymbol{q}}_\mathrm{lim} \Delta t}{2}, \tag{3.9}$$

where $\dot{\boldsymbol{q}}_\mathrm{lim}$ is a joint velocity limit vector. Thus, having two consecutive actions $\boldsymbol{a}(t) = -\boldsymbol{a}_\mathrm{max}$ and $\boldsymbol{a}(t+1) = \boldsymbol{a}_\mathrm{max}$ would satisfy the velocity limit $\dot{\boldsymbol{q}}_\mathrm{lim}$. Indeed, as the corrective action is applied to the reference trajectory, this does not guarantee preserved velocity limits. However, it is a meaningful standardized formulation of $\boldsymbol{a}_\mathrm{max}$ which can be used as a baseline value. Each point $\boldsymbol{q}_\mathrm{f}(t)$ is checked for joint, velocity, and acceleration limits before being commanded to the underlying controller. In case a limit violation is detected, a point is clipped to the corresponding limit.

In this work we use a classical vendor-provided Baxter robot controller as an inner-loop controller. It takes both joint positions as well as joint velocities to produce torque controls. Hence, in our case we define action $\boldsymbol{a}(t) \in \mathbb{R}^{2M}$, such that $\boldsymbol{a} = [\Delta\boldsymbol{q}, \Delta\dot{\boldsymbol{q}}]$. It is still straightforward to define the second term of $\boldsymbol{a}_\mathrm{max}$, using the acceleration limits in Eq. 3.9. Note that the presented action representation is generic, and it is straightforward to have any combination of joint positions and their derivatives as action representation.

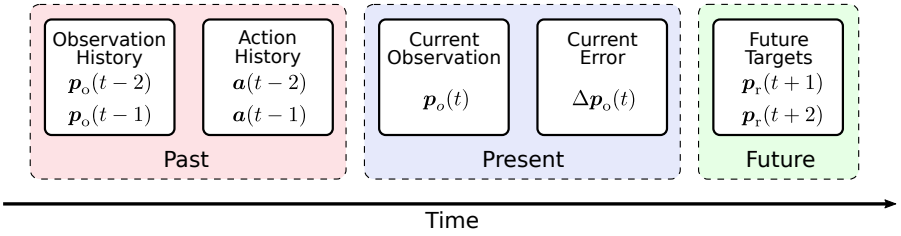


Figure 3.3: State representation. The state contains information about the manipulator from three major periods along the time axis. *The past:* History of observations and actions. *The present:* Currently observed manipulator state and deviation from the reference. *The future:* The next upcoming reference trajectory points. Such state representation reflects the dynamics of the manipulator and helps to mitigate the effects of latency.

3.4.2 *State Space*

The state $\boldsymbol{s}(t)$ of a robotic manipulator is a column vector:

$$\boldsymbol{s}(t) = \begin{bmatrix} \boldsymbol{p}_{\mathrm{o}}(t-2), \boldsymbol{p}_{\mathrm{o}}(t-1), \\ \boldsymbol{a}(t-2), \boldsymbol{a}(t-1), \\ \boldsymbol{p}_{\mathrm{o}}(t), \Delta\boldsymbol{p}_{\mathrm{o}}(t), \\ \boldsymbol{p}_{\mathrm{r}}(t+1), \boldsymbol{p}_{\mathrm{r}}(t+2) \end{bmatrix}, \tag{3.10}$$

where point $\boldsymbol{p}$ contains joint positions and velocities: $\boldsymbol{p} = [\boldsymbol{q}, \dot{\boldsymbol{q}}]$, $\boldsymbol{p}_{\mathrm{o}}$ is an observed point, read from the joint encoders, $\boldsymbol{p}_{\mathrm{r}}$ is a point from the reference trajectory, $\Delta\boldsymbol{p}_{\mathrm{o}}(t) = \boldsymbol{p}_{\mathrm{o}}(t) - \boldsymbol{p}_{\mathrm{r}}(t)$ is an observed error. Such state representation contains information about the manipulator from the past, the present, and the future.

The first four components contain past observations and actions. The next two components contain current observations, augmented with the observed error. The last two components contain the future desired joint positions and velocities. Our state representation provides the policy with information about the past, the current state, and the future targets. Inclusion of the past observations and actions helps to combat the negative effects of latency (Ibarz et al., 2021; Riedmiller, 2012). In addition, it provides information about the dynamics of the manipulator, further strengthened by the inclusion of joint velocities $\dot{\boldsymbol{q}}$. The state representation diagram is shown in Fig. 3.3.

Several future reference points $\boldsymbol{p}_{\mathrm{r}}$ provide more information about the desired motion of the manipulator. Each term in the state $\boldsymbol{s}(t)$ is rescaled to the $[-1, 1]$ interval. It is straightforward to do so for $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\boldsymbol{a}$, given joint limits $\boldsymbol{q}_{\mathrm{lim}}$, joint velocity limits $\dot{\boldsymbol{q}}_{\mathrm{lim}}$, and action magnitude $\boldsymbol{a}_{\mathrm{max}}$. However, there is no obvious way to choose a rescaling interval for $\Delta\boldsymbol{q}$ and $\Delta\dot{\boldsymbol{q}}$. In this work, we interpolate them within $[-(\mu_j + 3\sigma_j), \mu_j + 3\sigma_j]$, where μ_j and σ_j are the measured per-joint errors of positions and velocities. These parameters have to be

estimated empirically by executing several random trajectories with the baseline controller.

3.4.3 *Reward Function*

For each timestep t, we define the reward function:

$$r(t) = \omega r_q(t) + (1 - \omega) r_{\dot{q}}(t), \tag{3.11}$$

where $r_q \in [0, 1]$ is a reward term encouraging joint position tracking, $r_{\dot{q}} \in [0, 1]$ is a reward term encouraging joint velocity tracking, and $\omega \in [0, 1]$ is a relative importance scaling factor. Given that, $r \in [0, 1]$. To compute r_q and $r_{\dot{q}}$, we first define the cumulative absolute errors e_q for the joint position and $e_{\dot{q}}$ for the joint velocity:

$$e_q(t) = \sum_{j=0}^{j=M} |q_o(t, j) - q_r(t, j)|, \tag{3.12}$$

$$e_{\dot{q}}(t) = \sum_{j=0}^{j=M} |\dot{q}_o(t, j) - \dot{q}_r(t, j)|, \tag{3.13}$$

where M is the number of joints, $q_o(t, j)$ is the observed position of joint j, and $q_r(t, j)$ is the desired position at time step t. The procedure is analogous for $e_{\dot{q}}$. Finally, we use the smooth logistic kernel function K (Rodriguez and Behnke, 2021):

$$K(x, l) = \frac{2}{e^{xl} + e^{-xl}} \tag{3.14}$$

to define the reward terms:

$$r_q(t) = K(e_q(t), l_q), \tag{3.15}$$

$$r_{\dot{q}}(t) = K(e_{\dot{q}}(t), l_q), \tag{3.16}$$

where l is the kernel sensitivity parameter. When a trajectory is tracked perfectly, $r = 1$. We use the L^1 norm in the error calculations (Eq. 3.12 and Eq. 3.13). When we experimented with using the L^2 norm, we noticed that while learning, the policy would aggressively abuse joints with smaller errors to compensate for joints with bigger errors. That led to learning the unstable and counterintuitive motions of the manipulator.

We include the joint velocity tracking term $r_{\dot{q}}(t)$ into the reward function to promote trajectory smoothness. We observed that without this term, the policy would often learn a bang-bang style of control,

leading to non-smooth motions. Adding an explicit action cost term often leads to abusive behavior when the policy learns to generally avoid making any corrections for a while, followed by sudden corrections of large magnitude in-between. Tuning this action cost such that the policy achieves an intended trajectory tracking behavior is extremely challenging. Thus, we argue that rewarding joint velocity tracking is a natural way to promote trajectory smoothness by encouraging following the derivative of the reference path. Since the objective of this work is to improve joint position tracking accuracy, we set $\omega = 0.75$.

3.4.4 *Model*

In this work, we train the stochastic policy using the SAC algorithm (Haarnoja, A. Zhou, Abbeel, et al., 2018). The method is off-policy, making this choice natural for the real-robot learning due to its increased sample efficiency.

In the original SAC, the actions supplied by the actor network are bounded to a finite range through the use of a Gaussian policy with the squashing function. Instead, we use a beta policy (Chou, Maturana, and Scherer, 2017), which is bounded in the $[0, 1]$ range by definition. This is an advantage in the context of real-robot learning, since the maximum magnitude of the actions can be guaranteed, making the learning safer with an option to choose an optimal action magnitude, depending on the system and the task.

Each of the actor and critic networks is an MLP. Both of them have similar structures, consisting of two hidden fully-connected layers with tanh activation functions. The output layer of the Q-networks, when supplied with a state-action pair $\{\boldsymbol{s}(t), \boldsymbol{a}(t)\}$, provides a single Q-value. The actor network outputs two values for each dimension of the action space, resulting in $2 \cdot |\boldsymbol{a}|$ outputs. Each pair of these values parameterizes a beta distribution, from which the actions are sampled. The sampled actions are in the $[0, 1]$ range. It is straightforward to rescale them to the range of choice, such as $[-\boldsymbol{a}_{\max}, \boldsymbol{a}_{\max}]$, used in this work.

We use the sigmoid activation function in the output layer of the actor network, as it resembles well the $(0, \infty]$ ranges of beta distribution parameters α and β. Since beta distribution parameters $\in (0, \infty]$, the outputs of the actor are clipped to $[\epsilon, 1]$, where $\epsilon = 1 \times 10^{-5}$. Finally, we scale them up by a factor of 10 to provide the actor with enough freedom for distribution selection. During training, the actions are sampled from the distributions, while during inference we use their modes.

3.4.5 *Learning Process*

A single episode corresponds to an execution of a reference trajectory consisting of N joint positions $\boldsymbol{q}(t)$ and velocities $\dot{\boldsymbol{q}}(t)$, equally spaced in time with interval Δt. Assuming that an underlying low-level classical controller is capable of bringing the arm into the final keyframe of the trajectory, every episode has a finite duration.

During each episode, at each time step t with the observed state $\boldsymbol{s}_t$, the policy provides a reference correction action $\boldsymbol{a}(t)$. The correction is then added to the next keyframe of the reference trajectory. The modified reference is fed to the classical controller, leading to a state $\boldsymbol{s}(t+1)$ with reward $r(t)$. The tuple $\{\boldsymbol{s}(t), \boldsymbol{a}(t), r(t), \boldsymbol{s}(t+1)\}$ is stored in the replay buffer for experience replay. We perform SAC update iterations with a replay ratio (Fedus et al., 2020) of 1: one update per one point added to the buffer. The weights of the actor network are updated after each episode. For each SAC training iteration, a minibatch of uniformly sampled datapoints is generated.

3.4.6 *Informed Initialization*

It is beneficial to have a deliberate initialization of the model before applying it to the real robot, since a randomly initialized policy yields a low-reward behavior. Such exploratory behavior can be dangerous for the robot and its environment. Additionally, in case of the integration of the learning pipeline into the system while continuing normal use, low-reward behavior may cause the tasks to be failed. To mitigate these risks in the absence of an accurate simulation, we propose to learn a coarse simulation, represented by a NN, to pre-train the policy.

First, a small sample of random trajectories from the real robot is recorded. Then, we train an MLP consisting of three fully connected layers to predict $[\boldsymbol{q}_\mathrm{o}(t+1), \dot{\boldsymbol{q}}_\mathrm{o}(t+1)]$, given $[\boldsymbol{q}_\mathrm{o}(t), \dot{\boldsymbol{q}}_\mathrm{o}(t)]$ and $[\boldsymbol{q}_\mathrm{r}(t+1), \dot{\boldsymbol{q}}_\mathrm{r}(t+1)]$. The network architecture and training procedure are identical to the FIN from Section 2.4. Finally, we use this learned coarse dynamics model as a simulation environment to pre-train the stochastic policy before applying it to the real robot.

This pre-training step keeps the approach generic, as it does not require any additional knowledge about the manipulator dynamics. At the same time, it approximates latencies and noise from the real system, informing the policy about its future experiences. Thus, we refer to this approach as *informed initialization*. The training of the policy using this learned simulation is structurally identical to the learning on the real robot and is performed as described in Section 3.4.5. In addition, the data collected for the informed initialization is used to determine scaling factors for $\Delta\boldsymbol{q}$ and $\Delta\dot{\boldsymbol{q}}$ (Section 3.4.2).

3.5 EVALUATION

To evaluate the proposed approach, we apply it to the left arm of the Baxter robot. It is a 7 DoF manipulator with flexible joints. The underlying dynamics model is unknown, and flexible joints with coupled dynamics are challenging to control. In this evaluation we aim to answer the following questions:

- Does our approach improve the joint position trajectory tracking accuracy?
- Does it generalize to handle an unseen payload?
- Is the learning consistent and safe for the hardware?

3.5.1 *Setup*

The classical inner-loop controller, which we further refer to as the baseline, is the Baxter inverse dynamics controller. It calculates commanded torques from the supplied joint positions and velocities. We choose it as the baseline in our experiments because it shows the best joint position tracking accuracy compared to the Baxter position and velocity PD controllers. In the case of this baseline, we employ reference correction for both joint position and velocities; hence $\boldsymbol{a}(t) = [\boldsymbol{a}_q(t), \boldsymbol{a}_{\dot{q}}(t)] \in \mathbb{R}^{2M}$, where $\boldsymbol{a}_q(t)$ is the reference joint position correction and $\boldsymbol{a}_{\dot{q}}(t)$ is the reference joint velocity correction. With $M = 7$ this results in a 14-element action vector. The maximum magnitude of the part of the action designated for velocity correction is defined as described in Section 3.4.1, but using joint acceleration limits instead of the velocity limits.

According to the state representation from Eq. 3.10 and the provided 14-element $\boldsymbol{a}$ and $\boldsymbol{p}$, we obtain a state vector of $8 \times 14 = 112$ elements. We use the same size of 80 neurons for hidden layers for both critic and actor networks. This results in a $112 \times 80 \rightarrow 80 \times 80 \rightarrow 80 \times 28$ actor network and a $(112 + 14) \times 80 \rightarrow 80 \times 80 \rightarrow 80 \times 1$ critic network, as shown in Fig. 3.4.

We use the Adam optimizer with triangular learning rate scheduling, ranging from 1×10^{-4} to 4×10^{-4} with a period of 100 episodes. We set the discount factor $\gamma = 0.85$ and perform a hard critic update with $\tau = 1$ every 1000 iterations. We use a minibatch of 128 datapoints. The kernel sensitivities were set to $l_q = 32$ for the joint position tracking reward term and $l_{\dot{q}} = 7$ for the joint velocity tracking reward term.

Since the reward scale is critical for SAC due to the entropy maximization (Haarnoja, A. Zhou, Abbeel, et al., 2018), we set its value to 10, as we empirically found that larger or smaller values resulted in inferior learning. The training is performed on a regular laptop with an Intel i7-6700HQ CPU and 16 GB of random-access memory (RAM).

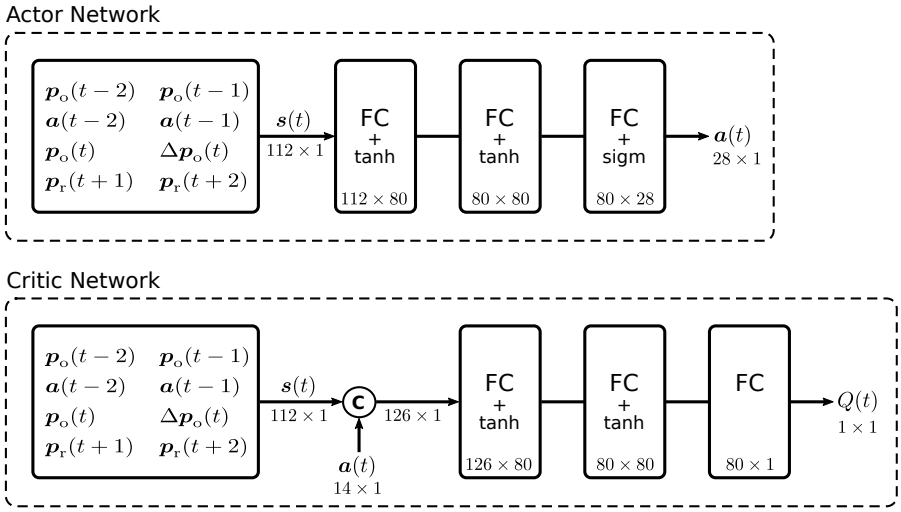


Figure 3.4: Actor and critic network architectures. Given a state $\boldsymbol{s}(t)$, the actor network outputs a corrective action $\boldsymbol{a}(t)$. Given a state-action pair $\boldsymbol{s}(t), \boldsymbol{a}(t)$, the critic network outputs state-action value estimate $Q(t)$. (C): Concatenation.

3.5.2 *Experiments*

To learn the stochastic reference correction policy (SRCP), we generate random trajectories from the workspace of Baxter's left arm. We cover the region Y in front and to the side of the robot with approximate dimensions of $1.4 \times 0.7 \times 1.0$ m. Each trajectory is defined by a start and a goal, plus 1-3 intermediate points. Each point represents a 6D end-effector pose and is drawn randomly from Y. An IK solver is used to convert trajectories to joint space. A trajectory is checked for joint, velocity, and acceleration limits, as well as for collisions, before execution. If the trajectory is unfeasible, it is discarded and a new one is generated.

To evaluate the effectiveness of the informed initialization, as proposed in Section 3.4.6, we also train the coarse dynamics model to represent the simulation. The model is a MLP and has three fully connected layers: $28 \times 64 \rightarrow 64 \times 32 \rightarrow 32 \times 14$. We train this model with the Adam optimizer and a learning rate of 10^{-4} until convergence on a small real-robot dataset of 200 trajectories with a total duration of 20 min. Finally, we train two variants of SRCP. The first one has random weights. The second one is first pre-trained in the learned simulation. We refer to the latter as SRCP+InformedInit.

We perform three training runs for each model. A run consists of 1000 random trajectories. One run completes in under 2 hours, with approximately 100 min dedicated to the trajectory execution. In Fig. 3.5, we show the training reward curves as well as the joint position and joint velocity tracking errors, averaged over the three runs. Since each trajectory has a different number of points, the cumulative

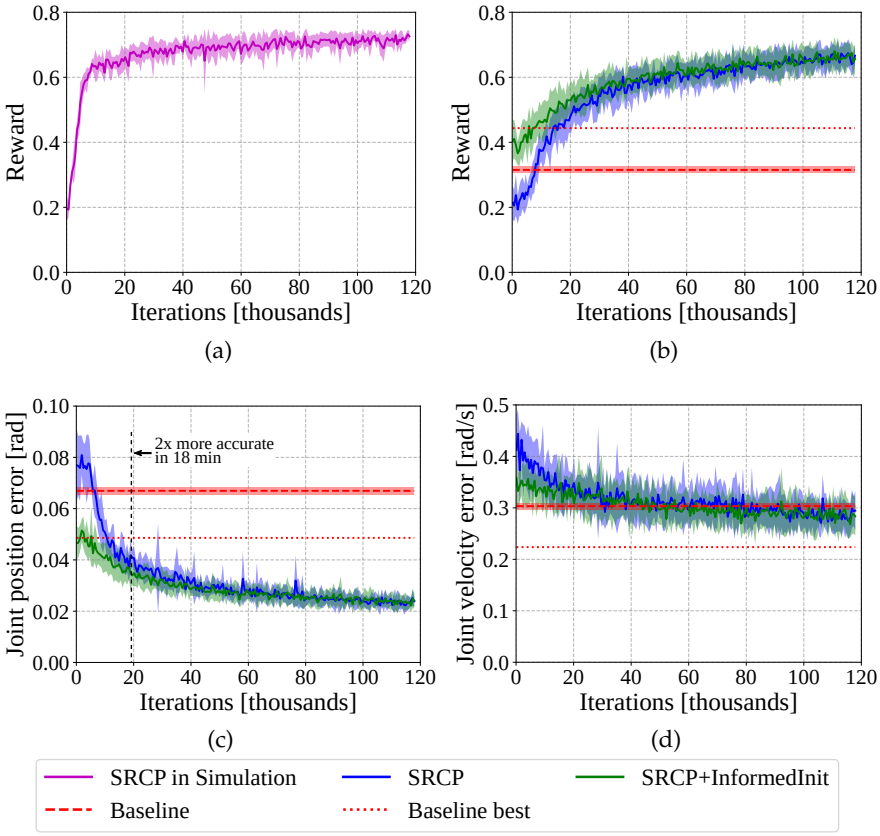


Figure 3.5: Training curves with 1000 trajectories. (a) SRCP reward in learned simulation. (b) Rewards on the real robot. (c) Average joint position tracking error. (d) Average joint velocity tracking error. Both variants of our approach achieve stable learning. SRCP with informed initialization (green) outperforms the baseline (red) directly from the start. Lines represent the mean, and shaded regions represent 95% confidence intervals, averaged over three runs. Baseline best: the best result achieved by the baseline controller. Average iterations per trajectory: 118. 10K iterations ≈ 9 min of real time.

values would not be representative, and we show all values averaged per-point.

The values for the baseline are calculated from 100 random trajectories. While training in a learned simulation, the SRCP quickly reaches relatively high rewards, as can be seen in Fig. 3.5a. The training in the simulation takes less than 20 min. On the real robot, the policy learns high-reward behaviors slower (Fig. 3.5b) due to the latencies and complex dynamics. One can see that SRCP+InformedInit directly starts in the reward region above the baseline, achieving higher rewards faster than the randomly initialized policy. Note that SRCP+InformedInit becomes twice as accurate as the baseline only in 18 min. Both policies converge to a similar performance in the end. The per-joint learning curves are shown in Fig. 3.6. Note that our method improves the

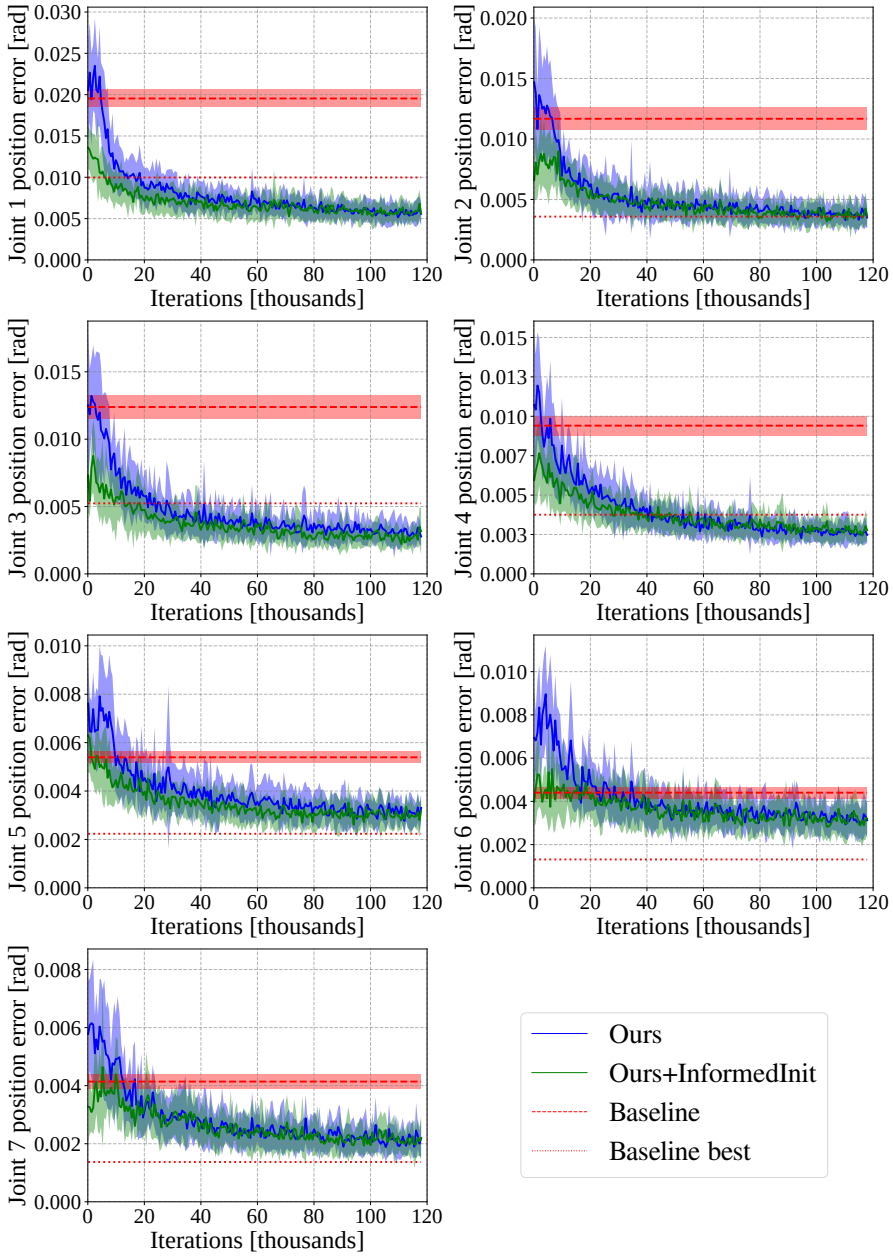


Figure 3.6: Individual joint position errors during training. Note that our method (blue and green) improves joint position tracking accuracy for every joint compared to the baseline classical controller (red). Lines: means. Shaded regions: 95% confidence intervals.

trajectory tracking accuracy of each joint. The joints are numbered from the lowest to the highest in the kinematic chain, as shown in Fig. 3.7.

The average cumulative joint position error e_q for a trajectory with M joints consisting of N points is calculated according to Eq. 3.12, summed over all points, and averaged over N. The average cumulative joint velocity error is calculated analogously. One can observe that the joint position tracking error, which we prioritize in Eq. 3.11, is

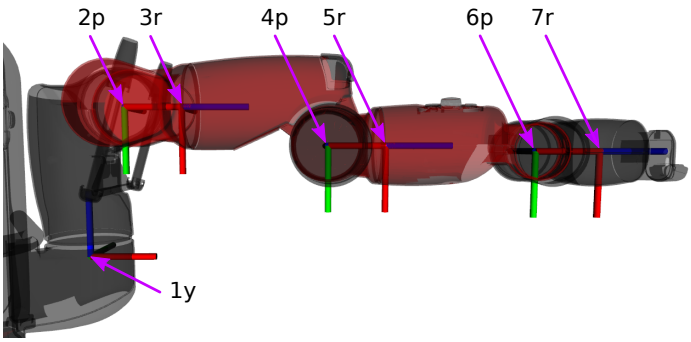


Figure 3.7: Joints of the left arm of Baxter. Letters "r", "p", "y" correspond to roll, pitch, and yaw. Joints 1 and 2: Shoulder. Joints 3 and 4: Elbow. Joints 5, 6, and 7: Wrist.

being reduced considerably during training. The joint velocity tracking error declines as well, serving as a smoothness term. SRCP inference takes $0.005 \pm 0.002\,\mathrm{s}$ on average per point, not causing noticeable delay compared to the original control loop. The entropy temperature parameter converges to $\alpha = 0.05 \pm 0.01$.

We observe that the discount factor γ significantly influences the learning process. For our case, we find $\gamma \in [0.75, 0.85]$ to be the best. A small value, such as $\gamma = 0.5$, typically results in a poor performance of the velocity component $r_{\dot{q}}$ and non-smooth trajectories. High values, like $\gamma \in [0.95, 0.99]$, result in much slower learning. We explain this observation by the fact that in our problem setting the policy does not have full control over the state and is always tied to the reference. Since the policy does not have access to the complete reference trajectory at once, accounting for the rewards that are far in the future makes the learning more challenging due to the inherent uncertainty.

There are two task-specific aspects in the presented approach. First, the low-pass filter of the actions introduces an additional delay that the model has to learn. However, removing the filtering step leads to frequently occurring jerky motions. Second, the scaling of $\Delta\boldsymbol{q}$ and $\Delta\dot{\boldsymbol{q}}$ components is determined empirically. In contrast to the filtering, these components can be removed for increased generality of the method without significantly harming the performance. However, we observe a speed-up of the learning process when these components are included in the state representation. The key hyperparameter for our approach is the maximum magnitude $\boldsymbol{a}_{\max}$ of the corrective action $\boldsymbol{a}$. While it is not trivial to choose its value, it provides an increased control over the learned policy. For example, in cases when the method is tried for the first time on a new manipulator, it is easy to safely test the approach using a small value of $\boldsymbol{a}_{\max}$.

We evaluate the learned SRCP by executing 100 unseen test trajectories and comparing the joint position tracking accuracy of the baseline + SRCP control against the vendor-provided baseline and to the open-loop method from Chapter 2. In addition, we perform the

Table 3.1: Comparison of joint position tracking errors [rad $\times 10^{-2}$].

Joint	Baseline	Open-loop	Ours
1	2.75 ± 1.83	0.75 ± 0.62	**0.52** ± **0.43**
2	1.13 ± 1.08	0.79 ± 0.67	**0.36** ± **0.27**
3	1.07 ± 0.87	0.47 ± 0.44	**0.21** ± **0.19**
4	0.66 ± 0.71	0.43 ± 0.46	**0.19** ± **0.20**
5	0.51 ± 0.35	0.37 ± 0.30	**0.27** ± **0.22**
6	0.38 ± 0.28	0.44 ± 0.35	**0.31** ± **0.27**
7	0.35 ± 0.26	0.39 ± 0.42	**0.19** ± **0.15**
$\sum$	6.87 ± 3.15	3.64 ± 1.68	**2.08** ± **0.87**

*Mean ± SD is shown.

same test with a payload of 0.9 kg to evaluate the performance in the presence of altered dynamics. Maximum payload for Baxter is 2.2 kg.

In Table 3.1 we show the average per trajectory point joint position tracking error for each joint. The joints are numbered from the start of the arm kinematic chain, as shown in Fig. 3.7. For all methods, the tracking error decreases with the ascending joint number. This happens due to the fact that the lower the joint is in the kinematic chain, the more weight it has to bear. Additionally, joints that are closer to the bottom of the chain are also affected by all the remaining joints that are higher in the chain. Combined together, these factors create complex disturbances that are making the trajectory tracking challenging.

In the conducted experiments, the shoulder joint tracking errors represent 56% of the total tracking error, and the elbow joints represent 25%, together representing 81% of the total error. The open-loop NN-based method from Chapter 2 significantly reduces the trajectory tracking error by a factor of two. Most of the improvement comes from the first three joints. It is worth noticing that the last two wrist joints have slightly higher tracking errors compared to the baseline. The method proposed in this chapter reduces the total joint position tracking error by more than a factor of three compared to the baseline and almost by a factor of two compared to the open-loop method from Chapter 2. It is worth noting that the proposed approach improves the tracking accuracy of each individual joint. We attribute this observation to the fact that the policy is trained online on the real robot and is applied in a closed-loop fashion, leveraging the advantages of live feedback.

In Table 3.2 we show joint position tracking errors for the same experiment as described above, but with a 0.9 kg payload. In the presence of an additional disturbance in the form of the weight held in the gripper, the joint position tracking accuracy decreases in all methods. The same observations from the experiment without the

Table 3.2: Comparison of joint position tracking errors with 0.9 kg payload [rad $\times 10^{-2}$].

Joint	Baseline	Open-loop	Ours
1	3.32 ± 2.32	0.95 ± 0.84	**0.47 ± 0.42**
2	2.31 ± 1.73	2.58 ± 1.49	**1.17 ± 0.62**
3	1.38 ± 1.10	0.60 ± 0.59	**0.27 ± 0.25**
4	0.79 ± 0.74	0.58 ± 0.84	**0.25 ± 0.20**
5	0.62 ± 0.53	0.46 ± 0.40	**0.38 ± 0.32**
6	0.62 ± 0.52	0.57 ± 0.54	**0.41 ± 0.37**
7	0.30 ± 0.24	0.41 ± 0.44	**0.16 ± 0.14**
$\sum$	9.36 ± 4.07	6.15 ± 3.05	**3.13 ± 1.16**

*Mean ± SD is shown.

payload apply: the error is the largest for the joints at the start of the kinematic chain. One can observe that joint 2 has the highest increase in tracking error when the payload is added. We explain this by the fact that it has a passive external spring, which further complicates its dynamics. The baseline demonstrates the worst trajectory tracking accuracy. The open-loop method reduces the trajectory tracking error, however, to a much smaller degree, compared to the experiment without the payload. That is natural when the live feedback is not utilized. At the same time, our method keeps the corresponding three- and two-fold accuracy improvement compared to the baseline and the open-loop method. Similarly, the proposed approach improves the tracking accuracy of each individual joint, as opposed to the open-loop method.

In Table 3.3 the average end-effector position tracking errors are shown for both experiments with and without the payload. Although the proposed approach focuses on improving the joint position tracking, the end-effector position tracking is improved implicitly, since a zero tracking error in the joint space corresponds to zero tracking error of the end-effector position. At the same time, such a metric is more intuitive for humans, compared to high-dimensional joint position configurations.

The overall results show the same tendencies as described in the joint position tracking error analysis. Specifically, the baseline shows

Table 3.3: Comparison of end-effector position tracking errors [cm].

	Baseline	Open-loop	Ours
No payload	3.12 ± 1.81	1.01 ± 0.61	**0.66 ± 0.42**
Payload 0.9 kg	4.35 ± 2.32	2.27 ± 1.13	**1.19 ± 0.60**

*Mean ± SD is shown.

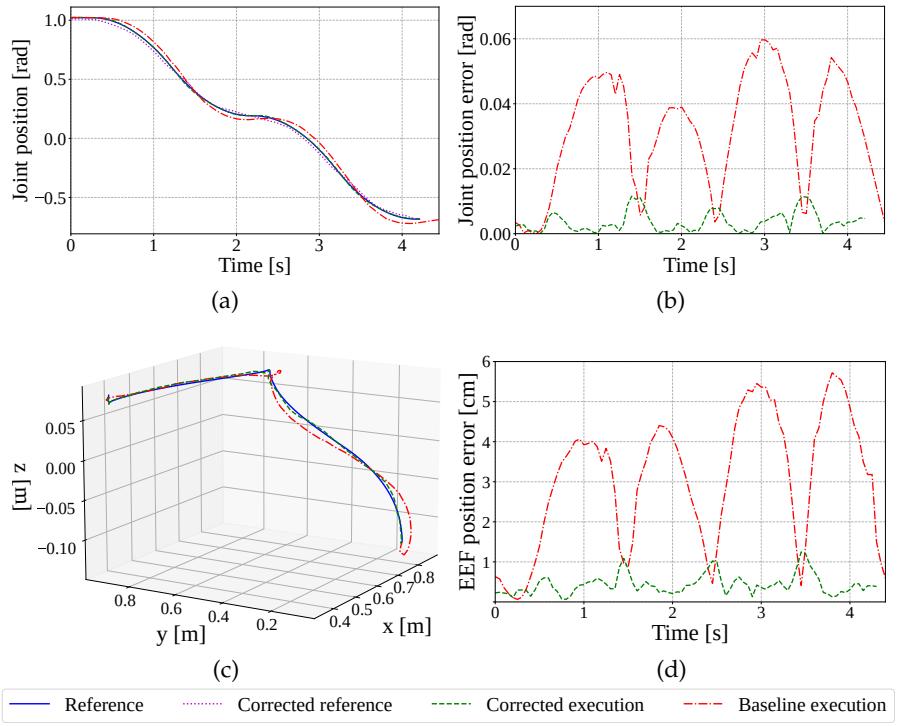


Figure 3.8: Example test trajectory. (a) Shoulder yaw position trajectory. (b) Tracking error of shoulder yaw. (c) Path of the end-effector. (d) End-effector position tracking error. While the baseline controller (red dash-dotted line) leads to significant deviations from the desired trajectory (blue line), combination with our learned controller leads to a more accurate tracking (green dashed line), achieved by reference correction (magenta dotted line).

the poorest performance, followed by the open-loop approach. Our method improves the trajectory tracking accuracy more than three times in both cases: without and with a previously unseen payload, achieving an average end-effector tracking error of 0.66 cm without a payload that is more than four times more accurate than the baseline. For reference, advanced model predictive control (MPC) controllers applied to Baxter (Rupert, Hyatt, and Killpack, 2015; Terry, Rupert, and Killpack, 2017) achieved 1-2.5 cm average steady-state error.

An example trajectory of the shoulder yaw position and the resulting end-effector path are shown in Fig. 3.8. The baseline controller frequently deviates from the reference trajectory while accelerating (seconds 0-1 and 2.5-3.5) or decelerating (seconds 1-2 and 3.5-4.5). The addition of the learned SRCP compensates for these deviations. An example trajectory from the experiment with the payload is shown in Fig. 3.9. A video of the experiment is available online[1].

1 https://www.ais.uni-bonn.de/videos/ICRA_2022_Pavlichenko

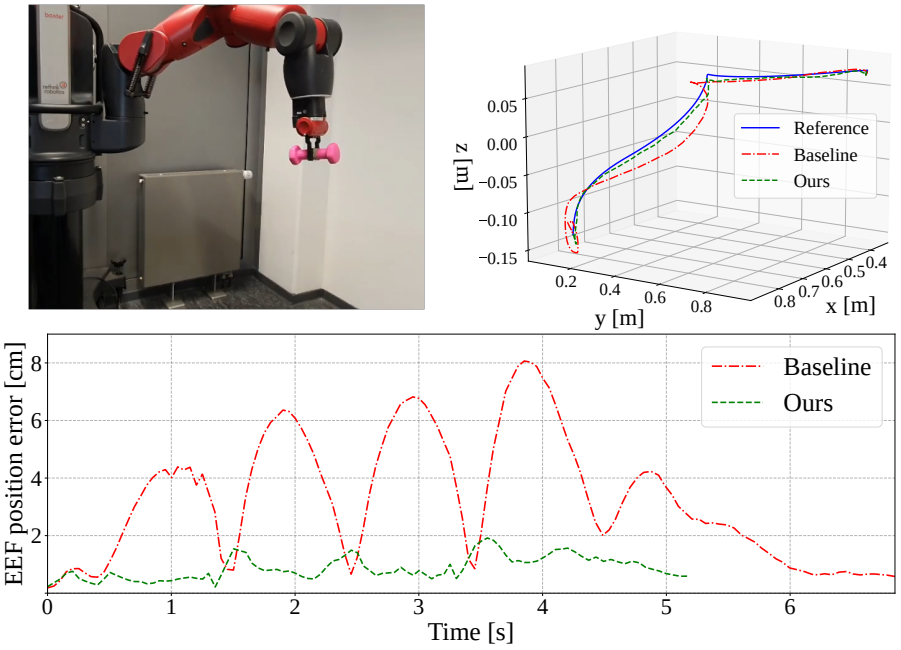


Figure 3.9: Trajectory with a payload. *Upper left:* Baxter executing the trajectory. *Upper right:* The resulting trajectory of the end-effector. *Bottom:* The resulting end-effector position tracking error. Our method (green dashed line) enables more accurate tracking of the reference (blue line), compared to the execution with the baseline (red dash-dotted line).

Overall, the conducted experiments show that the proposed approach consistently improves the performance of the policy while learning directly on the real robot. The proposed informed initialization through a learned simulation significantly reduces the tracking error at the beginning of learning. Thanks to the bounded reference correction actions, filtered by a low-pass filter, we do not observe jerky motions while training. The resulting stochastic policy improves the trajectory tracking accuracy more than three times, compared to the baseline, achieving sub-centimeter accuracy of the end-effector tracking. Finally, it demonstrates a persistent improvement of accuracy in the experiment with a previously unseen payload, suggesting that the policy learns to use live feedback from the robot, instead of simply memorizing the necessary corrections.

3.6 DISCUSSION

In this chapter we presented a model-free approach to learn a stochastic policy for improving joint trajectory tracking accuracy of a flexible-joint manipulator. The learning is performed with DRL directly on the real robot. The obtained policy serves as an additional outer-loop controller on top of the existing classical controller and provides reference correction actions.

We proposed to perform an informed initialization that uses a learned coarse dynamics model as a simulation, building on the findings from Chapter 2. The experiments indicated that the pre-training in this learned simulation significantly reduced the consequent trajectory tracking accuracy at the start of the training on the real robot. In fact, the warm-started policy outperformed the sole classical controller in trajectory tracking accuracy.

We demonstrated that the proposed method achieved consistent learning on the real 7 DoF manipulator of a Baxter robot without the need for an accurate hand-crafted simulation and a consecutive sim-to-real transfer. Our experiments indicated that the policy improved the trajectory tracking accuracy by more than a factor of three over the vendor-provided baseline controller. Dense reward formulation combined with bounded action definition allowed to learn the policy in less than two hours directly on the real robot, utilizing an ordinary computer for computations. This highlights the time-efficiency of the proposed approach.

The learned policy was general enough to demonstrate a persistent improvement when handling an unseen payload. The closed-loop nature of the method allowed for significantly more accurate trajectory tracking accuracy compared to the open-loop approach from Chapter 2.

4

DUAL-ARM TRAJECTORY OPTIMIZATION

PREFACE

This chapter is adapted from Pavlichenko, Rodriguez, Schwarz, Lenz, Periyasamy, and Behnke, 2018, previously published by IEEE and presented at the 18th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2018).

Statement of Personal Contribution

The author of this thesis substantially contributed to the following aspects of the publication (Pavlichenko, Rodriguez, Schwarz, Lenz, Periyasamy, and Behnke, 2018), including the literature survey, conception, design, and implementation of the proposed method for dual-arm trajectory optimization, the preparation and conduct of experiments and evaluation of the proposed approach, the analysis and interpretation of the experimental results, drafting the manuscript, as well as the revision and final approval of the version to be published.

The content presented in this chapter, unless otherwise stated, is the contribution of the author of this thesis.

ABSTRACT

In this chapter, we present an optimization-based method for dual-arm trajectory planning which is an extension of stochastic trajectory optimization for motion planning (STOMP). A multi-component cost function, enabling optimization with respect to multiple costs and constraints simultaneously, is introduced. We define an obstacle cost term based on the estimation of the worst-case overlap volume. Additionally, we address the closed kinematic chain constraint by splitting the chain into active and passive sub-chains. An implicit redundancy resolution for the passive sub-chain through the optimization of the initial configurations for the IK is proposed. We compare our approach to several planners and perform extensive ablation studies. Evaluations in simulation and in the real world demonstrate that our method consistently and quickly produces feasible trajectories that are optimized with respect to several costs and constraints.

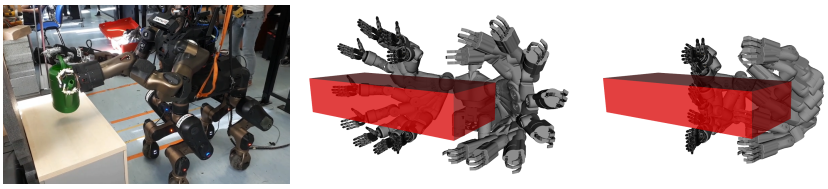


Figure 4.1: Typical scenario for dual-arm trajectory optimization. *Left:* The robot has to reach the pre-grasp pose next to the watering can with both arms moving independently. Once the can is grasped with both hands, moving the can is subject to the closed kinematic chain constraint. *Middle:* Trajectory obtained with our method that moves both arms simultaneously to a configuration above the shelf. *Right:* Trajectory obtained with our method that moves both arms back to the starting pose while satisfying the closed kinematic chain constraint.

4.1 INTRODUCTION

In previous Chapters 2 and 3, the trajectory tracking was improved by means of offline supervised learning and online DRL. The described approaches serve as outer-loop controllers and significantly increase the trajectory tracking accuracy utilizing reference correction methodology. In this chapter, we move up within the software hierarchy to address the problem of trajectory optimization. We present an efficient method for generating collision-free dual-arm trajectories from the given start to the goal configuration.

Trajectory generation is one of the essential components for autonomous robotic manipulators. The trajectory generation methods should facilitate the rapid planning of smooth, collision-free trajectories in unstructured environments. In addition, optimizing the duration of the trajectory, minimizing the torques, or adhering to the end-effector orientation constraints is often necessary. This is a challenging task that has been an active topic of research in the last decades (Ghafarian Tamizi, Yaghoubi, and Najjaran, 2023). In many scenarios, not one, but two manipulators require simultaneous trajectory generation. Such scenarios include manipulating bulky or heavy objects and using tools that require two hands to activate (Fig. 4.1). The doubling of DoF significantly increases the computational complexity. Moreover, certain dual-arm tasks necessitate satisfying additional constraints, such as the closed kinematic chain constraint. In the presence of obstacles in an unstructured environment, these aspects make generating feasible trajectories a challenging task.

In this chapter, we extend our previous work (Pavlichenko and Behnke, 2017), which is based on STOMP (Kalakrishnan et al., 2011) to facilitate efficient optimization of dual-arm trajectories. To achieve that, we adapt the existing multi-component cost function to support the dual-arm scenario. We propose a modified version of the obstacle cost function component that is based on an estimation of the worst-

case overlap volume. Such an obstacle cost function shows to be more effective for guiding the dual-arm setup out of collision regions, yielding faster convergence to the collision-free trajectories. Finally, we address the closed kinematic chain constraint for a dual-arm robot by introducing a corresponding cost term, penalizing deviations from the desired transformation between the end-effectors.

STOMP is based on exploration of the solution space by randomly sampling trajectories. Since stochastically selecting configurations that satisfy the closed kinematic chain constraint with strict margins has a near-zero probability (Kingston, Moll, and L. E. Kavraki, 2018), we introduce an IK-based approach. For that, we split the closed kinematic chain into active and passive sub-chains (Cohn et al., 2024; Xie and Amato, 2004). We refer to this approach as active-passive methodology (APM). In our case, the sub-chains correspond to the first arm and the second arm. The random sampling is performed for the trajectory of the active sub-chain, while the configurations of the passive sub-chain are obtained with IK so that the closed-chain constraint is satisfied.

This methodology, however, does not take the redundancy of the passive sub-chain into account. This may degrade the performance when applying the approach to manipulators with a high number of DoF. We address the passive arm redundancy resolution by optimizing the initial pose for the IK solver individually for each keyframe. At the same time, the closed-chain constraint cost term implicitly penalizes sampling in the regions where the IK problem can not be solved. In this way, we combine the benefits of the optimization method, such as STOMP, together with the APM for the closed kinematic chain problem.

We evaluate the proposed method quantitatively in simulation and qualitatively on the real robot. For the experiments, we use the Centauro robot (Klamt et al., 2020) that has two redundant 7 DoF arms attached to a human-like torso. We compare our method to several planning and optimization algorithms and perform ablation studies.

In summary, our main contributions are:

- a multi-component reward function for the dual-arm robot, enabling optimization with respect to six different costs,
- an obstacle cost term based on an estimation of the worst-case overlap volume, and
- an implicit redundancy resolution for an APM when optimizing under the closed kinematic chain constraint.

4.2 RELATED WORK

Generating collision-free trajectories for robotic manipulators is a fundamental task that has been studied thoroughly (H. Guo et al.,

2023). The methods for trajectory generation can be subdivided into sampling-based and optimization-based.

One of the fundamental sampling-based approaches is rapidly-exploring random tree (RRT) (Lavalle, 1998). Given the start and goal configurations, the algorithm iteratively explores the configurations adjacent to the leaves of the tree, starting at the start configuration, until the goal is reached. The main drawback of the approach lies in the nature of random exploration, resulting in a large number of states that have to be evaluated, and hence, slow convergence.

Multiple approaches were proposed to overcome this challenge (Burget, Bennewitz, and Burgard, 2016; Gammell, Srinivasa, and Barfoot, 2014; Gammell, Srinivasa, and Barfoot, 2015; James and Steven, 2000; Janson et al., 2015; Kunz and Stilman, 2014; Perez et al., 2011). Another fundamental sampling-based approach is probabilistic roadmap (PRM) (L. Kavraki et al., 1996). PRM pre-builds a map of waypoints in the configuration space, enabling quick multiple-query planning requests. However, these approaches often produce unnecessarily long or non-smooth trajectories, requiring an additional post-processing step (Geraerts and Overmars, 2007).

In contrast, optimization-based methods enable obtaining smooth trajectories that are ready to be executed. The covariant Hamiltonian optimization for motion planning (CHOMP) (Ratliff et al., 2009) employs a covariant gradient approach, which relies on the gradient of the cost function. This concept is similar to the elastic bands planning (Brock and Khatib, 2002), where repelling forces push the trajectory away from the obstacles. CHOMP quickly converges to a locally optimal trajectory using a signed distance field (SDF) to represent the environment, enabling gradient calculation even for parts of the trajectory that are not collision-free. However, like many gradient-based techniques, CHOMP is prone to getting trapped in local minima.

T-CHOMP (Byravan et al., 2014) is an extension of CHOMP that adds an extra dimension for the time, enabling optimization in both space and time. The authors note that the method is highly sensitive to parameter settings and may produce an infeasible "collision-free" solution where the robot moves slowly through the obstacles if not tuned properly.

STOMP (Kalakrishnan et al., 2011) builds on the environment representation used in CHOMP but differs by employing a sampling technique for cost minimization instead of relying on the gradient of the cost function. This approach allows the use of non-differentiable cost functions and reduces the likelihood of getting stuck in local minima. A local multiresolution approach for STOMP is introduced by Steffens, Nieuwenhuisen, and Behnke, 2016. This method plans the initial part of the trajectory with high resolution, while the segments of the trajectory that are further away in time are planned with lower resolution. This strategy reduces the computation time and enables the

application of the method in dynamic environments. However, similar to the original STOMP method, the total duration of the trajectory remains fixed.

Schulman, Ho, et al., 2013 introduce TrajOpt, a method for integrating collision avoidance into trajectory optimization for robotic motion planning. The approach uses sequential convex optimization with a hinge loss penalty for collisions, adjusting penalties as needed, and implements a continuous-time no-collision constraint to efficiently solve complex motion planning problems, even in environments with thin obstacles. It has been demonstrated to perform well for problems with many DoF, such as dual-arm mobile manipulation with PR2 and whole-body motion planning for the Atlas robot.

Some approaches focus on combining methods from the two families, attempting to combine exploration from sampling-based methods and exploitation from optimization-based methods. For example, batch informed trees (BIT*) (Gammell, Srinivasa, and Barfoot, 2015). Regionally-accelerated BIT* (Choudhury et al., 2016) extends BIT* by using an optimizer to adjust infeasible edges and move them out of collision. This method often results in a lower solution cost compared to BIT*. However, the frequent use of an optimizer can be computationally expensive, leading to longer runtimes.

Alwala and Mukadam, 2021 implement a similar idea using a roadmap planner to interleave edge creation and optimization steps. Another methodology in that direction builds on the idea of the planner proposing diverse plans that are consequently sent to the optimizer (Kuntz and Alterovitz, 2020; Xanthidis et al., 2020). Such plans can be efficiently generated using sparse roadmaps (Orthey, Frész, and Toussaint, 2020).

Although all previously mentioned methods can be applied to dual-arm setups directly or with slight modifications, there are numerous approaches designed specifically to tackle dual-arm planning problems. Szynkiewicz and Błaszczyk, 2011 introduce an optimization-based method for path planning for closed kinematic chain robotic systems. The problem is formulated as a function minimization task with both equality and inequality constraints expressed in terms of the joint variables. Vahrenkamp et al., 2009 propose two approaches for dual-arm planning: J^+ and IK-RRT. While the J^+ method does not require an IK solver, the IK-RRT approach demonstrates superior performance in both single and dual-arm tasks.

Alternatively, Cohen, Chitta, and Likhachev, 2014 introduce a heuristic-based approach that constructs a manipulation lattice graph and utilizes an informative heuristic. Although the algorithm's success is heavily dependent on the effectiveness of the heuristic, it demonstrates strong performance compared to several sampling-based planners. Shi et al., 2022 extend RRT by biasing the exploration process through child node selection that minimizes a cost function.

Manipulator number one only avoids collisions with the environment, while manipulator number two avoids both the environment and the self-collisions with moving manipulator number one. Although this extension can be effectively applied to the dual-arm systems, it still suffers from the main drawbacks of RRT.

Byrne, Naeem, and Ferguson, 2015 propose a method that integrates goal configuration sampling, sub-goal selection, and artificial potential fields (APF) motion planning. This approach has been demonstrated to enhance APF performance in both independent and cooperative dual-arm manipulation tasks. Jang et al., 2022 propose a PRM-based approach. The approach focuses on improving the sampling efficiency of the new connected nodes under the closed-chain constraint. The planner assumes an object being held by a multi-arm system and samples feasible configurations by directly sampling the object poses and solving the IK for each arm. The drawback of the approach is that an IK problem has to be solved for each manipulator.

Cohn et al., 2024 propose to leverage the analytic IK to parameterize the configuration space, obtaining a lower-dimensional representation. The parameterization can be applied to a broad range of existing planners. Similar to our method, a subdivision into active and passive kinematic chains is performed. In contrast, we propose a methodology that does not rely on the availability of an analytical IK solution and instead optimizes the initial poses for an underlying IK solver to implicitly resolve the redundancy of the passive sub-chain. Overall, our approach stands out thanks to the ability to optimize trajectories with respect to multiple constraints, while allowing for influencing the qualitative properties of the obtained solutions by adjusting the cost importance weights.

4.3 BACKGROUND

In this chapter, we present our method for dual-arm trajectory optimization. It builds on top of the approach from our Master Thesis (Pavlichenko, 2016), results of which were also published in (Pavlichenko and Behnke, 2017). In this section we briefly introduce the preliminaries for our method: first, the original STOMP (Kalakrishnan et al., 2011) followed by its extension STOMP-New (Pavlichenko and Behnke, 2017).

4.3.1 *STOMP*

In STOMP (Kalakrishnan et al., 2011), the planning task is framed as an optimization problem, with the goal of finding a trajectory that minimizes the cost defined by a given cost function. The input of STOMP is an initial trajectory $\boldsymbol{q}$ composed of N keyframes $\boldsymbol{q}_i \in \mathbb{R}^M$ in joint space with M joints: $\boldsymbol{q} = [\boldsymbol{q}_1 \dots \boldsymbol{q}_N]$. Each keyframe is

a configuration in joint space: $\boldsymbol{q}_i = [q_{i,1} \dots q_{i,M}]$, where $q_{i,j}$ is a joint angle of joint j at keyframe i. The keyframes are equally spaced in time, discretizing a predefined fixed duration T. A common naïve initial trajectory is the linear interpolation between the given start and goal configurations $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$. Throughout the optimization process, these start and goal configurations remain unchanged. The output of STOMP is an optimized trajectory, with the optimization problem being formulated as follows:

$$\min_{\tilde{\boldsymbol{q}}} \mathbb{E}\left[\sum_{i=1}^{N} \mathcal{C}(\tilde{\boldsymbol{q}}_i) + \frac{1}{2}\tilde{\boldsymbol{q}}^\top \boldsymbol{R}\tilde{\boldsymbol{q}}\right], \tag{4.1}$$

where $\tilde{\boldsymbol{q}} = \mathcal{N}(\boldsymbol{q}, \Sigma)$ is a noisy joint parameter vector, given that $\boldsymbol{q}$ is the mean and Σ is the covariance. $\mathcal{C}(\tilde{\boldsymbol{q}}_i)$ is a state cost function that includes obstacle costs, torque costs, and constraint costs. Each state $\tilde{\boldsymbol{q}}_i$ of the trajectory $\tilde{\boldsymbol{q}}$ has a cost assigned by this cost function. The term $\tilde{\boldsymbol{q}}^\top \boldsymbol{R}\tilde{\boldsymbol{q}}$ represents the sum of squared accelerations along the trajectory, which are calculated using a finite differencing matrix.

STOMP iteratively samples noisy trajectories, evaluates them with a cost function, and updates the noise distribution parameters, moving in the direction of the cost gradient. In this way, STOMP is able to find locally optimal trajectories, oftentimes escaping the local minima.

4.3.2 *STOMP-New*

The approach described in this chapter is based on our earlier work: STOMP-New (Pavlichenko and Behnke, 2017), which is the extension of STOMP. The main differences to the original STOMP are the following:

- Multi-component cost function, evaluating transitions between keyframes, as opposed to evaluating single keyframes,
- adaptive collision checking density, and
- two-phased optimization, first considering only the kinematic costs and in the second phase considering kinematic and dynamic costs at the same time.

In the original STOMP, the cost of a trajectory is defined as a sum of costs of individual keyframes $\boldsymbol{q}_i$. It is not trivial to estimate the length of the final solution in advance. Thus, one has to pre-define the fixed number of keyframes N, ensuring adequate collision checking density. If N is too small, the algorithm may position keyframes so that they skip over an obstacle, keeping the cost small and leading to a false-positive solution which is infeasible. That is why typically a large number of keyframes is chosen, such as $N = 100$. However, when moving through the obstacle-free regions, such dense collision

checking is unnecessary. Moreover, noise distribution updates for a larger number of keyframes are computationally more expensive.

In STOMP-New we propose to alleviate these drawbacks by introducing a cost function that evaluates transitions between the two consecutive keyframes $\boldsymbol{q}_i$ and $\boldsymbol{q}_{i+1}$. With this change, one can now safely choose a much smaller number of keyframes for the trajectory, such as $N = 10$. The collision checking density is defined online for each specific transition between the keyframes. The density is inverse-proportional to the distance to the closest obstacle at the start and the end of the transition. The multi-component transition-based cost function is defined as:

$$\begin{aligned} \mathcal{C}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) =& \mathcal{C}_\mathrm{o}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) + \mathcal{C}_\mathrm{l}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) + \mathcal{C}_\mathrm{c}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) \\ &+ \mathcal{C}_\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) + \mathcal{C}_\mathrm{t}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}), \end{aligned} \tag{4.2}$$

where $\mathcal{C}_\mathrm{o}$ is an obstacle cost that penalizes proximity to the obstacles, $\mathcal{C}_\mathrm{l}$ is a joint limit cost that penalizes violations of joint limits, $\mathcal{C}_\mathrm{c}$ is a constraint cost that penalizes violations of any custom constraints on the end-effector position or orientation, $\mathcal{C}_\mathrm{d}$ is a duration cost that penalizes long duration, and $\mathcal{C}_\mathrm{t}$ is a torque cost that penalizes high torques. Each cost component function $\mathcal{C}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) \in [0, 1]$, enabling straightforward relative importance scaling by adding an importance weight to each cost component. In this manner, one can influence the qualitative properties of the obtained trajectories. Note that costs for violations of constraints of each component, such as a collision, are much greater than 1 and make a trajectory with such a violation infeasible.

Finally, the optimization is performed in two stages. At the first stage, only the kinematic cost components $\mathcal{C}_\mathrm{o}$, $\mathcal{C}_\mathrm{l}$ and $\mathcal{C}_\mathrm{c}$ are considered. Once a feasible trajectory is found, the optimization continues through the second stage with the full cost, including both kinematic and dynamic cost components. In this way, the algorithm first aims at leaving strictly infeasible collision regions, speeding up the optimization process. Altogether, the described changes significantly improve the runtime of the algorithm and provide control over the qualitative properties of the obtained trajectories.

4.4 METHOD

Both STOMP and STOMP-New have a complexity linear in the number of DoF, and the optimization is taking place in the joint space. Thus, it is straightforward to add the second arm into the optimization problem. Moreover, the arms may have different kinematic structures and numbers of joints. Note that although in this chapter we focus on the dual-arm scenario, the proposed approach is generic and can be directly applied to an arbitrary number of arms without fundamental modifications.

The state cost function has the same structure as in the base STOMP-New method (Eq. 4.2):

$$\begin{aligned}\mathcal{C}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) =& \mathcal{C}_\mathrm{o}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) + \mathcal{C}_\mathrm{l}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) + \mathcal{C}_\mathrm{c}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) \\ &+ \mathcal{C}_\mathrm{d}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) + \mathcal{C}_\mathrm{t}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) + \mathcal{C}_\mathrm{cc}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}),\end{aligned} \tag{4.3}$$

where the first five terms are obstacle, joint limits, end-effector orientation constraints, duration, and torque costs, as discussed in Section 4.3.2. The new term $\mathcal{C}_\mathrm{cc}$ is an additional sixth term that penalizes violations of the closed kinematic chain constraint. This allows optimizing for coordinated arm movement, such as moving a bulky object that is held with both hands. The cost terms can be split into two categories: joint-related: $\mathcal{C}_\mathrm{o}$, $\mathcal{C}_\mathrm{l}$, $\mathcal{C}_\mathrm{d}$, $\mathcal{C}_\mathrm{t}$ and end-effector-related: $\mathcal{C}_\mathrm{c}$, $\mathcal{C}_\mathrm{cc}$.

The joint-related terms do not have to be changed for the dual-arm scenario. Keeping the duration term $\mathcal{C}_\mathrm{d}$ identical to the STOMP-New constitutes that both arms will have the same duration within their individual trajectories, independent of the lengths of the individual paths. This comes with two advantages: first, only one additional dimension is required for both arms; second, self-collision checking is not affected by the relative velocities of the arms and can be performed directly since both arms will move through the corresponding keyframes at the same timesteps. The drawback of such a duration cost design is that in the case of significantly different individual trajectory lengths, the arm with the shorter path length will be forced to move with a significantly lower velocity.

The terms related to end-effectors require modifications. The end-effector orientation constraint term $\mathcal{C}_\mathrm{c}$ is applied separately to both end-effectors:

$$\mathcal{C}_\mathrm{c}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) = \frac{1}{2}\mathcal{C}_\mathrm{c}(\boldsymbol{q}_i^\mathrm{l}, \boldsymbol{q}_{i+1}^\mathrm{l}) + \frac{1}{2}\mathcal{C}_\mathrm{c}(\boldsymbol{q}_i^\mathrm{r}, \boldsymbol{q}_{i+1}^\mathrm{r}), \tag{4.4}$$

where $\boldsymbol{q}_i^\mathrm{l}$ is a joint configuration for the left arm and $\boldsymbol{q}_i^\mathrm{r}$ is a joint configuration for the right arm respectively.

Subsequently, we describe two fundamental modifications introduced in this chapter. First, we modify the obstacle cost term $\mathcal{C}_\mathrm{o}$, grounding it on the estimation of the worst-case collision overlap volume, increasing the convergence speed. Second, we define the new closed kinematic chain constraint term $\mathcal{C}_\mathrm{cc}$, increasing the range of dual-arm motion generation problems that can be solved with the proposed approach and an IK-based methodology with redundancy resolution.

4.4.1 *Obstacle Cost*

We assume that the environment is static, with only the robot arms moving. Under this assumption, it is possible to represent the environ-

ment as a SDF, which is obtained with a signed Euclidean distance transform (Kalakrishnan et al., 2011). The moving part of the robot body is approximated with a set of overlapping spheres $\boldsymbol{B}$. Such environment and robot representation allows for very efficient collision checking. Moreover, it provides signed obstacle proximity information for any particular sphere $b \in \boldsymbol{B}$.

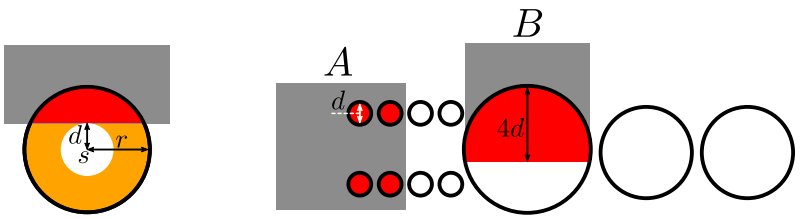
This enables formulating a smoother cost function so that it is efficient at steering the optimization towards the collision-free regions. We adopt the representation of the static components of the environment with two SDFs: one for the external objects and one for the static part of the robot body (Pavlichenko and Behnke, 2017). Such representation allows defining two independent obstacle proximity margins: high for the external objects and smaller for the robot's static part. This is motivated by the assumption that the poses of the robot's body parts are much more accurate compared to those of the external objects.

In our previous work (Pavlichenko and Behnke, 2017), the minimal distance from the collision model to the obstacles is used for obstacle cost computation. While that works well for one arm, such obstacle cost representation leads to a very complex gradient profile in the dual-arm scenario, slowing down convergence. For example, when one arm has a much greater collision depth than the other arm, that disguises the collision with lower depth from the algorithm.

On the other hand, in the original STOMP, the obstacle cost is represented as a sum of collision depths across all arm approximation spheres $b \in \boldsymbol{B}$. While such obstacle cost certainly represents the severity of a collision better, all spheres have equal contribution. This results in very coarse obstacle costs in cases when spheres are of different sizes and are distributed unequally along an arm. In fact, most arms have to be represented with spheres of different sizes to adequately represent the complex geometry of a gripper. For the same reason, a non-uniform sphere density is present as well, with more spheres typically located in the region of the gripper. In these cases, a direct sum of collision depths of individual spheres results in much higher costs in regions where spheres are placed with the highest density. This often creates multiple local minima, slowing down the convergence of the algorithm.

In contrast, we propose an obstacle cost based on estimation of the worst-case overlap volume. The severity of a collision is represented through an estimate of the worst-case colliding volume, as opposed to a direct sum of collision depths (Fig. 4.2b). Given a sphere b_i with radius r_{b_i} and its obstacle penetration depth $d_\mathrm{p} : r_{b_i} > d_\mathrm{p} > 0$ obtained from the environment SDF, the worst-case overlap volume $v^\mathrm{c}_{b_i}$ of the sphere b_i is a volume of a spherical shell (Fig. 4.2a):

$$v^\mathrm{c}_{b_i} = \frac{4}{3}\pi(r^3_{b_i} - (r_{b_i} - d_\mathrm{p})^3). \tag{4.5}$$



(a) (b)

Figure 4.2: Estimation of the worst-case overlap volume. Projection from 3D onto two-dimensional (2D) plane is shown. (a) Given a sphere with center s and radius r, the distance d to the closest obstacle (gray rectangle) is queried from the SDF. Given only the distance d, it is not possible to calculate the actual overlap volume (red). Instead, we estimate the worst-case overlap volume, which is a spherical shell (red and orange combined). (b) Manipulator approximated with a set of spheres colliding with the two obstacles (gray rectangles A and B). The sum of collision depths of spheres colliding with A equals the collision depth of the sphere colliding with B. However, the corresponding overlap volume is much greater in B than in A. Obstacle cost based on the worst-case overlap volume reflects this aspect.

An estimate of the total overlap volume of the arms in case of a self-collision can be done directly by iteratively computing $v^{\mathrm{c}}_{b_i}$ for each colliding sphere b_i and summing up the values. Finally, given an estimate of the total volume of the robot collision approximation, it is straightforward to compute the collision severity factor η:

$$\eta = \frac{\sum_{b \in \boldsymbol{B}} v^{\mathrm{c}}_{b_i}}{\sum_{b \in \boldsymbol{B}} v_{b_i}}. \tag{4.6}$$

This value represents an estimate of the portion of the robot body that is colliding with the environment. We utilize η in the obstacle cost computation to obtain continuous and smooth collision penalties:

$$\mathcal{C}_{\mathrm{o}}(\boldsymbol{q}_i) = \begin{cases} C_{\mathrm{o}} + \eta C_{\mathrm{o}}, & \text{if } d_{\mathrm{obst}} \leq d_{\mathrm{min}} \\ 0, & \text{if } d_{\mathrm{obst}} \geq d_{\mathrm{max}} \\ \lambda_o \cdot \left(1 - \frac{d_{\mathrm{obst}} - d_{\mathrm{min}}}{d_{\mathrm{max}} - d_{\mathrm{min}}}\right), & \text{otherwise} \end{cases}, \tag{4.7}$$

where $\lambda_o \in [0, 1]$ is the importance weight for the obstacle costs, d_{min} is a minimum acceptable distance to the obstacles, and d_{max} is a maximum distance to the obstacles that should be considered. d_{obst} is the distance to the nearest obstacle. $C_{\mathrm{o}} \gg 1$ is a predefined constant that ensures that infeasible configurations have very high costs. In case a configuration $\boldsymbol{q}_i$ is feasible and the distance to the nearest obstacle falls in the interval $(d_{\mathrm{min}}, d_{\mathrm{max}}]$, the cost is within the interval $[0, 1]$.

4.4.2 *Closed Kinematic Chain Constraint*

Given a start and goal configuration $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{\text{goal}}$, defining configurations of two arms in joint space, we compute the desired transformation $\boldsymbol{T}_{\text{desired}} = \boldsymbol{T}_{1\to 2}$ between the two 6D end-effector poses $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$, both at start and goal configurations. Since start and goal configurations stay fixed during the optimization, $\boldsymbol{T}_{\text{desired}}$ should be the same at both the start and the goal in order to optimize under the closed kinematic chain constraint. The new cost term $\mathcal{C}_{\text{cc}}$ for the closed-chain constraint is formulated as:

$$\mathcal{C}_{\text{cc}}(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) = \tfrac{1}{2} \max_j \mathcal{C}_{\text{ct}}(\boldsymbol{q}_j) + \tfrac{1}{2} \max_j \mathcal{C}_{\text{cr}}(\boldsymbol{q}_j), j \in \{1, \ldots, k\}, \tag{4.8}$$

where $\mathcal{C}_{\text{ct}}(\cdot)$ penalizes deviating from $\boldsymbol{T}_{\text{desired}}$ within the translation component between the end-effectors and $\mathcal{C}_{\text{cr}}(\cdot)$ penalizes deviating from $\boldsymbol{T}_{\text{desired}}$ within the corresponding rotation component. The configurations $\boldsymbol{q}_j$ are the intermediate equidistant configurations for the transition between $\boldsymbol{q}_i$ and $\boldsymbol{q}_{i+1}$ sampled to cover the transition with the desired density, defined by the distance traveled by the end-effector. In this work we use the density of one check per 1 cm traveled by the end-effector. This density defines the number of intermediate configurations k for each transition. The translation cost component $\mathcal{C}_{\text{ct}}$ is defined as:

$$\mathcal{C}_{\text{ct}}(\boldsymbol{q}_j) = \begin{cases} C_{\text{ct}} + C_{\text{ct}} \cdot \Delta d & \text{if } \Delta d \geq \Delta d_{\max} \\ \frac{\Delta d}{\Delta d_{\max}}, & \text{otherwise} \end{cases}, \tag{4.9}$$

where Δd is an L^2 norm of the translation component of the 6D transformation from the current 6D transformation between the end-effectors $\boldsymbol{T}_{\text{current}}$ and the desired transformation $\boldsymbol{T}_{\text{desired}}$. Finally, $\Delta d_{\max}$ is the maximum allowed deviation of the translation component, and $C_{\text{ct}} \gg 1$ is a predefined constant. Thus, $\mathcal{C}_{\text{ct}} \in [0, 1]$ if the deviation of the translation is below the threshold and $\mathcal{C}_{\text{ct}} \gg 1$ otherwise.

Similarly, we define the term $\mathcal{C}_{\text{cr}}(\cdot)$ for penalizing deviations in the rotation component. Given the transformation between $\boldsymbol{T}_{\text{current}}$ and $\boldsymbol{T}_{\text{desired}}$, we compute the corresponding angular difference $\Delta\alpha$ and compute the $\mathcal{C}_{\text{cr}}$ term:

$$\mathcal{C}_{\text{cr}}(\boldsymbol{q}_j) = \begin{cases} C_{\text{cr}} + C_{\text{cr}} \cdot \Delta\alpha & \text{if } \Delta\alpha \geq \Delta\alpha_{\max} \\ \frac{\Delta\alpha}{\Delta\alpha_{\max}}, & \text{otherwise} \end{cases}, \tag{4.10}$$

where $\Delta\alpha_{\max}$ is the maximum allowed angular difference, and $C_{\text{cr}} \gg 1$ is a predefined constant.

Incorporating the cost term $\mathcal{C}_{cc}$ into the cost function enables optimizing trajectories under the kinematic chain closure constraint. However, the probability of randomly sampling a configuration that satisfies the closure constraint is approaching zero (Kingston, Moll, and L. E. Kavraki, 2018). In order to increase the probability and, hence, the convergence speed, one can increase the values of maximum allowed translation and rotation deviations $\Delta d_{\max}$ and $\Delta \alpha_{\max}$.

While this allows the algorithm to converge quicker, the obtained trajectories are not strictly satisfying the closed-chain constraint. Thus, an additional post-processing step to satisfy the constraint is required. Such an approach fails in cases when desired end-effector poses that satisfy the constraint are not reachable due to joint limits or obstacles. Moreover, such a post-processing step can potentially deteriorate other properties of the trajectory obtained during the optimization. For example, maintained distance to the obstacles or trajectory duration.

To overcome these challenges, we further extend our approach with an APM. There, the kinematic chain is split into two sub-chains: active and passive. For the dual-arm setting, the division is already existing naturally and the sub-chains are arm number one and arm number two. The optimizer performs an optimization for the trajectory of an active chain, while the trajectory of the passive chain is determined with the IK, given the trajectory of the active chain, such that the constraint is satisfied. In this way, trajectories satisfying the closed-chain constraint can be reliably obtained by sampling random trajectories for the active chain. The drawback of such an approach is that the computational time is significantly affected by the IK computation, which is expensive in the case of high-DoF manipulators with redundancy. Furthermore, it is challenging to find locally optimal configurations for the passive chain from a set of viable IK solutions.

To address the challenges introduced by utilizing an IK solver, we enable the optimization algorithm to indirectly steer the IK solutions by having control over the initial configuration for the underlying IK solver. In such a manner, we allow the method to influence the resulting IK solution for each specific keyframe online. Since one of the arms is passive and does not require the method to directly optimize its configurations, its place is taken by the initial configuration for the IK solver. In this way, the number of the DoF on which the optimization is conducted remains the same.

Even though the configurations for the passive chain are obtained with IK, the introduced closed-chain constraint cost term $\mathcal{C}_{cc}$ remains useful when the IK runs out of allotted time or does not find an accurate solution. In these cases this term produces a high cost, which subsequently influences the starting position for the IK, chosen by the optimizer. This in turn, can lead to an IK solution with lower cost. It also implicitly penalizes long IK computation times in cases when the solution is not found and the IK returns an approximate solution that

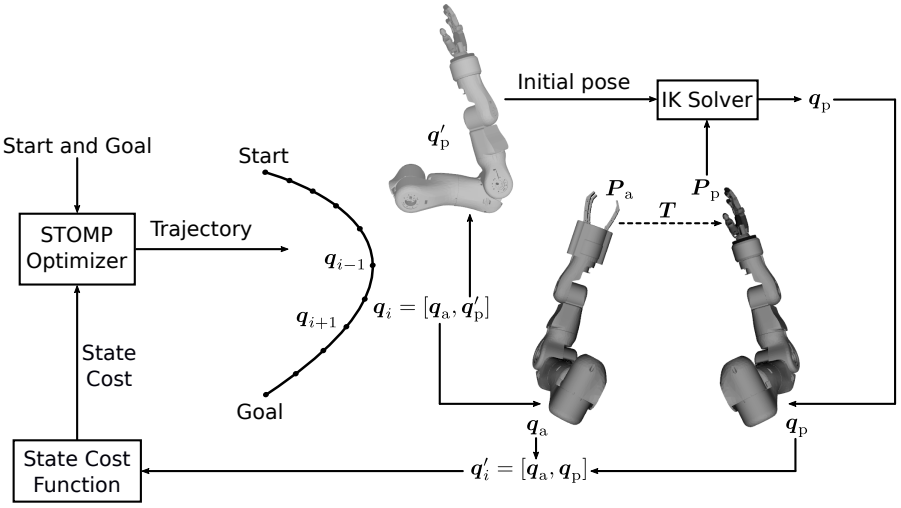


Figure 4.3: Optimization under the closed kinematic chain constraint with APM with an implicit redundancy resolution. STOMP optimizes a trajectory consisting of keyframes $\boldsymbol{q}_i$ in joint space. The first half $\boldsymbol{q}_a$ of a keyframe $\boldsymbol{q}_i$ is a configuration of the active arm, and $\boldsymbol{P}_a$ is a 6D pose of the corresponding end-effector. Given a desired transformation $\boldsymbol{T}$ of the closed-chain constraint, 6D pose $\boldsymbol{P}_p$ of the passive arm is obtained. The second half $\boldsymbol{q'}_p$ of the keyframe is an initial pose for the IK solver. Given $\boldsymbol{P}_p$ and $\boldsymbol{q'}_p$, the IK produces a joint configuration $\boldsymbol{q}_p$ for the passive arm, satisfying the constraint. Together with the obtained $\boldsymbol{q}_p$, the already known $\boldsymbol{q}_a$ forms the configuration $\boldsymbol{q'}_i$ that is evaluated with the state cost function. Optimizing for $\boldsymbol{q'}_p$ implicitly resolves the redundancy.

also yields high $\mathcal{C}_{cc}$ cost. Finally, other cost components, such as the obstacle cost term, also implicitly steer the IK solutions towards the regions with lower costs. The diagram of this approach is shown in Fig. 4.3.

The smoothness of the passive arm trajectory is implicitly imposed first by the fact that the passive arm trajectory is directly linked to the trajectory of the active arm, the smoothness of which is achieved through STOMP smoothness cost. Additionally, the initial configurations for the IK sampled by STOMP that are also subject to smoothness cost contribute to the overall coherency of the consequent arm configurations obtained through the IK. Finally, each transition of the passive arm from configuration $\boldsymbol{q}_i$ to $\boldsymbol{q}_{i+1}$ is evaluated by computing closed-chain constraint cost and checking the intermediate configurations for joint velocity and acceleration limits violations with desired density. This ensures that the transition is feasible. By combining STOMP and APM for handling the closed kinematic chain constraint, we aim to combine the benefits of both approaches.

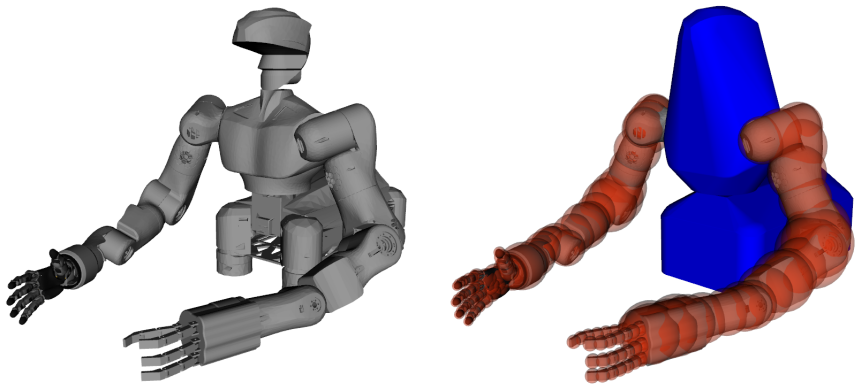


Figure 4.4: Model of a dual-arm Centauro robot. *Left:* Detailed model. *Right:* Collision model that is split into static and dynamic parts. The dynamic part (orange) is approximated with a set of spheres. The static part (blue) is approximated with a convex hull of the torso and the base.

4.5 EVALUATION

To evaluate the proposed approach, we perform quantitative experiments in simulation and qualitative experiments on the real-world system. First, we evaluate the runtime, success rate, as well as obtained trajectory lengths for an unconstrained problem. Second, we evaluate the same metrics while optimizing under the closed kinematic chain constraint.

4.5.1 *Setup*

We perform the evaluation on the Centauro robot (Klamt et al., 2020). It is a centaur-like mobile manipulation robot with a human-like torso and two 7 DoF arms. Each arm has a different hand. The left arm has a robust Heri hand (Ren et al., 2017) with four fingers that features robustness and capability of handling considerable payloads. The right arm has a dexterous human-like Schunk SVH hand with five fingers capable of dexterous manipulation. Both arms collision models are represented with a set of overlapping spheres: 18 for each arm, 37 for the left hand, and 39 for the right hand, resulting in a total of 112 spheres, as shown in Fig. 4.4. The collision model is split into two parts: dynamic, which includes the arms, and static, which includes the torso.

The static part is represented with a separate SDF. That is beneficial since a separate collision safety margin can be defined for self-collisions with the torso and the base. Typically, this safety margin can be lower compared to the safety margin for collisions with the environment. That is because the poses of the robot body parts are

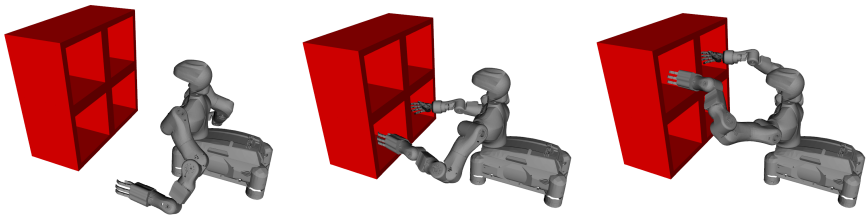


Figure 4.5: Shelf experiment environment. From left to right, three configurations used in the shelf experiment: Neutral, hands in the bottom row, hands in the top row.

known with higher confidence compared to the poses of external objects in the environment.

All evaluated methods use this collision model. For our method, we used the SDF discretization of 1 cm and collision model padding of 1 cm to compensate for the inaccuracies in the SDF due to the discretization. All experiments are performed on a laptop with an Intel Core i7-6700HQ 2.60 GHz CPU, 16 GB of RAM, and 64-bit Kubuntu 18.04 OS with robot operating system (ROS) Melodic. Experiments with TrajOpt are performed on the same laptop using Kubuntu 20.04 OS with ROS Noetic due to the software dependencies. All evaluated algorithms run on a single CPU core.

4.5.2 *Unconstrained Scenario*

We conduct simulation experiments to evaluate our method's ability to produce feasible trajectories for unconstrained problems. The experiments focus on optimizing the trajectories of two independent arms in a shelf environment. The robot is positioned in front of a shelf with four $40 \times 40 \times 40$ cm cells with walls that are 4 cm thick. The arms can be in one of the three configurations: neutral outside the shelf cells; inside the bottom row of the shelf cells; or inside the top row of the shelf cells.

The task is to obtain feasible trajectories for both arms, traversing between all combinations of the three configurations. This results in a total of six planning problems. In addition, we perform the task with two difficulties: "Easy" and "Hard." In "Hard" difficulty, the shelf is moved 10 cm closer towards the robot. In this case, the configurations with the hands inside the cells are immersed deeper. Such modification makes the task more challenging. The environment is shown in Fig. 4.5.

We compare our approach to two methods from open motion planning library (OMPL) (Şucan, Moll, and L. E. Kavraki, 2012): RRTConnect (James and Steven, 2000) and kinodynamic planning by interior-exterior cell exploration (KPIECE) (Şucan and L. E. Kavraki, 2010). These algorithms previously demonstrated a consistent performance (Meijer, Lei, and Wisse, 2017). Furthermore, we also compare

our approach to Tesseract implementation[1] of TrajOpt (Schulman, Ho, et al., 2013).

In addition, we conduct an ablation study of the proposed obstacle cost component. First, with the obstacle cost function from the original STOMP (Kalakrishnan et al., 2011). We refer to this method as Ours. Second, with the proposed obstacle cost component based on the estimation of the worst-case overlap volume, as described in Section 4.4.1. We refer to this method as Ours-OV (Ours with overlap volume). Each task is attempted 100 times, resulting in a total of 600 runs per difficulty setting for each method. We fine-tune the parameters for RRTConnect and KPIECE using the grid search. For RRTConnect the obtained parameters are:

1. *range* = 3.3,
2. *goal bias* = 0.4.

For KPIECE the obtained parameters are:

1. *range* = 1.8,
2. *goal bias* = 0.4,
3. *border fraction* = 0.5,
4. *failed expansion score factor* = 0.7,
5. *minimum valid path fraction* = 0.2.

Both TrajOpt and our method performed well out of the box with the default parameters and did not have to be fine-tuned for the specific task in this experiment. Each method is given 60 s for an attempt. If this duration is exceeded, the attempt is considered as failed. If the final trajectory violates joint limits or collides with the environment or the robot itself, the attempt is considered as failed. We record the success rate and runtime for each method, as well as the resulting path lengths in joint space and Cartesian space for the end-effectors.

In Table 4.1, success rates and average runtimes are shown. We use only the successful attempts in the runtime calculation. In the case of the "Easy" difficulty level, almost all methods achieve a success rate of 1.0. Only RRTConnect fails several times, achieving a 0.98 success rate. All methods demonstrate relatively low runtime, with RRTConnect achieving the fastest average runtime of 1.1 s. Our method with both obstacle cost variations achieves a similar average runtime of 1.52 s and a success rate of 1.0. The SDF computation for our method takes 0.32 ± 0.012 s on average in this experiment and is incorporated in the presented runtime. It is worth noting that our method achieves lowest maximum runtime and lowest standard deviation.

1 https://github.com/tesseract-robotics/trajopt

Table 4.1: Comparison of success rates and average runtimes in the shelf experiment.

Task	Method	Success rate	Runtime [s]			
			Mean	SD	Min	Max
Easy	RRTConnect	0.98	**1.1**	1.25	0.07	6.62
	KPIECE	**1.0**	1.98	2.61	**0.05**	21.2
	TrajOpt	**1.0**	2.27	1.69	0.19	5.0
	Ours	**1.0**	1.51	**0.28**	1.02	2.91
	Ours-OV	**1.0**	1.52	0.31	1.0	**2.72**
Hard	RRTConnect	0.87	8.15	9.07	**0.17**	50.2
	KPIECE	0.69	7.78	9.98	0.25	59.8
	TrajOpt	**1.0**	5.28	2.77	1.45	11.6
	Ours	**1.0**	3.67	3.73	1.19	26.8
	Ours-OV	**1.0**	**2.04**	**0.99**	1.07	**9.3**

SD: standard deviation.

In "Hard" difficulty level, however, the difference in performance becomes more significant. RRTConnect and KPIECE show higher rates of failures, achieving 0.87 and 0.69 success rates, respectively. At the same time, both these methods demonstrate significantly longer runtimes of around 8 s. TrajOpt and both variations of our method maintain a success rate of 1.0. However, there is a significant difference in runtimes. TrajOpt has an average runtime of 5.28 s, while our method with the worst-case overlap volume obstacle cost function demonstrates an average runtime of only 2.04 s, which is more than two times faster.

Notably, our method with the obstacle cost function from the original STOMP still outperforms TrajOpt, demonstrating an average runtime of 3.67 s, but is significantly slower than our method with the proposed worst-case overlap volume obstacle cost. Moreover, Ours-OV also achieves the lowest standard deviation and maximum runtime by a large margin.

We argue that the effect of the proposed obstacle cost component is revealed in the "Hard" difficulty level because the arms have to go deeper inside the cells of the shelf, creating more cases when the algorithm has to leave extended collision regions. Overall, the proposed method successfully solves all 1200 tasks combined across both difficulty levels and consistently achieves low runtimes. Example trajectories from this experiment are shown in Fig. 4.6. Notice how in the last trajectory the arms avoid self-collision with each other.

In Table 4.2 we show average trajectory lengths in joint space as well as average end-effector path lengths for the left, right, and both arms combined. RRTConnect and KPIECE produce trajectories with a lot of unnecessary movements that result in very long trajectories in joint and Cartesian space, especially in the "Hard" difficulty. Such

Table 4.2: Comparison of average trajectory lengths in joint and Cartesian space in the shelf experiment.

Task	Method	Joint space [rad]	End-effector [cm]		
			Left	Right	Combined
Easy	RRTConnect	18.6	130	139	269
	KPIECE	18.9	142	141	283
	TrajOpt	**10.8**	**67**	**69**	**137**
	Ours	11.1	71	70	141
	Ours-OV	11.1	71	70	141
Hard	RRTConnect	31.6	197	226	423
	KPIECE	26.5	180	199	379
	TrajOpt	**12.0**	**77**	**79**	**156**
	Ours	12.6	83	80	163
	Ours-OV	12.4	81	79	160

trajectories are very inefficient and require heavy post-processing to make them shorter and smoother before the execution.

TrajOpt and both variants of our method produce trajectories with short end-effector paths consistently across the difficulty levels. Our method yields trajectories that are slightly longer compared to TrajOpt. On average, the trajectories produced by our method are 3.5% longer in both joint and Cartesian space. We attribute it to the fact that STOMP, which is the base of our method, has a random sampling step that is beneficial for overcoming the local minima but may result in a sub-optimal solution. Both obstacle cost function variations result in trajectories of similar lengths.

Our approach is capable of obtaining qualitatively different trajectories when the cost component weights are adjusted. In Fig. 4.7 we show two trajectories obtained with low and high obstacle cost weights. A trajectory with a low obstacle cost weight is short but avoids obstacles with a low margin. In contrast, a high obstacle cost weight yields a longer but safer trajectory, avoiding the obstacles with a higher margin.

4.5.3 *Closed Kinematic Chain Constraint Scenario*

In this section we evaluate the performance of our method while optimizing trajectories under the closed kinematic chain constraint in the presence of obstacles. For that, we use the following problem setting. The robot has both arms in a neutral configuration, and the task is to bring the hands up while satisfying the closed kinematic chain constraint. The environment contains an obstacle placed in one of the three configurations: obstructing the direct path of the

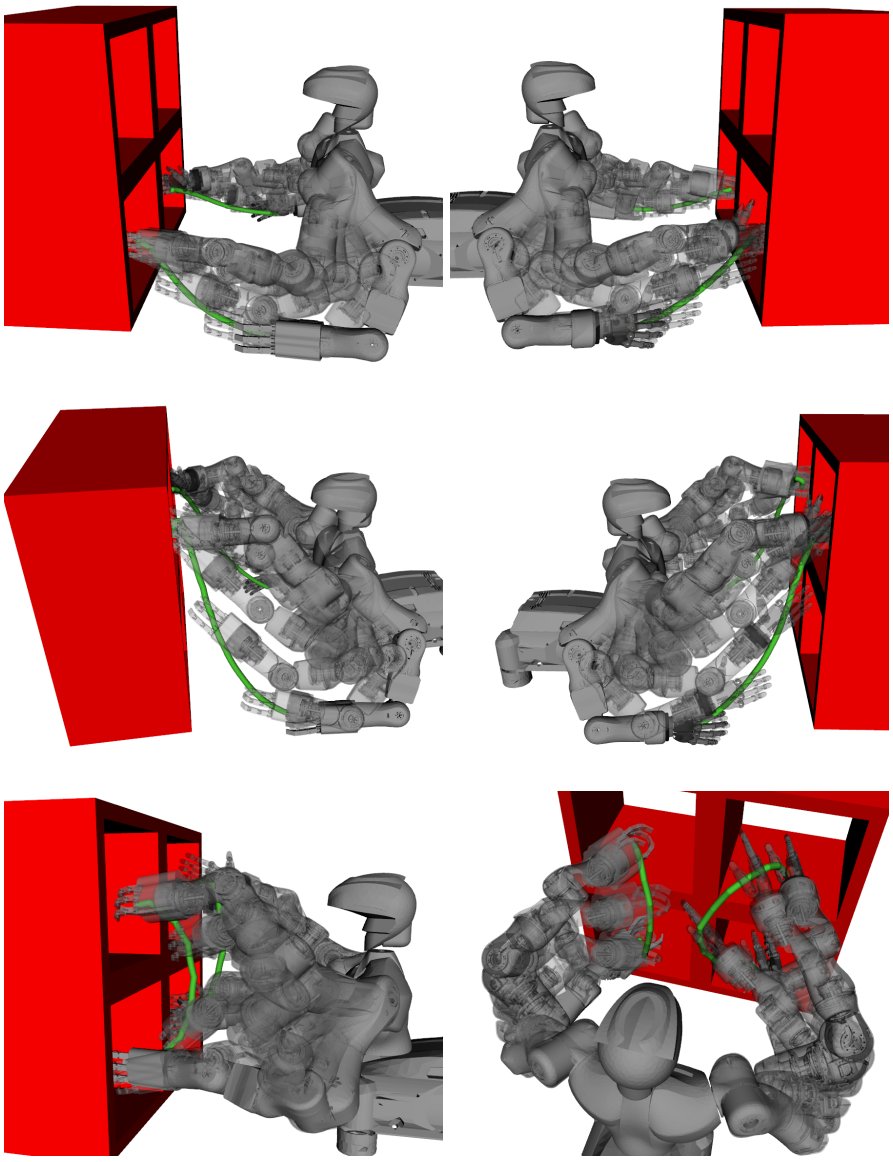


Figure 4.6: Example trajectory rollouts from the shelf experiment. Top to bottom: moving from neutral configuration to the bottom row of the shelf; moving from neutral configuration to the top row of the shelf; moving from the bottom row of the shelf to the top row of the shelf. Perspectives from the left and right sides of the robot are shown on the left and right, respectively. End-effector trajectories are shown in green.

right end-effector, obstructing the direct path of the left end-effector, and obstructing the direct path of both end-effectors. We refer to these scenarios as "Right", "Left", and "Middle", correspondingly. The environment is shown in Fig. 4.8.

Each scenario has two possible permutations of start and goal configurations: starting with hands below an obstacle and moving them above, and vice versa. Same as in the previous experiment, we perform 100 planning attempts per configuration permutation.

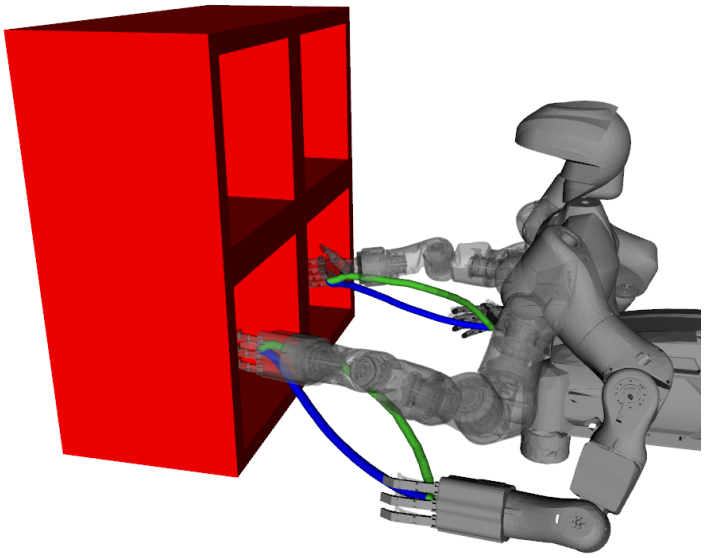


Figure 4.7: Example of qualitatively different trajectories obtained with different obstacle cost weights. Blue: Low weight, short trajectory with a minimal distance to the obstacles. Green: High weight, longer but safer trajectory, avoiding the obstacles with a higher margin.

That results in 200 attempts per scenario and 600 attempts in total. Each method is given 120 s per attempt. An attempt is considered successful if the runtime does not exceed 120 s, the obtained trajectory is collision-free, and the closed-chain constraint is preserved along the whole trajectory.

We compare the performance of our method to TrajOpt. We do not present the results for RRTConnect and KPIECE in this scenario, since both approaches fail to solve the task. In addition, we perform an ablation study to evaluate the method proposed in Section 4.4.2. We

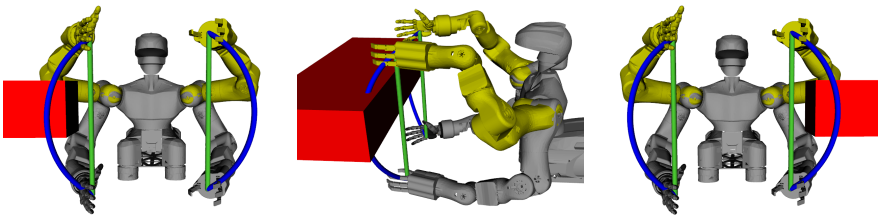


Figure 4.8: Closed kinematic chain constraint environment. From left to right: obstacle on the left, obstacle in the middle, obstacle on the right. Red: obstacle. Gray: start configuration. Yellow: goal configuration. Green: initial trajectories that are straight interpolation in Cartesian space, shown as trajectories of the end-effectors. Blue: initial trajectories that are straight interpolation in joint space, shown as trajectories of the end-effectors.

refer to it as Ours with APM with redundancy resolution (APMRR) (Ours-APMRR). The methods for the ablation study are the following.

First, we use APM with a convenient pose used as an initial pose for the IK solver: Ours with APM with convenient configuration (APMCC) (Ours-APMCC). The convenient configuration for each keyframe is defined as a corresponding pose from the shortest path between the start and the goal that satisfies the closed-chain constraint, without considering the obstacles.

Second, APM where the convenient configurations defined above have a random component added to them: Ours with APM with convenient configuration random (APMCCR) (Ours-APMCCR). The random component is drawn from a normal distribution with mean $\mu = 0\,\text{rad}$ and standard deviation $\sigma = 0.3\,\text{rad}$.

Finally, we also test an approach where the APM that relies on IK is omitted, and the closed-chain constraint is attempted to be satisfied solely using the cost term $\mathcal{C}_{\text{cc}}$ as defined in Section 4.4.2. This approach samples trajectories of both arms directly in the joint space and is referred to as Ours joint space (Ours-JS).

For the APM-based variants of our method, we use selectively damped least squares (SDLS) (Buss and Kim, 2005) to solve the IK. We choose the left arm of the robot as the active sub-chain and the right arm as the passive sub-chain. This choice is arbitrary, and the APM sub-chains can be chosen freely for each optimization task. We use the maximum allowed translation deviation $d_{\text{max}} = 0.015\,\text{cm}$ and maximum allowed orientation deviation $\alpha_{\text{max}} = 0.02\,\text{rad}$.

For Ours-JS variant, the choice of these parameters is crucial to achieving reliable performance. Setting them to the same strict values as for APM-based variants leads to 0 success rate. Thus, we relax them to enable Ours-JS to achieve a non-zero success rate: $d_{\text{max}} = 1.5\,\text{cm}$ and $\alpha_{\text{max}} = 0.15\,\text{rad}$. Note that the trajectories obtained with Ours-JS are then checked for satisfaction of the kinematic chain closure constraint with the same increased thresholds. Thus, such trajectories can be executed in practice only after a post-processing step where the constraint is strictly enforced. Such post-processing may not always be successful due to other constraints that have to be satisfied, such as obstacles and workspace boundaries.

In Table 4.3, success rates and average runtimes are shown. We use only the successful attempts in the runtime calculation. APMRR and APMCC variants of our method show a consistent success rate of 1.0 for each of the three tasks. TrajOpt and APMCCR fail in some of the attempts, demonstrating a success rate slightly below 1.0. The pure joint space variant of our method Ours-JS shows the lowest success rate of 0.71 for all tasks combined, even though it uses relaxed thresholds for the closed kinematic chain constraint. Ours-JS also shows the highest average and maximum runtime. This demonstrates that a naïve attempt to optimize under the closed-chain constraint

Table 4.3: Comparison of success rates and average runtimes in the closed kinematic chain experiment.

Task	Method	Success rate	Runtime [s]			
			Mean	SD	Min	Max
Right	TrajOpt	0.94	36.9	**2.7**	27.1	46.3
	Ours-APMRR	**1.0**	**10.2**	7.2	**3.4**	**45.5**
	Ours-APMCC	**1.0**	14.3	15.8	3.8	93.3
	Ours-APMCCR	**1.0**	16.3	12.4	4.0	58.3
	Ours-JS	0.83	39.2	32.5	7.8	118.3
Left	TrajOpt	0.99	28.3	**1.1**	27.5	**35.5**
	Ours-APMRR	**1.0**	**12.3**	9.4	**2.9**	56.8
	Ours-APMCC	**1.0**	15.7	13.7	3.4	114.5
	Ours-APMCCR	0.99	14.7	11.8	3.2	82.1
	Ours-JS	0.59	47.0	28.9	6.3	115.9
Middle	TrajOpt	0.99	34.0	**4.0**	29.1	**51.3**
	Ours-APMRR	**1.0**	**16.9**	10.8	**5.5**	82.6
	Ours-APMCC	**1.0**	17.5	10.9	6.6	98.5
	Ours-APMCCR	0.99	24.1	15.8	7.8	117.7
	Ours-JS	0.71	47.6	33.1	7.4	117.8

SD: standard deviation.

directly does not yield good results. At the same time, all APM-based variants of our method outperform both Ours-JS and TrajOpt.

Noticeably, the proposed APMRR outperforms both APMCC and APMCCR, where convenient configurations are used as start configurations for the IK. The random component added to the convenient configurations in APMCCR makes the optimizer converge longer compared to APMCC. It is worth noting that the biggest improvement of runtime of APMRR compared to APMCC is achieved in the "Right" task, where the obstacle is obstructing the path of the right end-effector. That is explained by the fact that we set the left arm to be the active sub-chain. The optimization of the initial configurations for the IK in APMRR helps to converge to the collision-free regions quicker compared to APMCC, where the algorithm can only influence the IK solution by changing the trajectory of the active arm.

Overall, the proposed method Ours-APMRR consistently demonstrates the lowest average runtimes as well as the lowest minimum runtime, outperforming TrajOpt by more than a factor of two and Ours-APMCC by 15%. The SDF computation for our method takes 0.19 ± 0.01 s on average and is incorporated in the total runtime.

In Table 4.4 we show average trajectory lengths in joint space as well as average end-effector path lengths for the left, right, and both end-effectors combined. Ours-APMCCR, followed closely by TrajOpt,

Table 4.4: Comparison of average trajectory lengths in joint and Cartesian space in the closed kinematic chain experiment.

Task	Method	Joint space [rad]	End-effector [cm]		
			Left	Right	Combined
Right	TrajOpt	23.3	129	107	236
	Ours-APMRR	**15.1**	**70**	**78**	**148**
	Ours-APMCC	15.4	**70**	81	152
	Ours-APMCCR	24.1	75	90	166
	Ours-JS	16.9	86	88	174
Left	TrajOpt	22.5	106	106	212
	Ours-APMRR	**15.8**	**75**	81	**156**
	Ours-APMCC	15.9	78	82	160
	Ours-APMCCR	22.3	76	81	157
	Ours-JS	16.2	82	**80**	162
Middle	TrajOpt	22.9	115	118	233
	Ours-APMRR	20.7	91	106	197
	Ours-APMCC	20.4	91	105	196
	Ours-APMCCR	26.9	90	104	194
	Ours-JS	**18.1**	**87**	**90**	**177**

produces the longest trajectories, both in Cartesian and joint spaces. Ours-APMRR produces the shortest trajectories in "Right" and "Left" tasks, closely followed by Ours-APMCC and then — by Ours-JS. In the "Middle" task, Ours-JS shows the best result. We explain it by the fact that given relaxed deviation tolerances for the closed kinematic chain constraint, Ours-JS has an option to avoid the obstacle in the middle following the shortest path while not strictly satisfying the constraint. In "Right" and "Left" tasks, that is not the case, since an obstacle is only on one side. The shortest paths of the arm avoiding the obstacle allow for the closed-chain constraint to be satisfied strictly by the other arm that has the freedom to move freely in the obstacle-free region.

It is worth noting that a shortcoming of using the APM-based approaches is that the end-effector of the passive chain usually has longer trajectories than the end-effector of the active chain. TrajOpt and Ours-JS do not share this property. Thus, they yield trajectories that are more similar in length for the left and the right arm correspondingly.

Example rollouts from this experiment are shown in Fig. 4.9. In addition, we show qualitatively different trajectories that avoid an obstacle while maintaining the kinematic chain closure constraint in Fig. 4.10. In Fig. 4.11, trajectories optimized with respect to torque and duration are shown. Even though the available range of motion

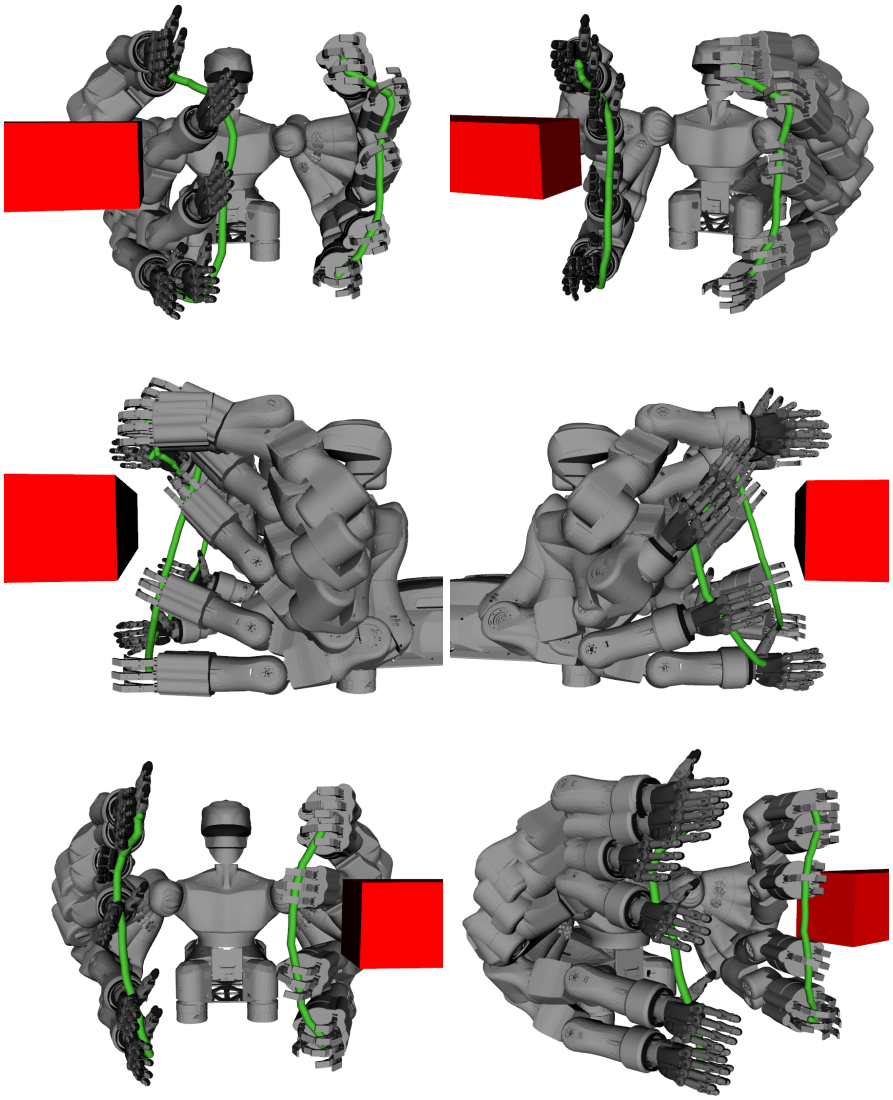


Figure 4.9: Trajectory rollouts from the closed kinematic chain constraint experiment. Top to bottom: avoiding the obstacle on the right; avoiding the obstacle in the middle; avoiding the obstacle on the left. In all examples, the end-effectors are constrained to preserve the closure constraint along the whole trajectory. Two different perspectives are shown per trajectory. End-effector trajectories are shown in green.

is heavily restricted by the kinematic constraints, our method still enables obtaining qualitatively different trajectories, depending on the requirements introduced by each specific task.

One important aspect to consider when using a planner such as STOMP is that the initial trajectory is extremely important and can affect the runtimes, success rates, and the resulting trajectories. For this reason, we compare two common initial trajectory choices for the Ours-APMRR method. The two types of initial trajectories we consider are straight-line interpolation in joint and Cartesian space.

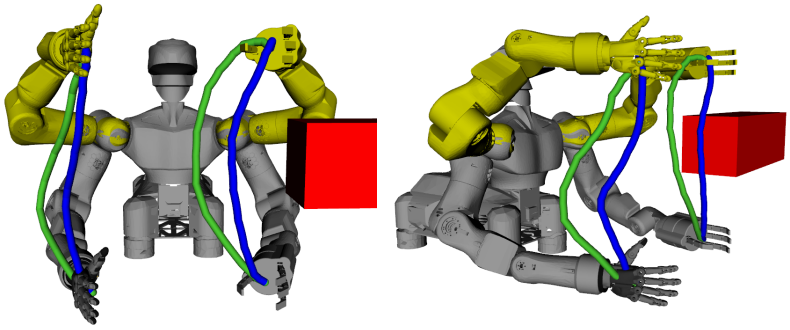


Figure 4.10: Qualitatively different trajectories obtained with different obstacle cost weights while satisfying the closed kinematic chain constraint. Blue: low weight, short trajectory at a minimal distance to the obstacle. Green: high weight, longer trajectory that maximizes the distance to the obstacles and is safer. Grey: start configuration. Yellow: goal configuration.

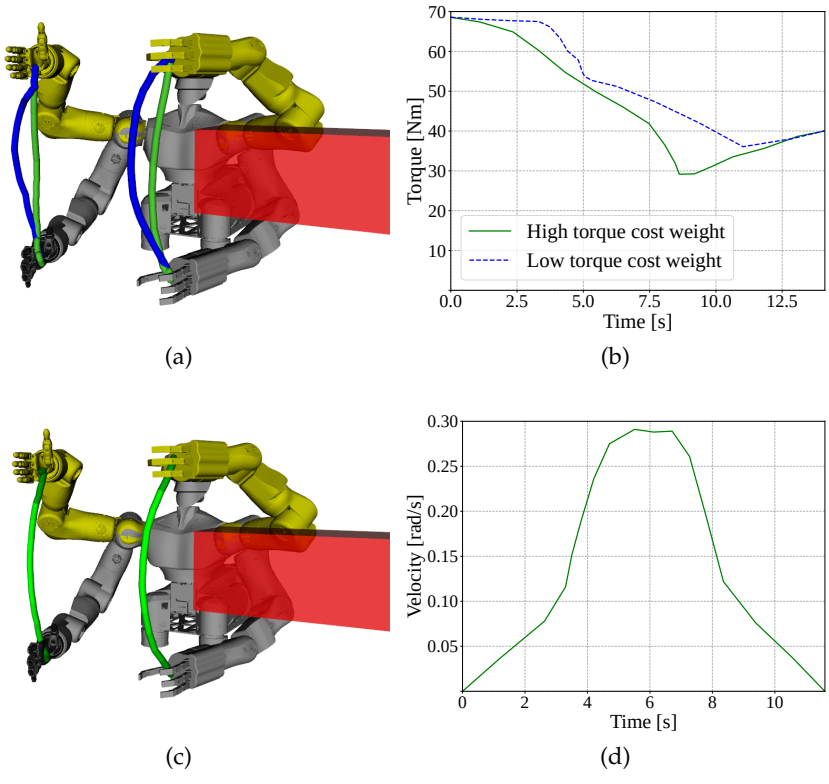


Figure 4.11: Trajectories with torque and duration optimization, avoiding an obstacle and satisfying the closure constraint. (a) Torque optimization. Trajectories of the end-effectors are shown. Blue: low torque cost weight. Green: high torque cost weight. Increasing the torque cost weight results in a trajectory with hands being closer to the torso, reducing the torque. (b) Total torque vs. time plot. (c) Duration optimization. The trajectory of the end-effectors is color-coded. The brighter, the higher the velocity. (d) Velocity profile of the joint with longest path.

Table 4.5: Average runtimes and success rates with closure constraint and different initial trajectories: straight-line interpolation in joint and Cartesian spaces.

Task	Initialization type	Success rate	Runtime [s]			
			Mean	SD	Min	Max
Right	Cartesian space	**1.0**	**10.2**	**7.2**	3.4	**45.5**
	Joint space	**1.0**	12.1	9.9	**2.9**	60.3
Left	Cartesian space	**1.0**	**12.3**	**9.4**	**2.9**	**56.8**
	Joint space	0.97	22.1	27.1	3.1	116.9
Middle	Cartesian space	**1.0**	**16.9**	**10.8**	**5.5**	82.6
	Joint space	**1.0**	18.2	12.1	5.6	**63.5**

SD: standard deviation.

For problems involving closed kinematic chain constraint, the straight-line interpolation in Cartesian space appears to be more natural, since the optimizer starts with a trajectory that already satisfies the constraint. To verify that, we perform the same experiment as described previously, analyzing the optimization performance with these two common initial trajectories.

In Table 4.5, success rates and average runtimes are shown. We use only the successful attempts in the runtime calculation. Both initialization types resulted in all tasks being solved, except for the "Left" scenario with the joint space interpolation initialization. That is because in this case such an initialization makes the trajectory of the active arm lie deep inside the obstacle, and the feasible solution is far away (Fig. 4.8). For the same reason, runtimes for this scenario with joint space initialization are also considerably higher compared to the straight line in Cartesian space initialization. Nevertheless, even less feasible joint space interpolation initialization yields consistent performance, solving almost all tasks and demonstrating a runtime lower than TrajOpt.

Table 4.6: Average path lengths with closure constraint and different initial trajectories: straight-line interpolation in joint and Cartesian spaces.

Task	Initialization type	Joint space [rad]	End-effector [cm]		
			Left	Right	Combined
Right	Cartesian space	**15.1**	**70**	**78**	**148**
	Joint space	17.2	75	87	162
Left	Cartesian space	**15.8**	**75**	**81**	**156**
	Joint space	16.9	81	86	167
Middle	Cartesian space	20.7	91	106	197
	Joint space	**20.3**	**90**	**103**	**193**

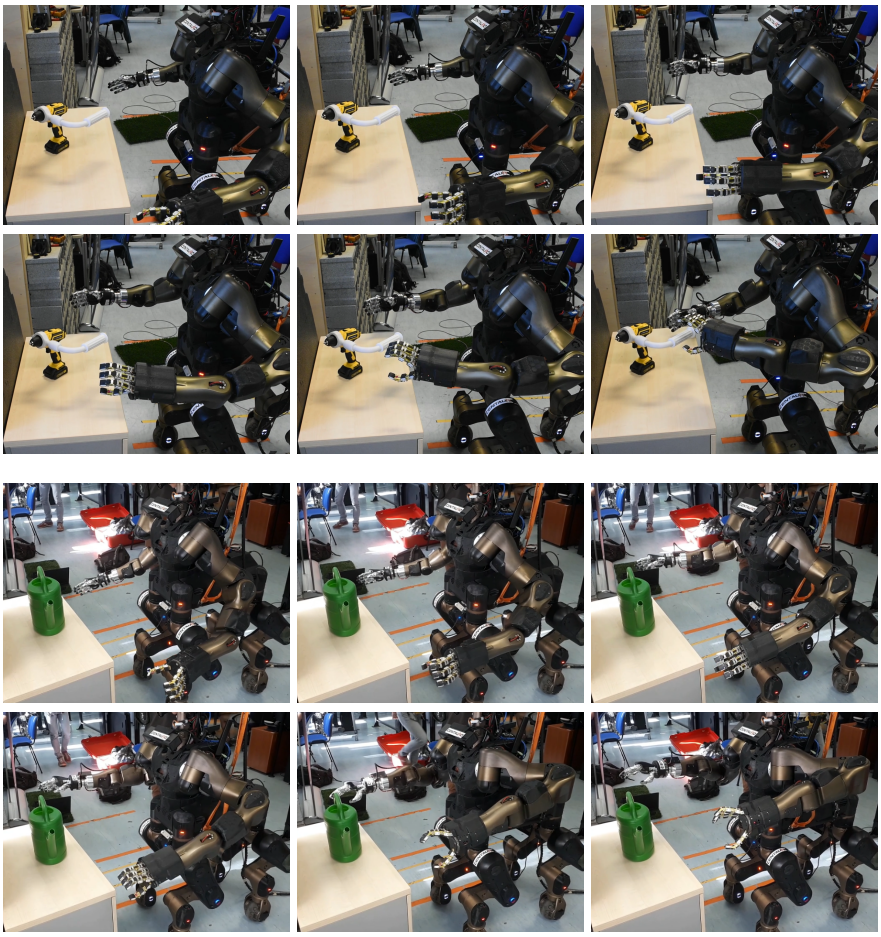


Figure 4.12: Snapshots of trajectory executions on Centauro robot. *Top:* Reaching a pre-grasp pose for the drill. *Bottom:* Reaching a pre-grasp pose for the watering can. In both cases, the arms successfully avoid the table and reach the required pre-grasp poses.

In Table 4.6 we show average trajectory lengths in joint space as well as average end-effector path lengths for the left, right, and both arms combined. The straight-line interpolation in Cartesian space initialization yields shorter paths in "Left" and "Right" environments, producing only slightly longer trajectories in the Middle environment. This experiment highlights the importance of the choice of the initial trajectory. At the same time, our method demonstrates consistent performance even when initialized with unfavorable trajectories, advocating for its robustness.

4.5.4 *Real-robot Experiments*

We apply our method on the real Centauro robot to plan for dual-arm trajectories with independent arm movement. The robot starts next to a table with its arms in a neutral configuration. The task is to reach given pre-grasp configurations for the objects on the table. Snapshots

of the trajectory executions from this experiment are shown in Fig. 4.12. We perform the experiment six times and in all the trials the robot successfully avoids the table and reaches the goal configuration. The average runtime of our method is 1.07 ± 0.34 s. This experiment demonstrates that our method can be efficiently applied in the real world. Videos of the experiments are available online[23].

4.6 DISCUSSION

In this chapter, we presented an approach for dual-arm trajectory optimization. The method leverages a multi-component cost function to optimize for multiple costs and constraints. We introduced an obstacle cost term that speeds up the convergence of the algorithm towards the collision-free regions. That was achieved by employing the proposed obstacle cost component based on the estimation of the worst-case overlap volume. The performed ablation studies indicated that the proposed modification of the obstacle cost term substantially decreased the average runtime of the method.

Finally, we addressed the closed kinematic chain constraint with a dedicated cost term and an APM that relies on an IK solver. We optimize for favorable initial configurations for the IK solver, implicitly resolving the redundancy. The method was quantitatively evaluated in simulation on the Centauro robot, featuring two 7 DoF arms. The experiments demonstrated that the proposed approach outperforms RRTConnect, KPIECE, and TrajOpt in terms of success rate and runtime for unconstrained planning problems. The approach also demonstrated lower runtime and a higher success rate compared to TrajOpt in obstacle avoidance tasks under the closed kinematic chain constraint. At the same time, qualitative experiments demonstrated that our method is capable of optimizing for several different costs while satisfying strict kinematic constraints.

The study of the effects of different initial trajectories showcased that our method reliably converged to a feasible solution even when initialized with unfavorable trajectories. Qualitative evaluation performed on the real robot demonstrated that the method can facilitate efficient dual-arm trajectory optimization in unstructured environments.

2 https://www.ais.uni-bonn.de/videos/Humanoids_2018_Pavlichenko
3 https://www.ais.uni-bonn.de/videos/Pavlichenko_Dual_Arm_Optimization

5

DEXTEROUS PRE-GRASP MANIPULATION WITH DEEP REINFORCEMENT LEARNING

PREFACE

This chapter is adapted from Pavlichenko and Behnke, 2023, previously published by IEEE and presented at the 19th IEEE International Conference on Automation Science and Engineering (CASE 2023), and Pavlichenko and Behnke, 2025, previously published by IEEE in the IEEE Transactions on Automation Science and Engineering (T-ASE).

Statement of Personal Contribution

The author of this thesis substantially contributed to all aspects of the publication (Pavlichenko and Behnke, 2023), including the literature survey, conception, design, and implementation of the proposed method, the preparation and conduct of experiments and evaluation of the proposed approach, the analysis and interpretation of the experimental results, drafting the manuscript, as well as the revision and final editing of the version to be published.

The author of this thesis substantially contributed to the following aspects of the publication (Pavlichenko and Behnke, 2025), including the literature survey, conception, design, and implementation of the proposed method, the preparation and conduct of experiments and evaluation of the proposed approach, the analysis and interpretation of the experimental results, drafting the manuscript, as well as the revision and final editing of the version to be published.

The content presented in this chapter, unless otherwise stated, is the contribution of the author of this thesis.

ABSTRACT

In this chapter, we present a methodology for learning a dexterous pre-grasp manipulation policy to achieve human-like functional grasps using DRL. Many objects, such as tools, can be used only if grasped in a very specific way—grasped functionally. Often, a direct functional grasp is not possible. We introduce a dense multi-component reward function that enables learning a single policy, capable of dexterous pre-grasp manipulation of novel instances of several known object categories. The policy is learned purely by means of DRL without any expert demonstrations. In addition, we propose two different ways to represent a desired grasp: explicit and more abstract, constraint-based.

The policy is trained in a highly-parallelized simulation on a single graphics processing unit (GPU) in under three hours. We show that our method consistently learns to successfully manipulate and achieve the desired functional grasps on previously unseen instances of known categories using both grasp representations.

5.1 INTRODUCTION

In Chapter 4 we addressed dual-arm trajectory optimization by introducing several modifications to the STOMP, including an improved obstacle cost function component and methodology with implicit redundancy resolution to optimize under the closed kinematic chain constraint. This approach efficiently finds trajectories to bring the robotic arms from given start to given goal configurations. After reaching the desired location, the next task is often to manipulate an object. This typically involves complex hand-to-object interactions.

In this chapter we address the problem of pre-grasp manipulation with a dexterous anthropomorphic hand through a DRL. Many objects are made for human hands and require a specific grasp for use. For example, a drill requires a power grasp with the index finger on the trigger. We refer to such grasps as *functional*. Often, a functional grasp cannot be achieved directly because the object is in the wrong pose. This can be addressed with pre-grasp manipulation: repositioning and reorienting the object until the desired functional grasp is achieved. However, robustly performing interactive functional grasping with a dexterous multi-finger hand is challenging. Addressing this problem will enable robots to utilize tools and functional objects designed for human use.

Inspired by our earlier work on functional re-grasping with a dual-arm setup (Pavlichenko, Rodriguez, Lenz, et al., 2019), we introduce a methodology that replaces several intricate classical components with a single data-driven approach. DRL has been applied to multiple complex robotic domains (D. Chen et al., 2021; Hwangbo et al., 2019; Pane et al., 2019; Rodriguez and Behnke, 2021). In this chapter, we utilize a highly efficient GPU-based simulation (Makoviychuk et al., 2021) in conjunction with DRL to train a policy for dexterous pre-grasp manipulation.

Many approaches focus on learning the policies directly from the low-level sensory inputs, such as camera images and point clouds (Mandikal and Grauman, 2020; Qin et al., 2022). We argue, however, that the majority of data points in these inputs, such as background pixels in an image, are irrelevant to the manipulation policy. Therefore, we assume that perception is performed by an external method, and our approach is provided with high-level semantic information. For example, such external methods could reconstruct the object shape from partial observations (X. Han, Laga, and Ben-

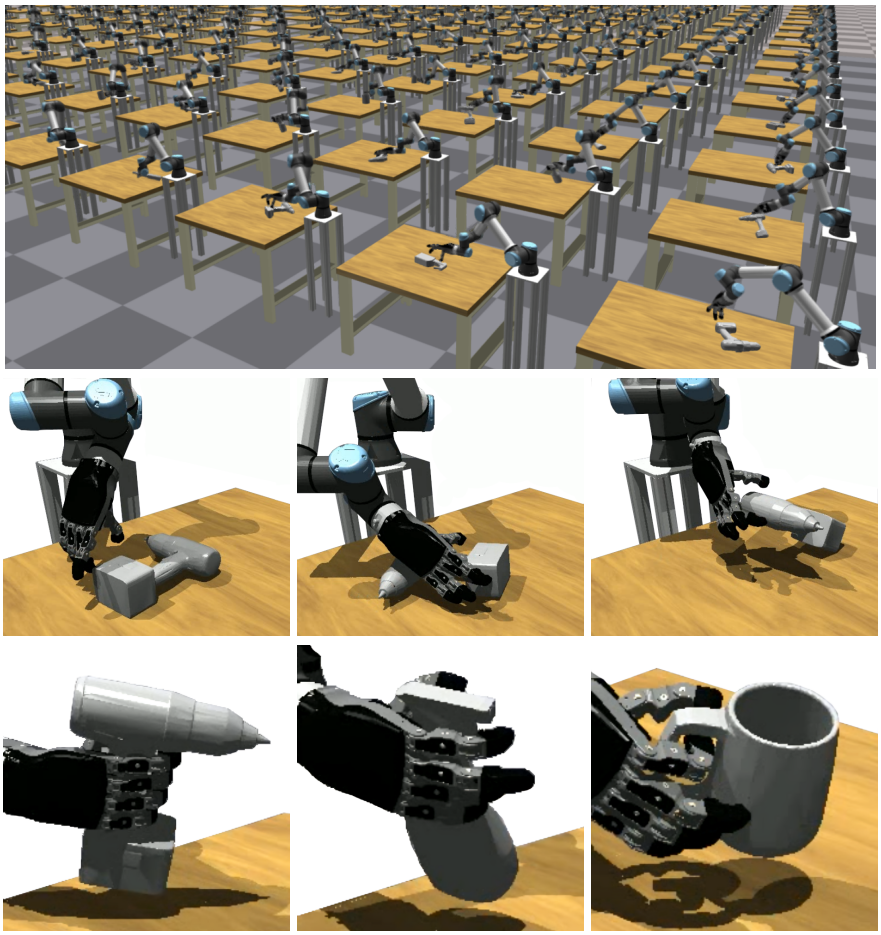


Figure 5.1: Dexterous functional grasping. Top to bottom: Learning human-like functional pre-grasp manipulation in multiple parallel environments; Dexterous pre-grasp manipulation that includes reorienting and repositioning a drill; Intuitive human-like functional grasps for three object categories, achieved only with a target index fingertip position and desired object orientation.

namoun, 2021; Rodriguez, Huber, and Behnke, 2020; D. Yang et al., 2021; L. Zhou et al., 2024), transfer the functional grasp (Rodriguez and Behnke, 2018; Wei et al., 2024; R. Wu et al., 2023; Y. Zhang et al., 2023), and estimate the 6D pose of the object (Amini, Periyasamy, and Behnke, 2022; Deng et al., 2020; Hofer et al., 2021; G. Wang et al., 2024).

This assumption enables speeding up the learning process since the policy can be represented by a model with fewer parameters. In addition, the expensive image rendering is avoided. By considering multiple object instances within the same category during training, we further reduce the inputs to the policy, as category-specific features of the object geometry and dynamics are implicitly learned. Finally, we eliminate the need for expert demonstrations by introducing a dense multi-component reward function that naturally encourages dexterous manipulation.

An ability to successfully achieve target functional grasp is the desired behavior of the learned policy. A target grasp representation greatly influences the speed of learning as well as the final result. In this work, we explore two possible target grasp representations. Each of them comes with its own strengths and weaknesses.

An explicit target grasp representation strictly defines desired hand and finger poses relative to an object. It introduces a clear desired result and prevents the policy from being stuck in a certain category of sub-optimal behaviors. However, it comes at the cost of having an external oracle defining these explicit targets, which may be challenging for novel object instances.

Alternatively, we propose to represent functional grasps more abstractly, through a constraint. As such, in this work, we select index fingertip position and hand orientation relative to an object. Such representation is more compact and enables the policy to learn a variety of different grasps that satisfy the constraint, such as the index finger on the trigger of a drill. Defining this constraint for novel instances is an easier task compared to an explicit grasp representation since accurate finger positions are not required. However, this comes at the cost of ensuring that the learned grasps are sufficient to securely lift and utilize the objects. We show that the proposed dense multi-component reward function can be effectively applied to both grasp representations with minimal modifications and consistently yields meaningful policies.

To evaluate the proposed method, a single policy is learned in simulation on three conceptually distinct rigid object categories: drills, spray bottles, and mugs. Using dense multi-component reward, the policy learns to perform dexterous pre-grasp manipulation on previously unseen object instances of known categories (Fig. 5.1). The learning is performed in simulation in less than three hours on a single GPU.

In summary, our main contributions are:

- a multi-component dense reward formulation that quickly yields policies capable of dexterous pre-grasp manipulation of novel objects using a multi-finger hand,
- a constraint-based target functional grasp representation, which is easier to define compared to an explicit grasp representation. Such representation enables exploring different grasp configurations, satisfying the functionality constraint, and
- a functional grasping object mesh dataset with three object categories.

5.2 RELATED WORK

Dexterous pre-grasp manipulation has been an active area of research for decades. Multiple classical model-based approaches have been proposed (L. Y. Chang, Srinivasa, and Pollard, 2010; Dogar and Srinivasa, 2010; L. Han and Trinkle, 1998; K. Hang, Morgan, and Dollar, 2019; Muhayyuddin et al., 2018). They work well with known objects and exact models but require carefully hand-crafted, task-dependent algorithms. Moreover, such approaches suffer from uncertainties introduced by highly dynamic, contact-rich manipulation.

In our prior research (Pavlichenko, Rodriguez, Lenz, et al., 2019), we address functional grasping of novel object instances of known categories by means of re-grasping with a dual-arm robot. There the main idea is to perform an arbitrary grasp with one hand, gaining control over the pose of the object. Then, position the hand that is holding the object in such a way that a direct functional grasp can be achieved by the other hand. The described manipulation pipeline is implemented with several classical approaches. While humans commonly employ both hands simultaneously for manipulation tasks, the dexterity of a single human hand significantly exceeds the requirements for functional pre-grasp manipulation. Building upon our prior work, a natural progression toward achieving comparable performance with robotic manipulators involves executing the same task using a single hand. However, achieving such highly dynamic manipulation with classical approaches is very challenging.

A promising solution to such problems is to leverage data-driven methods. In particular, DRL and imitation learning (IL) using NNs to represent policies for dexterous manipulation have gained much popularity in recent years (O. M. Andrychowicz et al., 2020; Levine et al., 2016; H. Zhu et al., 2019). By learning purely from observed experiences and/or provided demonstrations, these methods yield highly reactive policies capable of dexterous multi-finger manipulation.

W.-M. Zhou and Held, 2022 address pre-grasp manipulation of objects in ungraspable configurations through extrinsic dexterity. Their method uses model-free RL to learn to push objects against a wall to achieve a graspable pose. The method uses minimalistic object representation, similar to our approach. However, it has difficulties generalizing to objects with complex non-convex shapes. In our approach, this issue is resolved by learned implicit category-specific geometry knowledge. Similarly, Z. Sun et al., 2020 use model-free RL to obtain a policy for a dual-arm robot that pushes an object next to a wall and achieves a grasp with the other hand. Both works use parallel grippers, which limits the manipulation dexterity.

Qin et al., 2022 train a dexterous manipulation policy for an Allegro hand to grasp novel objects of a known category. Their approach uses point clouds as input to provide information about object geometry.

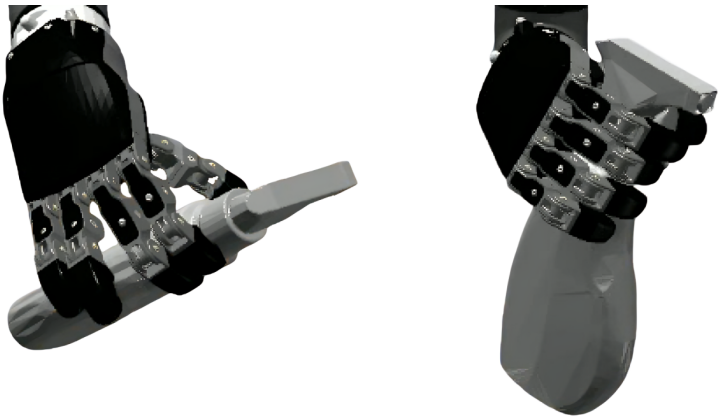


Figure 5.2: Example of a functional and an arbitrary grasp. *Left:* Arbitrary grasp. *Right:* Functional grasp. Note that an arbitrary grasp is suitable for pick-and-place tasks. At the same time, only functional grasp with an index finger on the trigger enables the designated use of the spray bottle.

The difference to our work is that the grasps are arbitrary, while we specifically address functional grasping, as shown in Fig. 5.2. Mandikal and Grauman, 2020 propose to learn a policy with object-centric affordances to dexterously grasp objects. Notably, the policy is learned with a prior derived from observing manipulation videos. This requires tedious annotation of human grasp regions in the images.

A wide range of works is based on real-world expert demonstrations (Mandikal and Grauman, 2022; Radosavovic et al., 2021; Rajeswaran et al., 2018). These approaches have to deal with the challenges of mapping human motion to the kinematics of the robot arm and hand. Hence, the direct applicability to different robotic setups is not straightforward. Z. Chen et al., 2022 address this issue by bootstrapping a small dataset of human demonstrations with a larger dataset including novel objects and grasps. The objects are deformed, and dynamically consistent grasps are generated. The policy is then trained in a supervised manner in simulation, followed by a direct transfer to the real world. This method struggles with objects of complex shapes. Palleschi et al., 2023 utilize human demonstrations to teach the policy to grasp a wide range of objects. The approach is evaluated on two grippers: soft and rigid. In contrast, in our work, we avoid using explicit demonstrations and instead rely on a general dense reward function to guide the policy towards dexterous manipulation.

A completely different approach is proposed by Dasari, Gupta, and Kumar, 2023: a combination of exemplar object trajectories with predefined pre-grasp configurations as training data. The policy learns to perform a wide variety of tasks in simulation without any task-specific engineering. The key difference to our work is that learned behaviors directly depend on supplied exemplar trajectories. Addi-

tionally, the manipulation is performed by a freely floating hand. Such an approach relaxes several constraints imposed by the kinematics of the robotic arm. This makes manipulation easier but less realistic, especially in cases when objects are on the edge of the workspace. The pre-grasp-based approaches were also introduced in (Baek et al., 2021; Kappler, L. Chang, et al., 2010; Kappler, L. Y. Chang, et al., 2012).

T. Wu et al., 2024 utilize a teacher-student approach combined with policy distillation. This method produces a policy capable of repositioning and reorienting objects across a wide range of categories. The goal is to achieve the required functional grasp object poses without actually grasping the object. In contrast, our approach focuses on achieving required functional grasps while simultaneously learning to perform the pre-grasp manipulation.

Agarwal et al., 2023 propose an object-hand manipulation representation for dexterous robotic hands, followed by a functional grasp synthesis framework, and evaluate the approach in the real world. The main disadvantage of this method is the necessity to compose a complex and large dataset. T. Zhu, R. Wu, J. Hang, et al., 2023 propose to use eigengrasps to reduce the search space of RL using a small dataset collected from human expert demonstrations. The target functional grasps are predicted with an affordance model. The approach is successfully transferred and evaluated in the real world.

In contrast to these approaches, we avoid any expert demonstrations. Instead, we focus on obtaining dexterous manipulation policies solely by means of a dense multi-component reward function that is formulated without using specific hand, arm, or object details.

5.3 BACKGROUND

The objective in this chapter is to learn a policy π that achieves pre-grasp manipulation of novel object instances with the aim of reaching a given functional grasp. The policy π_θ is represented by a DNN and is parameterized by weights θ, learned with DRL. The problem is modeled as a MDP: $\{S, A, P, r\}$ with state space $S \in \mathbb{R}^n$, action space $A \in \mathbb{R}^m$, state transition function $P\colon S \times A \mapsto S$, and reward function $r\colon S \times A \mapsto \mathbb{R}$. Since the problem has continuous state and action space, the policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{s})$ represents an action probability distribution when observing a state $\boldsymbol{s}(t)$ at a timestep t.
The objective of DRL is to maximize the expected reward:

$$J(\pi_\theta) = \sum_{t=0}^{T} \mathbb{E}[\gamma^t r(\boldsymbol{s}(t), \boldsymbol{a}(t))], \tag{5.1}$$

where $\gamma \in [0, 1]$ is the discounting factor.

In this chapter, the policy is provided with a target functional grasp to be reached. In the case of the explicit grasp representation, the

target is a pre-grasp, defined as a 6D hand pose in an object frame plus hand joint positions. This pre-grasp is very close to the desired functional grasp, so closing the hand will guarantee a successful grasp. The advantage of a pre-grasp is that it can be reached more freely, and slight inaccuracies in its definition are negligible, as discussed by Dasari, Gupta, and Kumar, 2023. In the case of the constraint-based target grasp representation, the target can be exact, as it does not include finger positions, making it less likely to have an unreachable target.

5.4 EXPLICIT TARGET GRASP REPRESENTATION

A target functional grasp is explicitly represented by a 6D pose of the end-effector in the object frame of reference and the joint positions of the fingers. This representation has the advantage of providing the policy with a concrete goal, which typically enhances learning speed and convergence stability. A disadvantage of such representation is twofold. First, very specific grasps that lead to a desired outcome have to be produced, which is not straightforward, especially for novel object instances. Second, an explicit target grasp disallows the policy to explore other grasp configurations. Oftentimes there are multiple grasp configurations that make the grasp functional but are different from the given explicit target grasp. In this section, we present the methodology for learning pre-grasp manipulation with explicit grasp representation with DRL.

5.4.1 *Action Space*

The policy produces actions $\boldsymbol{a}(t)$ with a frequency of 30 Hz. An action represents a relative displacement in 3D hand position, hand orientation, and hand joint position increments. With this action definition, hand joint targets are straightforward to obtain. The arm joint targets are calculated via IK. Finally, the joints are controlled with PD controllers.

In this work we apply the proposed method to a 6 DoF UR5e robotic arm with an attached 11 DoF Schunk SIH hand. The joints of the hand are coupled, leaving five controllable DoF. Thus, in this work, an action is an 11-element vector, where three elements define a displacement of the 3D hand position, three elements define a displacement of the hand rotation as Euler angles, and five elements define a displacement of the hand joint positions. We further assume a five-fingered hand with five controllable DoF. However, it is straightforward to apply our approach to a hand with an arbitrary number of fingers and DoF. We use Euler angles representation for the rotation-related part of the action, as it is straightforward to obtain the next target with small iterative increments while using a minimal number of variables.

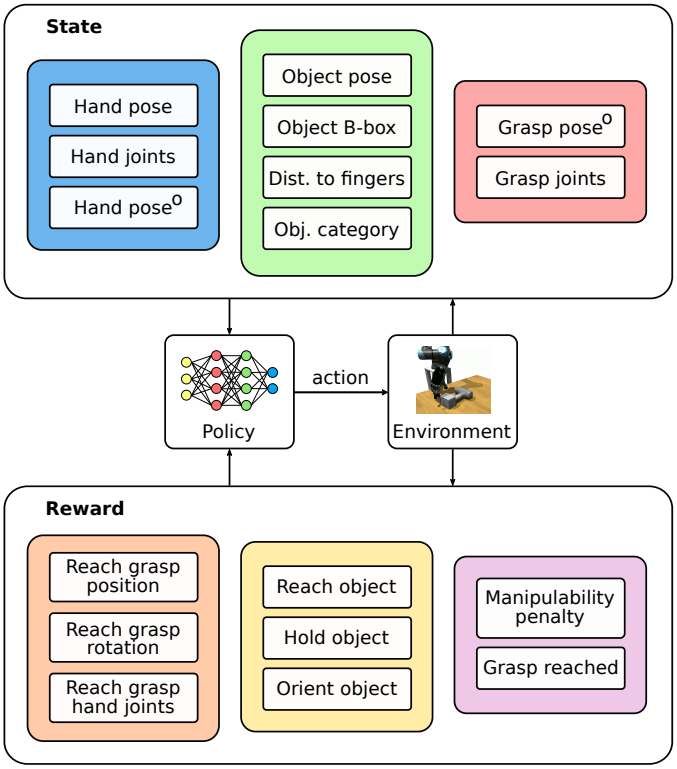


Figure 5.3: State representation and the reward function diagram. The state consists of information about the hand, the object, and the target functional grasp. "O" denotes object frame of reference. The reward function consists of a term encouraging reaching the target grasp, a term encouraging pre-grasp manipulation, and a low manipulability score penalty.

5.4.2 *State Space*

The top part of Fig. 5.3 illustrates the state vector $\boldsymbol{s}(t)$, which consists of three distinctive parts: information about the hand $\boldsymbol{h}$, information about the object $\boldsymbol{o}$, and information about the target functional grasp $\boldsymbol{g}$:

$$\boldsymbol{s} = [\boldsymbol{h}, \boldsymbol{o}, \boldsymbol{g}]. \tag{5.2}$$

Information about the hand is a column vector:

$$\boldsymbol{h} = [\boldsymbol{h}_{\mathrm{p}}, \boldsymbol{h}_{\mathrm{r}}, \boldsymbol{h}_{\mathrm{j}}, \boldsymbol{h}_{\mathrm{p}}^{O}, \boldsymbol{h}_{\mathrm{r}}^{O}], \tag{5.3}$$

where $\boldsymbol{h}_{\mathrm{p}} = [h_{\mathrm{p}_x}, h_{\mathrm{p}_y}, h_{\mathrm{p}_z}]$ is a 3D hand position vector, $\boldsymbol{h}_{\mathrm{r}}$ is a 4-element hand rotation vector represented by a quaternion, and $\boldsymbol{h}_{\mathrm{j}}$ is a 5-element hand joint position vector; $\boldsymbol{h}_{\mathrm{p}}^{O}$ and $\boldsymbol{h}_{\mathrm{r}}^{O}$ are hand position and rotation in the object frame of reference O. Thus, information about the hand $\boldsymbol{h}$ is a 19-element vector. In this chapter we frequently use the following subscripts: $\boldsymbol{x}_{\mathrm{p}}$ denoting 3D position, $\boldsymbol{x}_{\mathrm{r}}$ denoting rotation expressed as a quaternion, and $\boldsymbol{x}_{\mathrm{j}}$ denoting joint positions.

Information about the object is a column vector:

$$\boldsymbol{o} = [\boldsymbol{o}_\text{p}, \boldsymbol{o}_\text{r}, \boldsymbol{o}_\text{bb}, \boldsymbol{o}_\text{s}, \boldsymbol{o}_\text{c}], \tag{5.4}$$

where $\boldsymbol{o}_\text{p}$ is a 3D object position, $\boldsymbol{o}_\text{r}$ is a 4-element object rotation vector represented by a quaternion, $\boldsymbol{o}_\text{bb}$ is a 6-element vector representing an object bounding box by the two 3D positions of diagonally opposing bounding box corners, $\boldsymbol{o}_\text{s}$ is a 10-element vector of signed distances between fingertips and middles of the fingers to the object surface, and $\boldsymbol{o}_\text{c}$ is a C-element one-hot vector representing object category. Distances from fingers to object surface are efficiently calculated from a precomputed object SDF. We adopt this approach from the work of Mosbach and Behnke, 2022. Thus, information about the object is a $(23 + C)$-element vector. Such representation is compact; however, the general geometric features of the object categories are learned implicitly from the experience.
The desired functional grasp is provided as a column vector:

$$\boldsymbol{g} = [\boldsymbol{g}^O_\text{p}, \boldsymbol{g}^O_\text{r}, \boldsymbol{g}_\text{j}], \tag{5.5}$$

where $\boldsymbol{g}^O_\text{p}$ is a 3D hand position in the object frame of reference, $\boldsymbol{g}^O_\text{r}$ is a 4-element hand rotation vector represented by a quaternion, and $\boldsymbol{g}_\text{j}$ is a 5-element hand joint position vector. The target functional grasp is represented by a 12-element vector. In practice, functional grasps can be provided by methods such as (Rodriguez and Behnke, 2018; T. Zhu, R. Wu, X. Lin, et al., 2021).

In this work we have the number of categories $C = 3$. Thus, the state is a 57-element vector. It resembles a high-level semantic representation of the scene. This compact state can be computed fast on a GPU and thus facilitates quick learning. Moreover, compared to DRL models that learn directly from raw visual inputs, smaller models with fewer parameters can be used.

5.4.3 *Reward Function*

The bottom part of Fig. 5.3 illustrates the composition of the reward function $r(t)$ that is defined as:

$$r(t) = r_\text{grasp}(t) + r_\text{man}(t) + r_\text{MP}(t) + r_T(t), \tag{5.6}$$

where r_grasp encourages movement towards the target grasp $\boldsymbol{g}$, r_man encourages pre-grasp manipulation of an object, r_MP penalizes being in configurations with low manipulability, and r_T rewards reaching the target functional grasp $\boldsymbol{g}$. Each reward component is defined to be in $[-1, 1]$ and is described in detail below. For brevity, we omit specifying a dependency on time t, unless necessary.

First, we define the distance function ϕ between two quaternions $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ as the rotation between them:

$$\phi(\boldsymbol{q}_1, \boldsymbol{q}_2) = 2 \arccos((\boldsymbol{q}_1 \cdot \boldsymbol{q}_2^{-1})_4). \tag{5.7}$$

The grasp reward r_{grasp} is defined as:

$$r_{\text{grasp}} = r_{\boldsymbol{h}_\text{p}} + r_{\boldsymbol{h}_\text{r}} + \lambda r_{\boldsymbol{h}_\text{j}}, \tag{5.8}$$

where $r_{\boldsymbol{h}_\text{p}}$ encourages moving the hand position towards the target 3D grasp position, $r_{\boldsymbol{h}_\text{r}}$ encourages moving the hand rotation towards the target grasp rotation, and $r_{\boldsymbol{h}_\text{j}}$ encourages moving hand joint positions towards the target grasp joint positions. $\lambda \in [0, 1]$ is the grasp joint reward importance factor. Overall, the r_{grasp} reward encourages aligning the hand pose and joint positions with the target grasp pose and joint positions correspondingly. The hand position reward $r_{\boldsymbol{h}_\text{p}}$ is defined as:

$$r_{\boldsymbol{h}_\text{p}}(t) = \frac{\Delta \boldsymbol{h}_\text{p}(t-1) - \Delta \boldsymbol{h}_\text{p}(t)}{\Delta h_\text{p}^{\max}}, \quad \Delta \boldsymbol{h}_\text{p} = ||\boldsymbol{h}_\text{p}^O - \boldsymbol{g}_\text{p}^O||, \tag{5.9}$$

where $\Delta \boldsymbol{h}_\text{p}$ is the Euclidean distance from the hand position $\boldsymbol{h}_\text{p}^O$ to the target grasp hand position $\boldsymbol{g}_\text{p}^O$. $\Delta h_\text{p}^{\max}$ is a maximal hand position change during the step duration Δt: $\Delta h_\text{p}^{\max} = v_{\boldsymbol{h}_\text{p}}^{\max} \Delta t$ with $v_{\boldsymbol{h}_\text{p}}^{\max}$ being the maximal linear velocity of the hand. Given the UR5e arm that is used in this work, $v_{\boldsymbol{h}_\text{p}}^{\max} = 1\,\text{m/s}$ and $\Delta t = 0.0333\,\text{s}$.
The hand rotation reward is defined as:

$$r_{\boldsymbol{h}_\text{r}}(t) = \frac{\Delta \boldsymbol{h}_\text{r}(t-1) - \Delta \boldsymbol{h}_\text{r}(t)}{\Delta h_\text{r}^{\max}}, \quad \Delta \boldsymbol{h}_\text{r} = \phi(\boldsymbol{h}_\text{r}^O, \boldsymbol{g}_\text{r}^O), \tag{5.10}$$

where $\Delta \boldsymbol{h}_\text{r}$ is a distance from the hand rotation $\boldsymbol{h}_\text{r}^O$ to the target grasp hand rotation $\boldsymbol{g}_\text{r}^O$, calculated according to Eq. 5.7. $\Delta h_\text{r}^{\max}$ is a maximal hand rotation change during time Δt. It is defined analogously to $\Delta h_\text{p}^{\max}$. We use $v_{\boldsymbol{h}_\text{r}}^{\max} = \pi\,\text{rad/s}$.
Finally, the hand joint reward is defined as:

$$r_{\boldsymbol{h}_\text{j}}(t) = \frac{\Delta \boldsymbol{h}_\text{j}(t-1) - \Delta \boldsymbol{h}_\text{j}(t)}{\Delta h_\text{j}^{\max}}, \Delta \boldsymbol{h}_\text{j} = \frac{1}{M} \sum_{i=0}^{M} |h_{\text{j}_i} - g_{\text{j}_i}|, \tag{5.11}$$

where M is the number of controllable hand joints, $\Delta \boldsymbol{h}_\text{j}$ is an average per-joint distance to the target grasp joint positions, and $\Delta h_\text{j}^{\max}$ is a maximal joint position displacement during time Δt. It is defined similarly to the maximal position and rotation displacements through the maximal joint velocity. We use $v_{\boldsymbol{h}_\text{j}}^{\max} = \pi\,\text{rad/s}$.

The hand joint importance factor λ is defined as:

$$\lambda = \left(1 - \frac{\min(h_{\mathrm{p}}^{\mathrm{prox}}, \Delta \boldsymbol{h}_{\mathrm{p}})}{h_{\mathrm{p}}^{\mathrm{prox}}}\right)\left(1 - \frac{\min(h_{\mathrm{r}}^{\mathrm{prox}}, \Delta \boldsymbol{h}_{\mathrm{r}})}{h_{\mathrm{r}}^{\mathrm{prox}}}\right), \tag{5.12}$$

where $h_{\mathrm{p}}^{\mathrm{prox}}$ is a predefined constant, representing a proximity distance between the hand position and the target grasp position. When the displacement between the target and the hand is less than this distance, the hand joint position reward $r_{\boldsymbol{h}_{\mathrm{j}}}$ becomes active. Otherwise, $\lambda = 0$ disables the contribution of $r_{\boldsymbol{h}_{\mathrm{j}}}$, as shown in Eq. 5.8. We set the distance $h_{\mathrm{p}}^{\mathrm{prox}}$ to be equal to the length of the hand. Similarly, $h_{\mathrm{r}}^{\mathrm{prox}}$ is a rotation proximity distance; we use $h_{\mathrm{r}}^{\mathrm{prox}} = 1\,\mathrm{rad}$. Overall, incorporating λ leads to ignoring the hand joint reward when the hand is far from the target grasp pose. This facilitates using the fingers for manipulation rather than pursuing yet distant target grasp positions.
The manipulation reward r_{man} is defined as:

$$r_{\mathrm{man}} = r_{\mathrm{reach}} + r_{\mathrm{hold}} + r_{\mathrm{orient}}, \tag{5.13}$$

where r_{reach} encourages moving the hand towards the object, r_{hold} encourages holding the object in the hand, and r_{orient} encourages orienting the object towards a nominal rotation, where the target grasp is more likely to be reachable. Thus, the manipulation reward term encourages an intuitive *reach* $\rightarrow$ *hold* $\rightarrow$ *orient* behavior for the pre-grasp object manipulation. All terms in this reward function are strictly positive.
The hand reach reward is defined as:

$$r_{\mathrm{reach}}(t) = \frac{\sum_{k=1}^{K}\left(d(\boldsymbol{H}_{\mathrm{p}_k}(t-1)) - d(\boldsymbol{H}_{\mathrm{p}_k}(t))\right)}{\Delta h_p^{\max}}, \tag{5.14}$$

where d is a function that takes a set of 3D points and returns signed distances from the points to the object surface, utilizing the precomputed object SDF. $\Delta h_{\mathrm{p}}^{\max}$ is a maximal position displacement, defined in Eq. 5.9. $\boldsymbol{H}_{\mathrm{p}}$ is a set of K 3D points located between the thumb and the other fingers, described in detail below. In the context of this reward, these points guide the hand towards a position where the object is contained between the thumb and the other fingers, which is advantageous for manipulation.
The object hold reward is defined as:

$$r_{\mathrm{hold}} = \frac{1}{K}\sum_{k=1}^{K}\frac{d(\boldsymbol{H}_{\mathrm{p}_k}) - \rho}{d_k^{\max}}, \tag{5.15}$$

where ρ is a predefined constant radius of spheres with positions $\boldsymbol{H}_{\mathrm{p}}$ as centers and $d_k^{\max}$ is a per-point maximum possible distance from the point to the closest finger surface. The set of hold-detect points $\boldsymbol{H}_{\mathrm{p}}$ is positioned between the tip of the thumb and between the tip

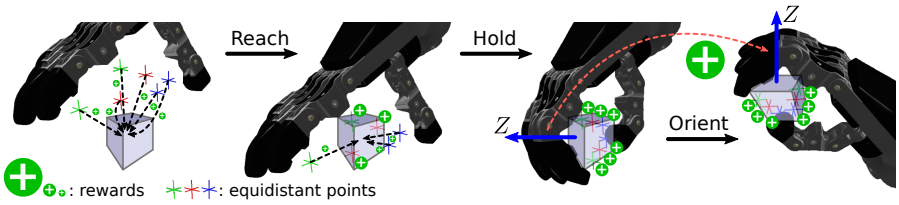


Figure 5.4: Manipulation reward r_{man} is composed of three components: reach, hold, and orient, representing a sequence of interconnected tasks. Equidistant points between the thumb tip and middle fingertip & center, used to query distances to the object, are red, blue, and green crosses. Rewards: plus signs, size is proportional to the reward value. First, the motion of the equidistant points to the object surface is rewarded by the reach reward. Second, equidistant points that are inside the object yield a bigger hold reward. Finally, orienting the object towards the nominal orientation yields an even bigger reward. Note that closing the hand brings the equidistant points closer together, often pushing them inside the object. Such design implicitly rewards grasping behaviors without using expensive contact information or explicitly rewarding specific movement primitives.

and middle of the rest of the fingers. Thus, points between fingertips represent positions where objects can be pinch-grasped, and points between the thumb tip and middles of the fingers represent positions where objects are grasped more securely. Each direction tip $\rightarrow$ tip or tip $\rightarrow$ middle of a finger has three equidistant points. This ensures a positive response when an object is positioned between the thumb and other fingers imperfectly.

When the hand closes, the equidistant points come closer to each other, which promotes closing the hand around an object. Note that the maximum r_{hold} is achieved when fingers evenly embrace the object, which naturally resembles a grasp. For simplicity, we use only the thumb $\rightarrow$ middle finger lines in this work, which yields six equidistant points, as shown in Fig. 5.4.

The intuition behind this design choice is that in the case when an object is contained between the middle finger and the thumb, it is also contained between the index and the ring fingers, as defined by the hand topology. While it is straightforward to utilize several finger pairs at the same time, we observed in practice that using only middle $\rightarrow$ thumb lines for this reward term was sufficient to learn grasping behaviors. At the same time, having all other fingertips close to the object is simultaneously encouraged by the reach reward term r_{reach}. The object orient reward is defined as:

$$r_{\text{orient}}(t) = \frac{\Delta \boldsymbol{o}_{\text{r}}(t-1) - \Delta \boldsymbol{o}_{\text{r}}(t)}{\pi}, \Delta \boldsymbol{o}_{\text{r}} = \phi(\boldsymbol{o}_{\text{r}}, \boldsymbol{o}_{\text{r}}^{\text{nominal}}), \tag{5.16}$$

where $\Delta \boldsymbol{o}_{\text{r}}$ is the distance from the object rotation to the nominal object rotation $\boldsymbol{o}_{\text{r}}^{\text{nominal}}$. A nominal rotation resembles a natural object

orientation as intended for functional use: the object Z-axis points upwards, and the object X-axis (the direction of the tool tip) points away from the hand. Although there are many other feasible object orientations to perform a functional grasp, we find that such definition is generic and not too biased. In practice, it provides good guidance on how to reorient an object when it is in a state where a direct functional grasp is not possible. Together, the reach, hold, and orient rewards represent a sequence of interconnected tasks that help to steer the policy towards dexterous manipulation behaviors (Fig. 5.4).
The manipulability penalty reward is defined as:

$$r_{\text{MP}} = 1 - 2 \Big/ \left(1 + \left(\frac{\min(|\boldsymbol{J}|, |\boldsymbol{J}|_{\max})}{|\boldsymbol{J}|_{\max}}\right)^3\right), \tag{5.17}$$

where $|\boldsymbol{J}|$ is a determinant of the end-effector Jacobian $\boldsymbol{J}$ and $|\boldsymbol{J}|_{\max}$ is a maximum determinant value that is penalized. We define $|\boldsymbol{J}|_{\max}$ to be 15% of maximal observed $|\boldsymbol{J}|$ across 100 random arm configurations. This reward penalizes coming close to singularities and leads to learning more intuitive motions.
Finally, the target grasp reward is defined as:

$$r_T = \begin{cases} 1 & \text{if } \Delta \boldsymbol{h}_{\text{p}} < T_{\text{p}} \wedge \Delta \boldsymbol{h}_{\text{r}} < T_{\text{r}} \wedge \Delta \boldsymbol{h}_{\text{j}} < T_{\text{j}} \\ 0 & \text{otherwise,} \end{cases} \tag{5.18}$$

where $T_{\text{p}}, T_{\text{r}}, T_{\text{j}}$ are the distance thresholds for hand position, rotation, and hand joint positions to the target grasp. They define the accuracy with which the target grasp has to be achieved. We use $T_{\text{p}} = 1\,\text{cm}$, $T_{\text{r}} = 0.15\,\text{rad}$, and $T_{\text{j}} = 0.1\,\text{rad}$. The episode ends when the target grasp is reached, as defined by the discussed thresholds.

In the proposed reward function, we widely use the differential distances. Compared to directly using the velocities, such an approach naturally avoids learning overshooting behaviors, which oftentimes may slow down the convergence. Note that all reward terms are defined in a generic way and can be easily configured for an arbitrary robotic arm, hand, and a set of rigid objects. All reward components are defined to be in the interval $[-1, 1]$ or $[0, 1]$. This allows applying relative scaling easily. For the best performance, we scale the rewards inverse-proportionally to the frequency of their achievement: $r_T \gg r_{\text{orient}} \gg r_{\text{hold}} \gg r_{\text{reach}}$. Reward terms that are harder to achieve receive a larger scaling factor. We leave the other reward components unscaled. This reduces the probability that the policy gets stuck in the local minima created by accumulating rewards granted for actions that can be achieved easier than the final goal.

To summarize, the multi-component reward function can be split into three terms:

1. the grasp reward r_{grasp} encourages reaching the given functional grasp,

2. the manipulation reward r_{man} encourages reaching, holding, and reorienting the object, and

3. the manipulability penalty reward r_{MP} penalizes being close to singularities and thus helps to avoid unintuitive behavior.

Each component is a continuous dense reward. The components combine to effectively guide the policy towards learning a robust, dexterous pre-grasp manipulation. Finally, a sparse component r_T rewards reaching the target grasp.

5.4.4 *Curriculum*

In this work, we avoid having any explicit expert demonstrations and focus on learning robust and natural policies for object pre-grasp manipulation through pure DRL with dense reward shaping. To facilitate faster and more stable learning, we propose a simple two-stage curriculum. In the first stage, we place the objects in poses where target functional grasps can be reached directly. After that, the second stage has full difficulty, taking advantage of the warm-start provided by the first stage.

During the first stage, the objects are positioned on the table in their nominal poses 5 cm away from the inner side of the hand. The arm is set to a neutral configuration with a high manipulability score. We disable the r_{man} reward term during the first stage so that the policy can converge faster without being stuck in potential multiple local minima. Note that this curriculum is agnostic to object-specific details. Thus, we keep the approach general while achieving faster policy convergence.

5.5 CONSTRAINT-BASED TARGET GRASP REPRESENTATION

In Section 5.4 we presented a DRL problem formulation with an explicit target grasp representation. Alternatively, a target functional grasp can be represented using a constraint that defines the grasp as functional. For instance, with a drill, such a constraint might be positioning the index fingertip on the trigger to make the activation of the drill possible. In this section, we propose a methodology for learning a pre-grasp manipulation policy using a constraint-based target grasp representation. Specifically, we represent the target grasp as a 3D target position of the index fingertip and the end-effector rotation in the object frame of reference.

This representation offers two advantages. First, it allows the agent to explore various grasp configurations that satisfy the given target functional grasp constraint. This facilitates the learning combinations of manipulation strategies and grasp configurations end-to-end. Second, it relaxes the requirements for an external oracle that has to

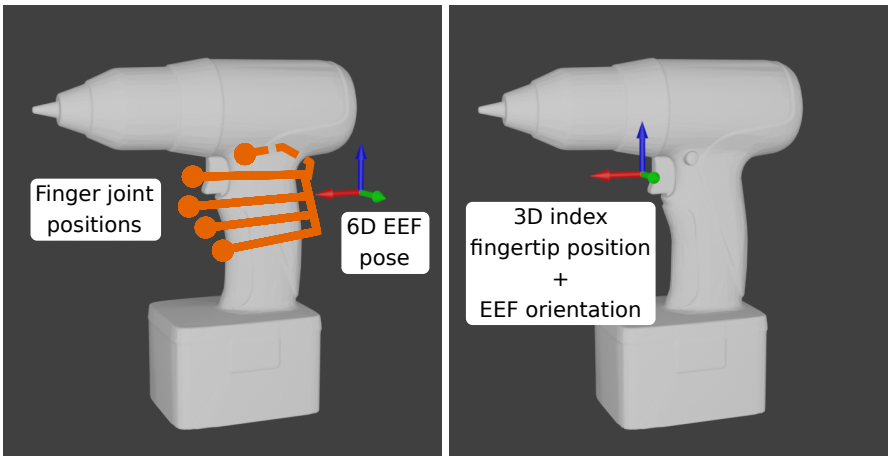


Figure 5.5: Two proposed target grasp representations. *Left:* Explicit grasp representation, consisting of 6D end-effector pose and finger positions. *Right:* Constraint-based representation. The grasp is represented with a 3D position of the index fingertip and end-effector orientation. Note that this representation allows the policy to explore different grasp configurations, satisfying the constraint.

provide the target grasp. This is because identifying keypoints like the trigger and the desired end-effector orientation is easier than specifying a full explicit grasp configuration.

We argue that both grasp representations require the orientation of the object. Additionally, they both need two distinct 3D positions. These include the object position in one case and a specific point of interest in the other. However, the explicit grasp representation also requires the exact joint angles of all fingers. This additional requirement introduces more possibilities for errors. Consequently, it increases the likelihood of failed grasps. This is especially important when dealing with previously unseen object instances. Defining specific joint angles for each finger is more challenging than defining a 3D position of the point of interest on the object, such as a trigger of the drill, spray bottle, or handle of a mug. The downside of this approach is a more complex learning pipeline. Both target grasp representations are shown in Fig. 5.5.

5.5.1 *Action Space*

We keep the action representation unchanged, as in Section 5.4.1. The 6D pose of the end-effector is controlled through the IK, and finger positions are controlled directly in joint space through a coupled-joints embedding. Such action representation is generic and fits learning with constraint-based target grasp representation well.

5.5.2 *State Space*

The state representation is almost identical to the one described in Section 5.4.2. The grasp representation part is changed to reflect the constraint target grasp representation, addressed in this section. The desired functional grasp is thus provided as a column vector:

$$\boldsymbol{g} = [\boldsymbol{g}^O_{\text{ifp}}, \boldsymbol{g}^O_{\text{r}}], \tag{5.19}$$

where $\boldsymbol{g}^O_{\text{ifp}}$ is a 3D index fingertip position in the object frame of reference, $\boldsymbol{g}^O_{\text{r}}$ is a 4-element hand rotation vector represented by a quaternion. Thus, the target functional grasp is represented implicitly through a functional grasp constraint with a 7-element vector. The full state is a 52-element vector.

5.5.3 *Reward Function*

To reflect the semantic changes introduced by the constraint-based target grasp representation, we change the definition of a successfully achieved functional grasp from Eq. 5.18 accordingly:

$$r_T = \begin{cases} 1 & \text{if } \Delta \boldsymbol{i}_{\text{p}} < T_{\text{p}} \wedge \Delta \boldsymbol{h}_{\text{r}} < T_{\text{r}} \wedge o_{\text{p}_z} > T_z \\ 0 & \text{otherwise,} \end{cases} \tag{5.20}$$

where $\Delta \boldsymbol{i}_{\text{p}}$ is the distance from the current index finger position $\boldsymbol{i}_{\text{p}}$ to the desired one, $\Delta \boldsymbol{h}_{\text{r}}$ is the distance from the current end-effector rotation to the desired one, and o_{p_z} is the Z coordinate of an object. We use $T_{\text{p}} = 1\,\text{cm}$, $T_{\text{r}} = 0.15\,\text{rad}$, and $T_z = z_{\text{table}} + z_{\text{offset}}\,\text{cm}$, where z_{table} is a height of the surface of the table and $z_{\text{offset}} = 15\,\text{cm}$ is an offset chosen such that any object from the dataset positioned at such height above the table can not touch the surface of the table.

The last condition requires an object to be lifted off the table. This is necessary because without it the policy could learn to satisfy the functional grasp constraint without achieving a stable grasp. Thus, by accepting only grasps that are having an object lifted off the table, we introduce an implicit grasp stability constraint. Note that it is not necessary in the case of an explicit target grasp representation. In that case we assume that provided explicit grasps are feasible, allowing for lifting and using the object. This, however, is one of the drawbacks of the explicit target grasp representation, as it adds more responsibility to the external oracle. In contrast, in constraint-based grasp representation, this responsibility is transferred to the policy.

Given the modifications from above, we modify the reward function from Eq. 5.6 accordingly:

$$r(t) = r_{\text{grasp}}(t) + r_{\text{lift}}(t) + r_{\text{man}}(t) + r_{\text{MP}}(t) + r_T(t), \tag{5.21}$$

where r_{grasp} encourages movement towards target grasp $\boldsymbol{g}$, r_{man} encourages pre-grasp manipulation of an object, r_{MP} penalizes being in configurations with low manipulability, and r_T rewards reaching the target functional grasp $\boldsymbol{g}$, and the new term r_{lift} encourages lifting an object off the table:

$$r_{\text{lift}}(t) = \min(\max((o_{\text{p}_z} - z_{\text{table}})/z_{\text{offset}}, 0), 1). \tag{5.22}$$

The term $r_{\text{lift}} \in [0, 1]$. Not having negative rewards allows the agent to explore freely. Not having an increment-based reward (as in Eq. 5.9) prevents the agent from maximizing the reward by simply repeatedly moving an object up and down.

Finally, the grasp reward term r_{grasp} is modified such that:

$$r_{\text{grasp}} = r_{\boldsymbol{i}_{\text{p}}} + r_{\boldsymbol{h}_{\text{r}}}, \tag{5.23}$$

where $r_{\boldsymbol{i}_{\text{p}}}$ encourages moving the index fingertip position towards the target index fingertip position, $r_{\boldsymbol{h}_{\text{r}}}$ encourages moving the hand rotation towards the target grasp rotation. Both position and rotation are in the object frame of reference. The $r_{\boldsymbol{i}_{\text{p}}}$ term is defined in the same way as Eq. 5.9:

$$r_{\boldsymbol{i}_{\text{p}}}(t) = \frac{\Delta \boldsymbol{i}_{\text{p}}(t-1) - \Delta \boldsymbol{i}_{\text{p}}(t)}{\Delta i_{\text{p}}^{\max}}, \tag{5.24}$$

where $\Delta \boldsymbol{i}_{\text{p}}$ is the Euclidean distance from the index fingertip position $\boldsymbol{i}_{\text{p}}^{O}$ to the target grasp index fingertip position $\boldsymbol{g}_{\text{p}}^{O}$. $\Delta i_{\text{p}}^{\max}$ is a maximal index fingertip position change during the step duration Δt, which is computed as a sum of maximal possible end-effector velocity and maximal possible index fingertip velocity relative to a hand.

Overall, the reward function has the same structure and ideas as described in Section 5.4.3, with modifications that reflect a more compact and abstract target grasp representation through an index fingertip position constraint. All reward terms are dense and continuous, defined to be in the range $[-1, 1]$ or $[0, 1]$, such that relative reward component scaling is straightforward.

5.5.4 *Curriculum*

Lifting an object above the table introduces an additional complexity on the way to learning a meaningful policy. To compensate for that and keep the learning time short, we modify the curriculum described in Section 5.4.4 by introducing an additional curriculum step. The three-step curriculum now is formulated as follows:

1. Learning how to reach the target grasp without lifting an object. The objects are spawned upright in nominal configuration, very

close to the hand. Lifting an object is excluded from the success criterion formulation.

2. Learning how to reach the target grasp and lift an object. The hand is further away and lifting the object is required. Otherwise this step is the same as Step 1.

3. Learning the complete objective of the pre-grasp manipulation. The objects are initialized in any combination of roll and yaw on the table.

The three-step curriculum subdivides the given learning problem into stages of gradually increasing difficulty while maintaining a generic formulation that is straightforward to apply to arbitrary objects, arms, and hands.

5.6 EVALUATION

To evaluate the proposed approach, we apply it to the 6 DoF UR5e robotic arm with the attached 11 DoF Schunk SIH hand. The joints of this wire-driven hand are coupled, leaving 5 controllable DoF. In this evaluation we try to answer the following questions:

- Does our approach reliably produce robust manipulation policies, capable of dexterous pre-grasp manipulation of unseen object instances of a known category?
- Does the multi-component manipulation reward r_{man} lead to policies with higher success rates?
- Does the curriculum improve convergence stability?
- Does our approach enable learning of feasible manipulation policies for both target grasp representations?

5.6.1 *Setup*

We use PPO (Schulman, Wolski, et al., 2017) to train the policies. We employ the RL Games (Makoviichuk and Makoviychuk, 2021) high-performance implementation for GPU parallelization. We use the findings of Mosbach and Behnke, 2022 as a base, and keeping the learning algorithm hyperparameters the same, with 5×10^{-4} learning rate and discounting factor $\gamma = 0.95$. The policy is represented by a three-layer fully-connected NN. In our case, the input is a 57-element vector. The network is a MLP and has the following structure:

$$57 \times 512 \rightarrow 512 \times 256 \rightarrow 256 \times 128 \rightarrow 128 \times 11.$$

In our experiments, we pursue the objective of learning a single functional grasping policy for three rigid object categories: drills, spray bottles, and mugs. To this end, we prepared a mesh dataset of

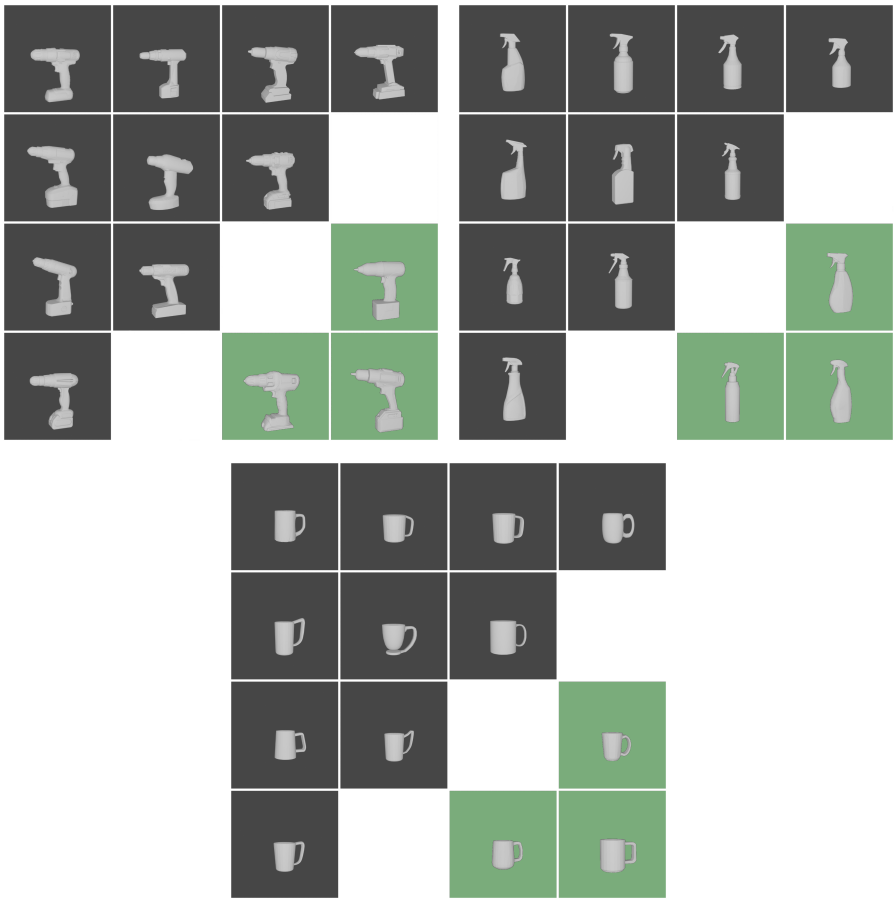


Figure 5.6: Functional grasping dataset with three object categories: drills, spray bottles, and mugs. Each category has 10 train objects (gray background) and 3 test objects (green background).

39 objects: 13 of each category, where ten objects are for training and the remaining three objects are used for testing (Fig. 5.6). The dataset is composed of meshes from (Rodriguez, Di Guardo, et al., 2018) and of meshes available online[1]. The dataset is available online[2].

We select these three specific object categories as they represent objects that are functionally grasped in three different ways relative to their center of mass (CoM). Drills are functionally grasped roughly at the CoM. Spray bottles are functionally grasped far from the CoM along the Z axis of the object. Finally, mugs are grasped far from the CoM along the X axis of the object. At the same time, these categories have diverse geometrical shapes. Drills have complex shapes with multiple graspable regions. Spray bottles are elongated, making them difficult to position upright. At the same time, some of them feature smooth cylindrical shapes that can be easily rolled, while the others resemble parallelepiped-like shapes. Finally, mugs feature a design with two distinct cavities: a large, open space for containing liquid

1 https://free3d.com, https://3dsky.org
2 https://github.com/AIS-Bonn/fun_cat_grasp_dataset

and a smaller, through-passage formed by the handle. These three categories differ fundamentally in their geometry and functional grasp regions. These differences make them suitable candidates for forming a compact but diverse dataset. Due to the high variability in grasping strategies, this dataset requires a certain level of generalization.

In this work, we use the high-performance GPU physics simulator Isaac Gym (Makoviychuk et al., 2021). The experiments are performed on a single NVIDIA RTX A6000 GPU with 48 GB of VRAM.

We assume that the objects are located on a table in front of the robot. Thus, there are three possible natural poses in which drills, spray bottles, or mugs can be. They include standing upright and lying on their left or right side. All other possible poses on a flat surface are unstable and transition quickly to one of the described poses. Mugs can additionally be positioned upside down. However, we do not use this configuration in our experiments to ensure that the results are comparable across all object categories.

Actions are generated with a frequency of 30 Hz. The objects are positioned on the table in front of the robot, such that at least 75% of their bounding box is in the manipulation workspace. Poses in which objects are lying on their sides are the most challenging for functional grasping because of the occlusion created by the table. For this reason, we focus on such poses and use the following object rotation distribution: 20% of the objects are upright, 40% are on their left side, and 40% are on their right side. The yaw angle and the object position are sampled uniformly. The hand starts at a random 6D pose above the table. Notably, objects lying on the right side require more complex pre-grasp manipulation for functional grasping with the right hand. Learning is performed on the training set of 30 objects. A target functional pre-grasp and the constraint for the index fingertip are manually defined for each object.

To make the simulation setup more realistic, Gaussian noise is applied to all observations supplied to the policy. For positions and distances, the zero-mean noise has a standard deviation $\sigma = 3\,\text{mm}$. For rotations, the zero-mean noise has a standard deviation $\sigma = 5°$. Before each action is generated, the noise values are drawn from the distribution specified above and added to the ground truth values from the simulation. The only two observations that do not have noise are the object category and the target grasp.

In each environment, an object is assigned a realistic random mass. The mass distribution in kg per category is represented by a Gaussian: $\mathcal{N}(1.4, 0.2)$ for drills, $\mathcal{N}(0.5, 0.15)$ for spray bottles, and $\mathcal{N}(0.3, 0.07)$ for mugs. Both noise and mass are limited to deviate from the mean for not more than 3σ. The new mass values are set before each episode.

We keep the reward value for reaching the target grasp $r_T = 5000$, as it is the default value in the RL Games framework. We scale the reward components: orienting reward r_{orient} by 500 and holding reward r_{hold}

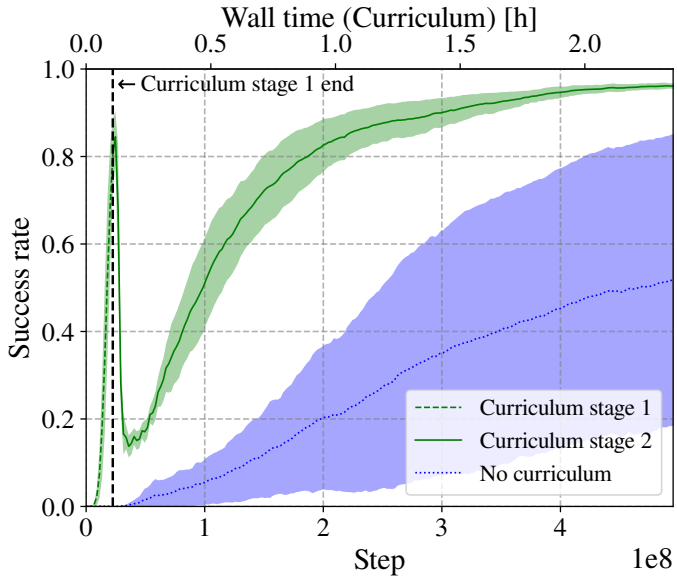


Figure 5.7: Training curves in the curriculum ablation experiment for explicit target grasp representation. The two-stage curriculum significantly improves convergence stability compared to the runs without the curriculum. Lines: means. Colored areas: 95% confidence intervals.

by 25. The other reward components are not scaled. Both scaling factors are chosen to be one order of magnitude less than the final reward and the corresponding reward component with the higher scaling. The order of scaling is defined by the desired action sequence: first the hand approaches the object (scale 1), then it gains control over the object by holding it (scale 25), then it orients the object (scale 500), and finally, after repositioning the object, the target grasp is achievable (scale 5000).

Switching the order of scaling often hinders the progress of learning to achieve the target grasp. That is because the policy greedily maximizes reward through, for example, holding an object, rather than exploring more difficult and failure-prone orienting the object if it is yielding less reward. The exact values of these scaling factors are not affecting the learning speed significantly, as long as the overall proportions reflect the desired logical sequence: approach, hold, orient, grasp.

5.6.2 *Explicit Target Grasp Representation*

In this section, we evaluate the approach using an explicit target grasp representation that is presented in Section 5.4. We train the policy on a single GPU with 16,384 parallel environments. Each policy in this evaluation is trained three times with three different seeds to assess convergence stability. An episode terminates when: (i) a target functional pre-grasp is reached, (ii) an object falls from the table, or

(iii) a maximum number of steps is reached. We set the maximum number of steps to 200, which corresponds to $\approx$6.7 s.

We assume that the provided explicit target pre-grasps are valid and enable lifting and manipulating an object. Thus we do not require the policy to lift objects off the table once the grasps are achieved during the training stage. First, we perform an ablation study of the two-stage curriculum proposed in Section 5.4.4. During the first stage, the objects are positioned with a nominal rotation and close to the hand. Since at this stage the grasp is easily reachable, we disable the manipulation reward r_{man} to ensure quicker convergence. The first stage continues until at least a 50% success rate is achieved for each object. During the second stage, the objects are spawned with the rotations described above, and the full reward is used.

Fig. 5.7 shows training curves obtained with and without the curriculum. In addition, the wall time is shown for the curriculum runs. The wall time of the other runs is similar ($\pm$10 min). The first stage of the curriculum is completed quickly. The second stage takes longer, but all three runs reliably converge and achieve a success rate of 97% in under three hours with little variance.

Without the curriculum, the policy achieves only $\approx$50% success rate and has a large variance within runs. Hence, the two-stage curriculum significantly improves convergence speed, stability, and the resulting success rate. We use a default discounting factor $\gamma = 0.95$ in all experiments. Lower values, such as $\gamma = 0.9$, decrease the learning speed significantly. This is explained by the fact that in such cases, only short time spans are represented in rewards, leaving out longer-term consequences of complex manipulation. We observed that higher values, such as $\gamma = 0.975$, did not provide any significant improvement.

Next, we perform an ablation study of the proposed reward function, analyzing the contributions of each individual component. We train five policy variants. First, the proposed method with a full reward. Second, a variant with a disabled reward component encouraging moving the hand towards the object r_{reach}. Third, a variant with a disabled reward component encouraging holding the object r_{hold}. Fourth, a variant with a disabled reward component encouraging rotating the object towards the nominal rotation r_{orient}. And finally, the fifth variant, where the whole manipulation component $r_{\text{man}} = r_{\text{reach}} + r_{\text{hold}} + r_{\text{orient}}$ is disabled.

Each variant is trained with three different seeds and the same hyperparameters as in the curriculum experiments, as described before. Fig. 5.8 shows training curves for this ablation study. One can observe that when a single component of the manipulation reward is disabled, the policy learns to achieve the goal slower but still reliably makes progress towards a high success rate.

The most influential factor is the removal of the r_{hold} component. Without r_{hold}, the policy has the highest variance within the runs and

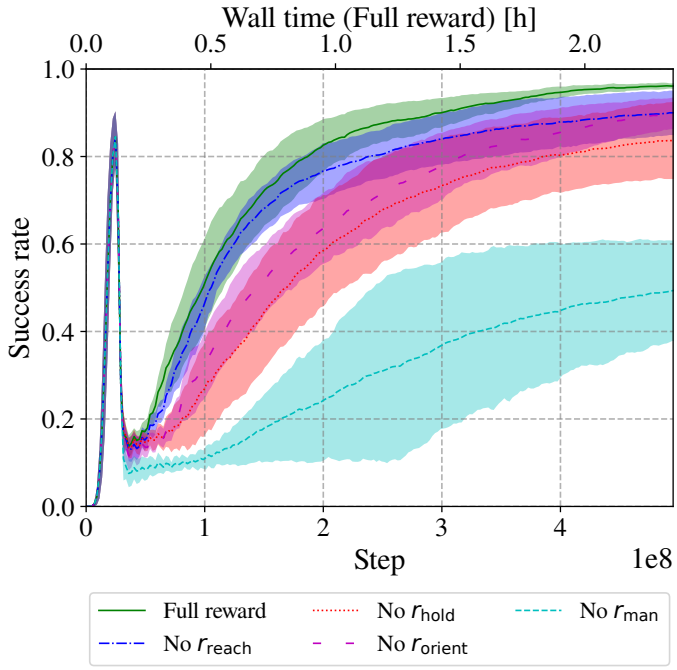


Figure 5.8: Training curves in the manipulation reward ablation experiment for explicit target grasp representation. Disabling individual reward components slightly reduces the convergence rate and stability. Disabling the whole manipulation reward component makes the learning process significantly slower and less stable. Lines: means. Colored areas: 95% confidence intervals.

achieves the lowest success rate among the single-component ablations. This shows that the reward component encouraging holding behavior is the most important within the proposed manipulation reward.

A deteriorated but still reliable convergence without single reward terms suggests that although each component is important, the formulation is generic enough to not depend on every detail of the reward function. In contrast, disabling the whole manipulation reward r_{man} has a drastic negative effect on the performance of the policy. Although it achieves a success rate of $\approx 50\%$, it struggles to learn a robust behavior for objects in difficult configurations. Overall, this ablation study demonstrates that the proposed manipulation reward component significantly speeds up the learning of dexterous pre-grasp manipulation.

To evaluate the generalization capability of the learned policy, we measure the success rate of the policy learned with a curriculum and full reward on the training and test sets. The training set consists of 30 objects, ten for each of the three categories. The test set consists of nine novel objects of the known categories. We perform 100 grasping attempts for each object. This results in 3000 attempts for the training set and 900 attempts for the test set. Object initial poses are sampled as during learning: 20% upright, 40% on the left side, and 40% on the right side.

Table 5.1: Average success rates per category in % with the explicit target grasp representation.

Category	Training set	Test set
Drills	96.0 ± 1.2	94.3 ± 2.6
Spray bottles	97.7 ± 0.9	92.3 ± 3.0
Mugs	99.3 ± 0.5	95.6 ± 2.3
Σ	97.7 ± 0.5	94.1 ± 1.5

*Mean $\pm$ 95% confidence interval is shown.

Once the target pre-grasp is reached, the success is tested by closing the hand. If the object stays in the hand and the key condition of a functional grasp, such as an index finger on the trigger, is satisfied, an attempt is considered successful. We allocate 300 steps or 10 s per episode.

The observed success rates for all object categories are reported in Table 5.1. On the training set, the learned policy shows a high success rate of 97.7%. As expected, on the test set the success rate is lower, but still a high value of 94.1%. The highest success rates are achieved for mugs. This is because they are relatively easy to flip over from the side position and have a simple geometry. The hardest object category is the spray bottles. This is because spray bottles are narrow, have a high CoM, and can be tipped over easily.

Fig. 5.9 shows example rollouts of the policy for three objects from the test set. One can observe that dexterous interactive pre-grasp manipulation has been learned that leads to functional grasps for all three object categories. Videos of the learned interactive functional grasping behaviors are available online[3]. It can be seen that complex pre-grasping strategies such as repositioning the object, reorienting and up-righting, and re-grasping have been attained. The policy also learned to reattempt the grasping in case of failures.

5.6.3 *Constraint-based Target Grasp Representation*

In this section, we evaluate the approach using a more abstract constraint-based target grasp representation, presented in Section 5.5. The policy is represented with the same model as in the experiments with an explicit target grasp representation (Section 5.6.2). The training procedure, simulation setup, and hyperparameters are also identical. An episode is terminated when: (i) a provided target constraint, defining the functional grasp, is satisfied and the object is lifted off the table, (ii) an object falls from the table, or (iii) a maximum number of 200 steps is reached.

3 https://www.ais.uni-bonn.de/videos/TASE_2024_Pavlichenko

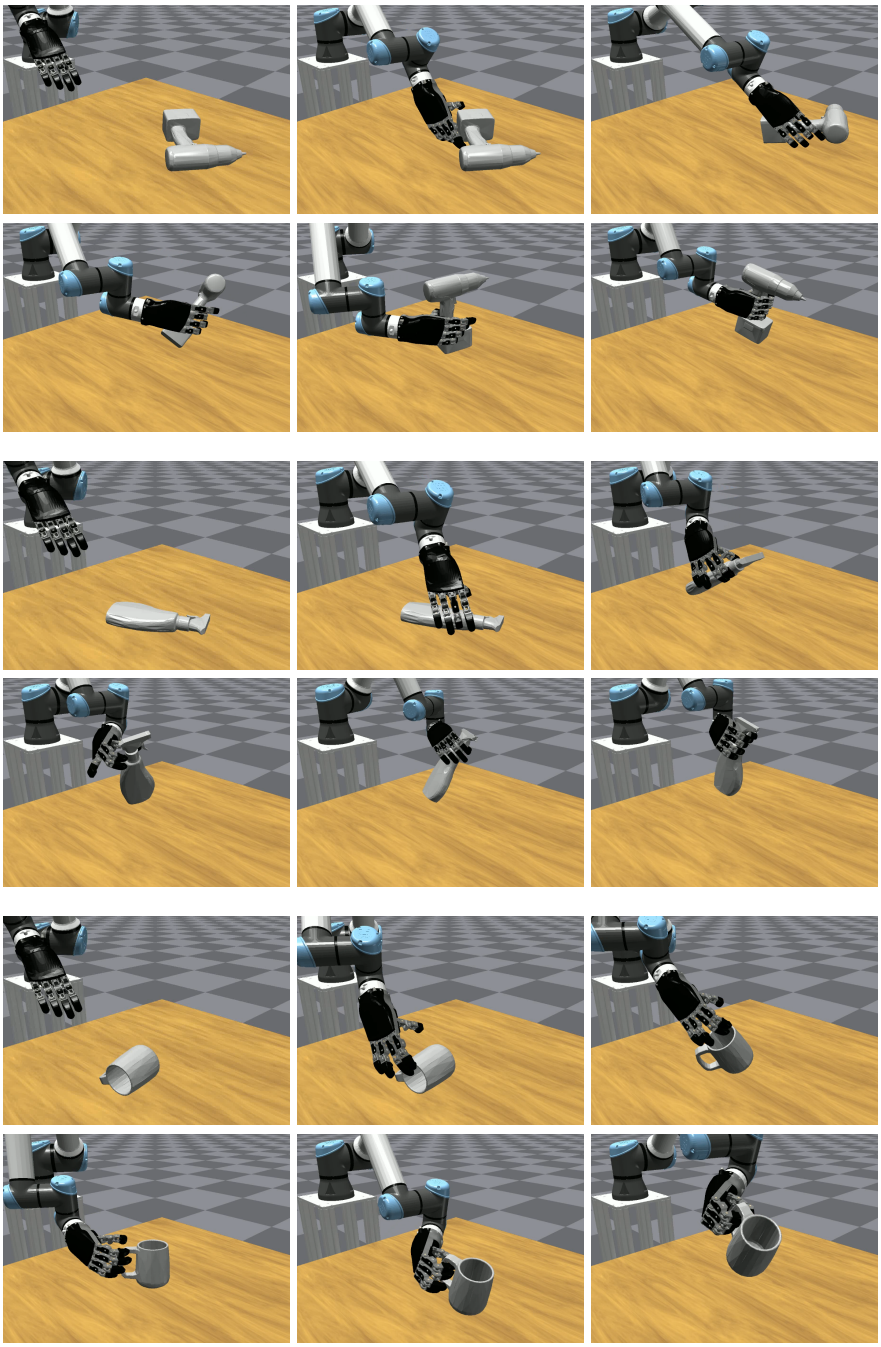


Figure 5.9: Rollouts of policy manipulating unseen objects of known categories, positioned in a way that a direct functional grasp is impossible. An explicit target grasp representation is used. Top to bottom: drill, spray bottle, and mug. The policy demonstrates advanced capabilities in repositioning and reorienting objects. Note the functional grasps achieved in the end.

First, we perform an ablation study of the proposed three-stage curriculum described in Section 5.5.4. The first stage is dedicated to learning to satisfy given functional grasp constraints while the objects are in easily accessible configurations. The second stage requires the

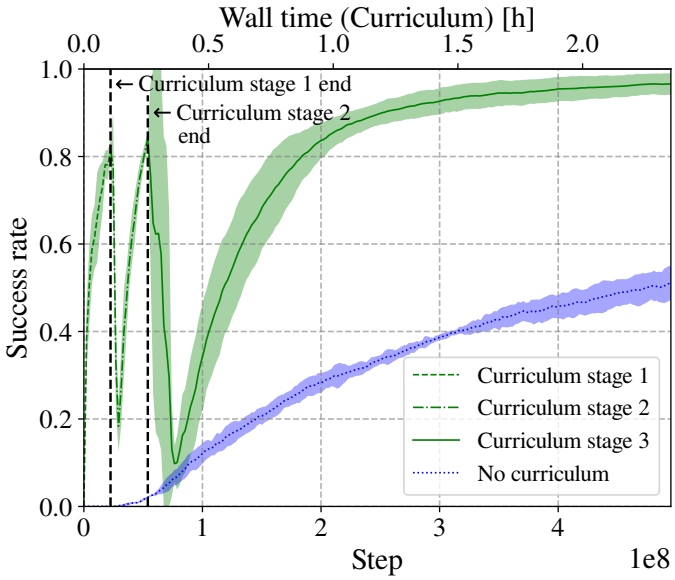


Figure 5.10: Training curves in the curriculum ablation experiment with constraint-based target grasp representation. The three-stage curriculum significantly improves learning speed. Lines: means. Colored areas: 95% confidence intervals.

policy to lift the objects off the table. With this stage, we implicitly enforce the policy to learn reliable grasp configurations. At the third stage, the objects are positioned in configurations where a direct functional grasp cannot be reached. The training curves averaged over the three runs are shown in Fig. 5.10.

One can see that with the curriculum, the policy reliably converges to a success rate of $\approx 93\%$ in under three hours, with the two first stages of the curriculum being completed quickly. In contrast, without the curriculum, the policy reaches only a subpar success rate of $\approx 50\%$. It is worth noting that there is not much variance between runs without the curriculum compared to a lot of variance between runs without the curriculum in the case of an explicit target grasp representation (Fig. 5.7). We attribute it to the fact that the learning problem has higher difficulty in the case of the constraint-based target grasp representation. The policy has to find on its own a way to grasp the objects to enable successful lifting. Given that, the probability of having a lucky run where the policy learns how to manipulate and grasp all different categories of objects without the curriculum is much lower compared to runs with an explicit target grasp representation.

Additionally, we also perform an ablation study of the proposed multi-component reward function in the context of the constrained target grasp representation. We train five policy variants. First, a variant with a full reward. Second, a variant with a disabled reward component r_{reach}, encouraging moving the hand towards the object. Third, a variant with a disabled reward component r_{hold}, encouraging holding the object. Fourth, a variant with a disabled reward component

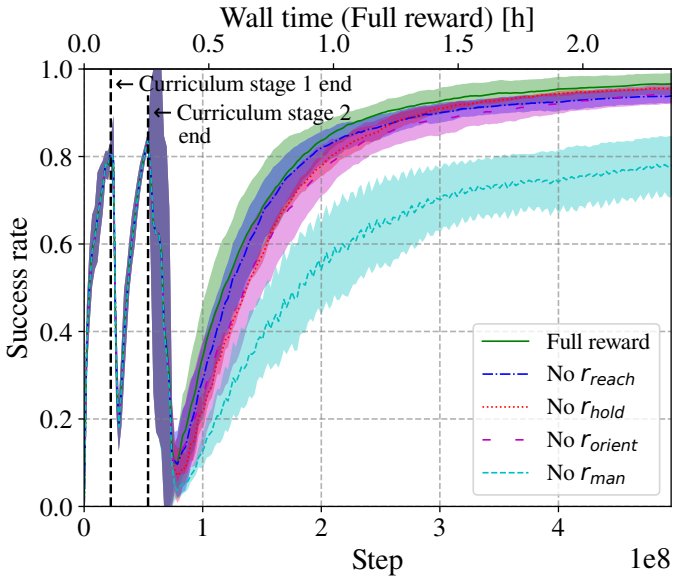


Figure 5.11: Training curves in the manipulation reward ablation experiment with constraint-based target grasp representation. Disabling individual reward components slightly decreases the convergence rate and stability. Disabling the whole manipulation reward component makes the learning process slower and less stable. Lines: means. Colored areas: 95% confidence intervals.

r_{orient}, encouraging rotating the object towards the nominal rotation. Fifth, a variant, with the whole manipulation component $r_{\text{man}} = r_{\text{reach}} + r_{\text{hold}} + r_{\text{orient}}$ being disabled.

Fig. 5.11 shows the training curves for this ablation study. One can see that, similar to Fig. 5.8, all ablation runs yield worse success rates compared to the policies with full reward. However, the difference is less significant compared to the explicit grasp representation ablation. While disabling the whole manipulation reward component significantly decreases learning speed, disabling individual components affects the learning performance only slightly. We attribute this to the fact that in the case of the constraint-based target grasp representation, the policy has to figure out the way to grasp objects by itself, forced by the requirement to lift the objects. While doing that, the policy implicitly acquires holding behaviors. Nevertheless, the ablation study demonstrated that the proposed manipulation reward facilitates quicker and more stable convergence for policies learned with the constraint-based grasp representation as well.

To quantitatively evaluate the learned policy, we perform 100 attempts for each object from the training set and for novel instances of known categories in the test set, same as in Section 5.6.2. This results in 3000 attempts for the training set and 900 attempts for the test set. The resulting success rates are shown in Table 5.2. As expected, the policy has a higher success rate on the training set. On the novel instances from the test set, the policy achieves a 90% success rate,

Table 5.2: Average success rates per category in % with the constraint-based target grasp representation.

Category	Training set	Test set
Drills	92.7 ± 1.7	90.3 ± 3.3
Spray bottles	93.8 ± 1.7	88.6 ± 3.5
Mugs	92.0 ± 1.8	91.1 ± 3.1
Σ	92.8 ± 1.0	90.1 ± 1.9

*Mean $\pm$ 95% confidence interval is shown.

which is slightly lower than 94% in the case when an explicit target grasp representation is used (Table 5.1).

We attribute it both to a more difficult task, which includes lifting an object, as well as to a more abstract grasp representation. However, such a policy has the advantage of relaxed requirements for an external oracle that provides grasp targets, compared to the policy with an explicit target grasp representation. In particular, in our experiments we had to put a lot of effort into specifying feasible explicit target grasps in the previous experiment, while defining the target constraints in this experiment was quick and straightforward.

Example policy rollouts can be seen in Fig. 5.12. The policy learns complex repositioning and reorienting behaviors to eventually satisfy the given target grasp constraints and successfully lift the objects. Close-up snapshots of the achieved grasps using policies trained with different grasp representations can be seen in Fig. 5.13. Note that for the explicit grasp representation, the targets are carefully designed by hand. In addition, after the target pre-grasp is reached, the hand is closed and the object is lifted to confirm success. Such an approach is chosen since it is extremely challenging to provide an explicit grasp pose that has all fingers positioned perfectly and maintains consistent tight contact with the object.

In the case of the constraint-based grasp representation, the policy has to learn grasps that enable lifting the objects on its own. One observation is that in the case of the constraint-based grasp representation, the spray bottles and mugs have the middle finger fully extended. For mugs, our observation is that the policy often uses the middle finger to support the mug from the side, also tilting the hand to the right side to facilitate such a supporting approach. For spray bottles, our intuition is that it is challenging to place the middle finger below the trigger. While trying to do so, the target index fingertip position may be disturbed, resulting in smaller reward.

It is worth noting that the policy learns natural human-like ways to grasp the objects without having any explicit instructions on how to do so. We attribute it to a generic multi-component dense reward function and an introduced requirement to lift the objects while satisfying the target grasp constraint. Thus, our methodology implicitly guides the

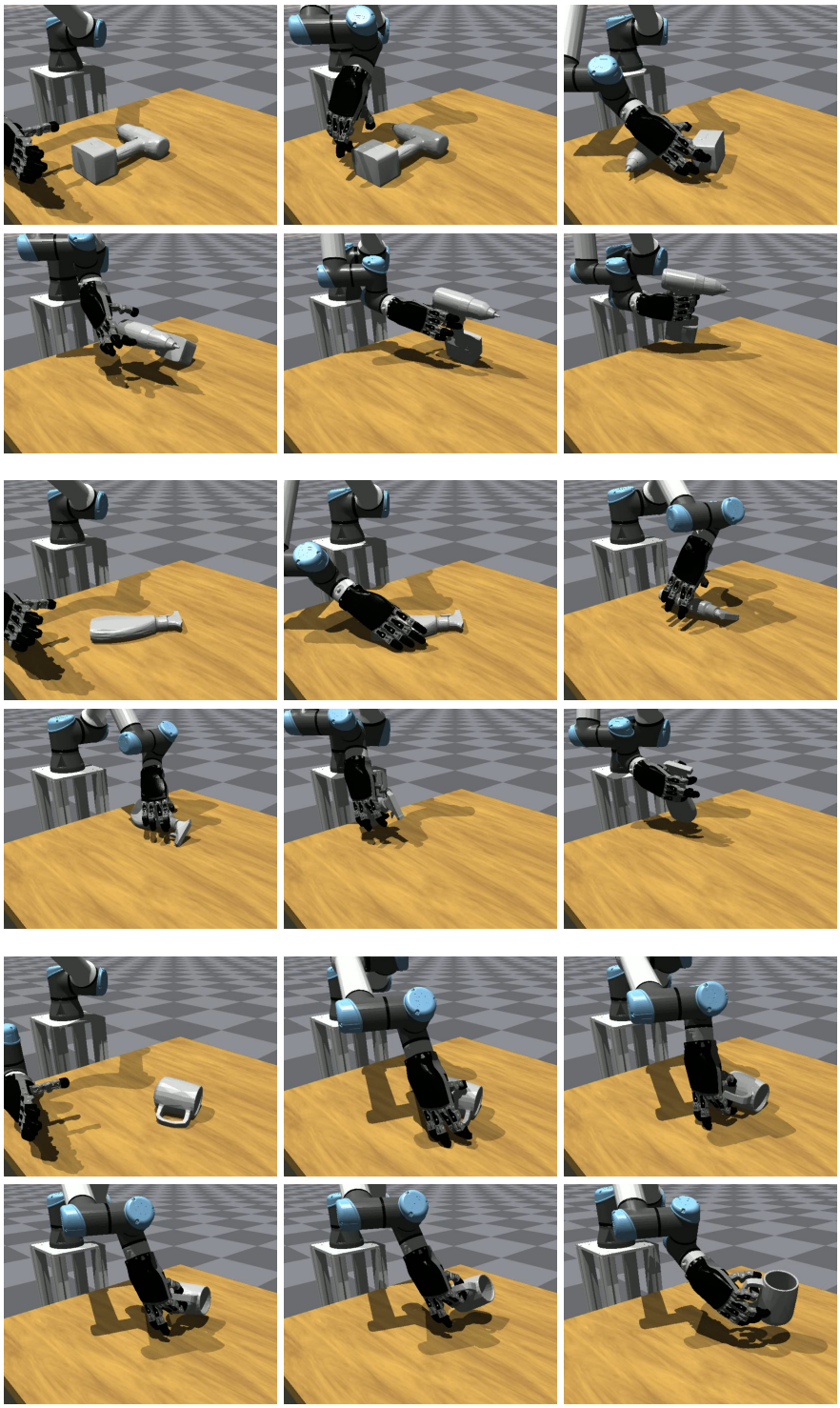


Figure 5.12: Rollouts of policy manipulating unseen objects of known categories. Top to bottom: drill, spray bottle, and mug. The constraint-based target grasp representation is used. Note complex repositioning and reorienting behaviors for the drill and spray bottle. The mug task usually can be solved with a straightforward manipulation strategy.

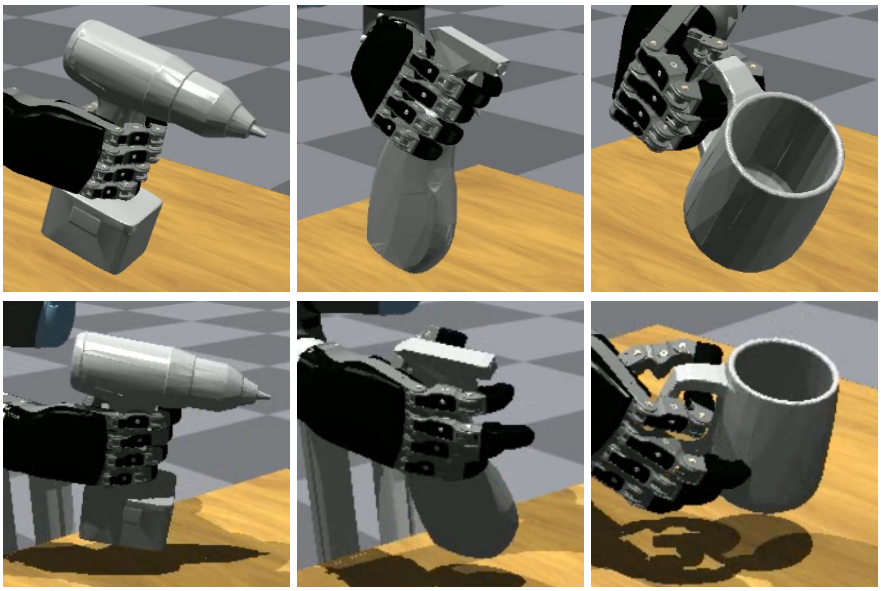


Figure 5.13: Close-up snapshots of functional grasps achieved by policies learned using two different grasp representations. *Top:* Using an explicit grasp representation. *Bottom:* Using a constraint-based grasp representation. In the second case, given only an abstract constraint, the policy learned to produce natural-looking grasps. Note that grasp targets for the explicit grasp representation were carefully designed by hand.

policy to discover effective object grasping strategies, often aligning with the ways they are naturally designed to be grasped by humans.

The policies achieve success rates of 94% for the explicit target grasp representation and 90% for the constraint-based target grasp representation. In both cases, failures occur when the object is repositioned and reoriented to a workspace location with low manipulability, leading the policy to become stuck in repetitive local minima behavior. To address this issue, an additional later stage in the training curriculum could be introduced, where most of the reward components have a negative range. Having all reward components with a negative range from the start results in very slow learning, with policies often being unable to discover complex manipulation behaviors.

This evaluation shows that the proposed approach consistently yields robust dexterous manipulation policies for functional grasping in simulation. The main strength of the approach is the generality of the proposed reward function and curriculum, which do not require any category- or instance-specific engineering. At the same time, the multi-component reward function provides dense, continuous rewards that quickly guide the policy towards general and robust behavior without a need for expert demonstrations. In combination with a high-performance GPU simulation, complex pre-grasping strategies are learned in under three hours. The state and the reward are formulated in a way that is agnostic of the robotic arm kinematics and can be easily

adapted to a hand with an arbitrary number of fingers and controlled DoF. Two possible target grasp representations are explored. In both cases, stable learning that yields policies that successfully manipulate previously unseen object instances is demonstrated.

The main limitation of the proposed approach is its reliance on frequent and accurate estimation of the target object pose. In the real world, in the presence of the interacting robotic hand, 6D object pose estimation is challenging (Amini, Periyasamy, and Behnke, 2022). Transferring our approach to a real robot is a prominent future work. Although the learned policies show robust behavior, additional real-world learning is likely necessary to close the sim-to-real gap. For this, we envision learning the weights of the last layer of our policy online on the real robot. A DRL approach presented in Chapter 3 can be utilized to achieve this goal.

5.7 DISCUSSION

In this chapter, we presented a DRL approach for dexterous categorical pre-grasp manipulation for functional grasping with an anthropomorphic hand. We proposed two target grasp representations: explicit and a more abstract, constraint-based one. We introduced a dense multi-component reward function and a curriculum for each grasp representation that share the same basic principles. They allowed to quickly learn policies for dexterous manipulation of complex objects of three categories with both target grasp representations.

Our experimental results demonstrated that the proposed approach reliably converged during training and generated policies yielding high success rates, even when applied to novel object instances of known categories. The method successfully learned complex pre-grasping strategies, including repositioning, reorienting, re-grasping, and up-righting the objects. A single policy successfully manipulated novel instances of three known object categories.

The performed ablation studies confirmed the importance of the proposed multi-component reward function and the curriculum. Our approach utilizes a high-performance GPU-based simulation, and the policies for both target grasp representations were learned on a single GPU in less than three hours. The policy using an explicit target grasp representation achieved a 94% success rate for functional grasping of novel object instances. The policy utilizing a constraint-based target grasp representation achieved a 90% success rate. It simultaneously learned human-like grasp configurations solely from the provided functional grasp constraints. The stable convergence of both policies, along with their consistently high success rates, underscores the robustness and generality of the proposed learning framework.

6 CONCLUSION

"The impediment to action advances action.
What stands in the way becomes the way."

— Marcus Aurelius

In this thesis, we presented several novel approaches that enable efficient robotic manipulator motion generation. We utilized both learning-based and optimization-based approaches. In particular, we addressed trajectory tracking control, dual-arm trajectory planning, and dexterous manipulation with a human-like hand.

Initially, a method for improving trajectory tracking accuracy with reference correction through a policy learned with supervised learning was proposed. The policy was learned offline from a small real-world dataset. The model represented an open-loop feed-forward outer-loop controller that ran on top of the underlying classical controller. We introduced a one-step future prediction module, hardwired within the model architecture to facilitate elements of planning-oriented behavior, as opposed to a purely reactive policy.

Such a module enabled reusing the same dataset to learn two different modalities, first providing a one-step future prediction, followed by the reference correction. In this way, we increased the data efficiency of the method. The approach was evaluated in the real world on the 7 DoF left arm of the Baxter robot. The arm features flexible joints that increase the safety of the system. However, combined with inexpensive hardware, they significantly increase the difficulty of control. We compared the proposed method with several alternative model architectures that were trained on the same dataset as well as with the vendor-provided classical controller.

The conducted experiments demonstrated that the proposed method increased trajectory tracking accuracy the most, compared to other approaches. Even in the presence of previously unseen payloads, the method demonstrated a persistent improvement of the trajectory tracking accuracy. The ablation study of the one-step future prediction indicated that incorporating such a module within the network architecture resulted in higher accuracy of trajectory tracking.

To further improve the trajectory tracking accuracy, we introduced a methodology to learn a reference correction policy online, directly on the real robot. In this way, our method does not require a separate data collection step and can be seamlessly applied to a robot during

its normal operation. The policy was learned with DRL using a data-efficient SAC approach.

The stochastic policy was represented with a beta distribution to ensure that the generated actions are within the provided safety margins. The compact state representation that included observations from the past, the present, and the future reference targets, together with the proposed dense reward function, enabled the method to achieve a stable convergence while learning on a single robot instance. The policy converged within two hours of training on a regular laptop, showcasing the efficiency of the proposed learning pipeline.

In addition, we explored a possibility of using a coarse dynamics approximation learned from a small real-world dataset as a simulation to pre-train the policy before the learning process on the real robot is started. Although our approach can be safely and efficiently applied from scratch directly on the real robot, it is always beneficial to reduce the effects of the initial exploration actions of a policy initialized with random weights. The experiments demonstrated that the policy, pre-trained in the learned simulation, had a substantial improvement in the trajectory tracking accuracy at the start of the learning.

It was shown that the presented approach significantly improved the trajectory tracking accuracy of a 7 DoF arm of the Baxter robot compared to the open-loop approach presented in Chapter 2 and to the vendor-provided classical controller.

After addressing the trajectory tracking, we proposed an optimization approach based on STOMP to achieve efficient dual-arm trajectory planning. A multi-component cost function was adapted for the dual-arm setup. It allowed obtaining trajectories that satisfy several costs and constraints while moving two arms simultaneously. To improve the speed of convergence towards obstacle-free regions, we introduced an obstacle cost term based on estimation of the worst-case overlap volume. Together with collision approximation defined as a set of spheres, such a cost function better reflects the severity of a collision in the presence of two independent controllable kinematic chains. This contrasts with traditional estimations based on the sum of penetration distances, which typically work well for a single arm.

To address the closed kinematic chain constraint, we split the chain into active and passive sub-chains. The trajectory of the active sub-chain is optimized directly. At the same time, the trajectory of the passive sub-chain is defined by projection given the trajectory of the active sub-chain. Although such an approach has been previously applied to different planners, we further enhance it by implicitly resolving the redundancy of the passive sub-chain by optimizing for the initial configurations of the IK solver. We applied our approach to a Centauro robot that has two 7 DoF arms, resulting in a total of 14 controllable DoF in the context of arm trajectory optimization. We evaluated the proposed method both in simulation and on the real

robot. It was shown that our approach achieved faster runtimes while maintaining high success rates and short trajectory lengths compared to several well-established planners.

Finally, we addressed dexterous object manipulation by means of a stochastic policy learned with DRL. In particular, we tackled a problem of a pre-grasp manipulation for functional grasping. In order to be functional, a grasp should satisfy strict criteria, such as the index finger on a trigger of a drill. Because of that, the direct functional grasp can not always be achieved, thus requiring the pre-grasp manipulation.

We studied two possible target grasp representations: an explicit one and a constraint-based one. While the first one requires the policy to exactly achieve a specific provided configuration, the latter one only specifies the constraint to be fulfilled, such as the index fingertip position. The advantage of the latter is that the policy is free to explore the ways to grasp objects while satisfying the provided constraint. This reduces the requirements on the external oracle that provides the grasp targets. Additionally, it creates an opportunity to learn the way to grasp the objects in conjunction with the respective manipulation strategies.

The proposed multi-component dense reward function, together with the learning curriculum, enabled fully omitting any expert demonstrations and other expensive data collection procedures. The policies were learned from scratch in a highly-parallelized GPU-based simulation. For both target grasp representations, intuitive dexterous behaviors were achieved, utilizing a multi-finger hand to reposition and reorient the objects before grasping. The policies achieved stable convergence and high success rates on a single computer in under three hours.

We evaluated the method in simulation on previously unseen instances of the three known object categories: drills, spray bottles, and mugs. Both grasp representations yielded robust policies that could recover from failed manipulation attempts while utilizing all fingers of the dexterous human-like hand. A single policy was shown to successfully achieve target functional grasps for objects of three different categories.

OUTLOOK AND FUTURE WORK

The methods introduced in this thesis open several directions for future research. The two-stage model for feed-forward open-loop trajectory tracking with reference correction can be extended to look several steps into the future. This would enhance the planning capabilities of the resulting policy and further improve trajectory tracking accuracy.

The closed-loop reference correction approach for trajectory tracking, learned directly on the real robot with DRL could benefit from

adaptive learning of maximum allowed action magnitude. In this work we utilize an analytical methodology to define this parameter, using joint velocity and acceleration limits. However, dynamically learned maximum action magnitude could make the method more flexible and automatically adjust to safety requirements for each specific robot and environment.

In the presented approach on dual-arm trajectory optimization, the trajectory duration is shared among both arms. This results in self-collision avoidance that is easier to achieve. However, in some cases an arm with a geometrically short trajectory moves unnecessarily slowly, sharing the duration with the other arm. The approach could be extended to support independent duration for each arm. In addition, consideration of dynamic obstacles is another interesting direction for research. These improvements would enable the method to effectively address a broader range of tasks.

Finally, the proposed dexterous manipulation policy relies on continuous and accurate information about the object shape and pose. In real-world scenarios, observations are often noisy, and object geometry estimations are incomplete. Producing a more robust policy that requires less accurate information about an object could be achieved with policy distillation techniques. An additional reward term, penalizing occluding the perception sensor while manipulating an object, would bring the approach even closer to the real-world application.

In this thesis we consider only visual perception in the context of object manipulation. However, humans heavily rely on tactile sensing when performing manipulation, especially when it comes to highly dynamic motions. Incorporating additional sensory modalities into the input of the policy, such as the measurements of the force-torque sensors, could significantly increase the range of tasks that can be tackled successfully.

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
6D	six-dimensional
A2C	advantage actor-critic
A3C	asynchronous advantage actor-critic
ANC	adaptive neural control
APF	artificial potential fields
APM	active-passive methodology
APMCC	APM with convenient configuration
APMCCR	APM with convenient configuration random
APMRR	APM with redundancy resolution
BIT*	batch informed trees
BRNN	bi-directional recurrent neural network
CHOMP	covariant Hamiltonian optimization for motion planning
CoM	center of mass
CPU	central processing unit
DDP	differential dynamic programming
DDPG	deep deterministic policy gradient
DNN	deep neural network
DoF	degrees of freedom
DRL	deep reinforcement learning
DWH	dynoNet Wiener-Hammerstein
FIN	forward-inference network
FK	forward kinematics
GP	Gaussian process
GPR	Gaussian process regression
GPU	graphics processing unit
HER	hindsight experience replay
IIR	infinite impulse response
IK	inverse kinematics
IL	imitation learning

ILC	iterative learning control
KPIECE	kinodynamic planning by interior-exterior cell exploration
LQR	linear quadratic regulator
LSTM	long short-term memory
LTI	linear time-invariant
MDP	Markov decision process
MIMO	multiple-input multiple-output
MLP	multi-layer perceptron
MPC	model predictive control
MPN	motion planning network
MSE	mean square error
NN	neural network
OMPL	open motion planning library
PD	proportional-derivative
PID	proportional-integral-derivative
PPO	proximal policy optimization
PRM	probabilistic roadmap
RAM	random-access memory
RBF	radial basis function
RC	repetitive control
ReLU	rectified linear unit
RL	reinforcement learning
RNN	recurrent neural network
ROS	robot operating system
RRT	rapidly-exploring random tree
SAC	soft actor-critic
SDF	signed distance field
SDLS	selectively damped least squares
SGD	stochastic gradient descent
SISO	single-input single-output
SRCP	stochastic reference correction policy
STOMP	stochastic trajectory optimization for motion planning
TRPO	trust region policy optimization

BIBLIOGRAPHY

Agarwal, A., S. Uppal, K. Shaw, and D. Pathak (2023). "Dexterous functional grasping." In: *Conference on Robot Learning (CoRL)*.

Alwala, K. and M. Mukadam (2021). "Joint sampling and trajectory optimization over graphs for online motion planning." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4700–4707.

Amini, A., A. S. Periyasamy, and S. Behnke (2022). "YOLOPose: Transformer-based multi-object 6D pose estimation using keypoint regression." In: *International Conference on Intelligent Autonomous Systems (IAS)*. Vol. 577, pp. 392–406.

An, C. H., C. G. Atkeson, and J. M. Hollerbach (1988). *Model-based control of a robot manipulator*. MIT Press, p. 254.

Andrychowicz, M., F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba (2017). "Hindsight experience replay." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5048–5058.

Andrychowicz, O. M., B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba (2020). "Learning dexterous in-hand manipulation." In: *The International Journal of Robotics Research (IJRR)* 39.1, pp. 3–20.

Arimoto, S. (1990). "Learning control theory for robotic motion." In: *International Journal of Adaptive Control and Signal Processing* 4.6, pp. 543–564.

Baek, I., K. Shin, H. Kim, S. Hwang, E. Demeester, and M.-S. Kang (2021). "Pre-grasp manipulation planning to secure space for power grasping." In: *IEEE Access* 9, pp. 157715–157726.

Bayiz, Y. E. and R. Babuska (2014). "Nonlinear disturbance compensation and reference tracking via reinforcement learning with fuzzy approximators." In: *IFAC Proceedings Volumes* 47.3, pp. 5393–5398.

Bristow, D., M. Tharayil, and A. Alleyne (2006). "A survey of iterative learning control." In: *IEEE Control Systems Magazine* 26.3, pp. 96–114.

Brock, O. and O. Khatib (2002). "Elastic strips: A framework for motion generation in human environments." In: *The International Journal of Robotics Research (IJRR)* 21.12, pp. 1031–1052.

Burget, F., M. Bennewitz, and W. Burgard (2016). "BI2RRT*: An efficient sampling-based path planning framework for task-constrained mobile manipulation." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3714–3721.

Buss, S. R. and J.-S. Kim (2005). "Selectively damped least squares for inverse kinematics." In: *Journal of Graphics Tools* 10, pp. 37–49.

Byravan, A., B. Boots, S. Srinivasa, and D. Fox (2014). "Space-time functional gradient optimization for motion planning." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6499–6506.

Byrne, S., W. Naeem, and S. Ferguson (2015). "Improved APF strategies for dual-arm local motion planning." In: *Transactions of the Institute of Measurement and Control* 37.1, pp. 73–90.

Calderon-Cordova, C. and R. Sarango (2023). "A deep reinforcement learning algorithm for robotic manipulation tasks in simulated environments." In: *Engineering Proceedings* 47.1, p. 12.

Callar, T.-C. and S. Böttger (2022). "Hybrid learning of time-series inverse dynamics models for locally isotropic robot motion." In: *IEEE Robotics and Automation Letters (RA-L)* 8, pp. 1061–1068.

Cao, S., L. Sun, J. Jiang, and Z. Zuo (2023). "Reinforcement learning-based fixed-time trajectory tracking control for uncertain robotic manipulators with input saturation." In: *IEEE Transactions on Neural Networks and Learning Systems* 34.8, pp. 4584–4595.

Chang, L. Y., S. S. Srinivasa, and N. S. Pollard (2010). "Planning pre-grasp manipulation for transport tasks." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2697–2704.

Chen, D., Q. Qi, Z. Zhuang, J. Wang, J. Liao, and Z. Han (2021). "Mean field deep reinforcement learning for fair and efficient UAV control." In: *IEEE Internet of Things Journal* 8.2, pp. 813–828.

Chen, Q., L. Shi, Y. Nan, and X. Ren (2016). "Adaptive neural dynamic surface sliding mode control for uncertain nonlinear systems with unknown input saturation." In: *International Journal of Advanced Robotic Systems (IJARS)* 13.5, pp. 1–14.

Chen, S. and J. T. Wen (2019). "Neural-learning trajectory tracking control of flexible-joint robot manipulators with unknown dynamics." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 128–135.

– (2021). "Industrial robot trajectory tracking control using multi-layer neural networks trained by iterative learning control." In: *Robotics* 10.50, pp. 1–20.

Chen, Z., K. V. Wyk, Y.-W. Chao, W. Yang, A. Mousavian, A. Gupta, and D. Fox (2022). "Learning robust real-world dexterous grasping policies via implicit shape augmentation." In: *Conference on Robot Learning (CoRL)*.

Chou, P.-W., D. Maturana, and S. Scherer (2017). "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution." In: *International Conference on Machine Learning (ICML)*, pp. 834–843.

Choudhury, S., J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. A. Scherer (2016). "Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal

path planning." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4207–4214.

Cohen, B., S. Chitta, and M. Likhachev (2014). "Single- and dual-arm motion planning with heuristic search." In: *International Journal of Robotics Research* 33.2, pp. 305–320.

Cohn, T., S. Shaw, M. Simchowitz, and R. Tedrake (2024). "Constrained bimanual planning with analytic inverse kinematics." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6935–6942.

Cuiyan, L., Z. Dongchun, and Z. Xianyi (2004). "A survey of repetitive control." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 2, pp. 1160–1166.

Dasari, S., A. Gupta, and V. Kumar (2023). "Learning dexterous manipulation from exemplar object trajectories and pre-grasps." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3889–3896.

Deng, X., Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox (2020). "Self-supervised 6D object pose estimation for robot manipulation." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3665–3671.

Dogar, M. R. and S. S. Srinivasa (2010). "Push-grasping with dexterous hands: Mechanics and a method." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2123–2130.

Fedus, W., P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney (2020). "Revisiting fundamentals of experience replay." In: *International Conference on Machine Learning (ICML)*.

Forgione, M. and D. Piga (2021). "dynoNet: A neural network architecture for learning dynamical systems." In: *International Journal of Adaptive Control and Signal Processing* 35.4, pp. 612–626.

Franceschetti, A., E. Tosello, N. Castaman, and S. Ghidoni (2022). "Robotic arm control and task training through deep reinforcement learning." In: *Intelligent Autonomous Systems 16*, pp. 532–550.

Gammell, J. D., S. S. Srinivasa, and T. D. Barfoot (2014). "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2997–3004.

Gammell, J. D., S. S. Srinivasa, and T. D. Barfoot (2015). "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3067–3074.

Geraerts, R. and M. H. Overmars (2007). "Creating high-quality paths for motion planning." In: *The International Journal of Robotics Research (IJRR)* 26.8, pp. 845–863.

Ghafarian Tamizi, M., M. Yaghoubi, and H. Najjaran (2023). "A review of recent trend in motion planning of industrial robots." In: *International Journal of Intelligent Robotics and Applications* 7, pp. 1–22.

Giri, F. and E.-W. Bai (2010). "Block oriented nonlinear system identification." In: *Lecture Notes in Control and Information Sciences*, pp. 746–758.

Greff, K., R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber (2017). "LSTM: A search space odyssey." In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10, pp. 2222–2232.

Guo, H., F. Wu, Y. Qin, R. Li, K. Li, and K. Li (2023). "Recent trends in task and motion planning for robotics: A survey." In: *ACM Computing Surveys* 55, pp. 1–36.

Guo, Q., Y. Zhang, B. G. Celler, and S. W. Su (2019). "Neural adaptive backstepping control of a robotic manipulator with prescribed performance constraint." In: *IEEE Transactions on Neural Networks and Learning Systems* 30, pp. 3572–3583.

Ha, H., J. Xu, and S. Song (2020). "Learning a decentralized multi-arm motion planner." In: *Conference on Robot Learning (CoRL)*, pp. 21–36.

Haarnoja, T., H. Tang, P. Abbeel, and S. Levine (2017). "Reinforcement learning with deep energy-based policies." In: *International Conference on Machine Learning (ICML)*, pp. 1352–1361.

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." In: *International Conference on Machine Learning (ICML)*. Vol. 80, pp. 1861–1870.

Haarnoja, T., A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine (2018). "Soft actor-critic algorithms and applications." In: *preprint arXiv:1812.05905*.

Hafner, D., T. P. Lillicrap, M. Norouzi, and J. Ba (2021). "Mastering atari with discrete world models." In: *International Conference on Learning Representations*.

Han, H., X. Wu, L. Zhang, Y. Tian, and J. Qiao (2019). "Self-organizing RBF neural network using an adaptive gradient multiobjective particle swarm optimization." In: *IEEE Transactions on Cybernetics* 49, pp. 69–82.

Han, H., L. Zhang, Y. Hou, and J. Qiao (2016). "Nonlinear model predictive control based on a self-organizing recurrent neural network." In: *IEEE Transactions on Neural Networks and Learning Systems* 27, pp. 402–415.

Han, L. and J. Trinkle (1998). "Dextrous manipulation by rolling and finger gaiting." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 730–735.

Han, X., H. Laga, and M. Bennamoun (2021). "Image-based 3D object reconstruction: State-of-the-art and trends in the deep learning era." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.5, pp. 1578–1604.

Hang, K., A. S. Morgan, and A. M. Dollar (2019). "Pre-grasp sliding manipulation of thin objects using soft, compliant, or underactuated hands." In: *IEEE Robotics and Automation Letters (RA-L)* 4.2, pp. 662–669.

He, W., Z. Yan, Y. Sun, Y. Ou, and C. Sun (2018). "Neural-learning-based control for a constrained robotic manipulator with flexible joints." In: *IEEE Transactions on Neural Networks and Learning Systems* 29, pp. 5993–6003.

Hofer, T., F. Shamsafar, N. Benbarka, and A. Zell (2021). "Object detection and autoencoder-based 6D pose estimation for highly cluttered bin picking." In: *IEEE International Conference on Image Processing (ICIP)*, pp. 704–708.

Hu, Y. and B. Si (2018). "A reinforcement learning neural network for robotic manipulator control." In: *Neural Computation* 30.7, pp. 1983–2004.

Huang, W., Y. Lin, M. Liu, and H. Min (2024). "Velocity-aware spatial-temporal attention LSTM model for inverse dynamic model learning of manipulators." In: *Frontiers in Neurorobotics* 18, p. 1353879.

Hwangbo, J., J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter (2019). "Learning agile and dynamic motor skills for legged robots." In: *Science Robotics* 4.26, pp. 58–72.

Ibarz, J., J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine (2021). "How to train your robot with deep reinforcement learning: Lessons we have learned." In: *The International Journal of Robotics Research (IJRR)* 40.4-5, pp. 698–721.

Iriondo, A., E. Lazkano, L. Susperregi, J. Urain, A. Fernandez, and J. Molina (2019). "Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning." In: *Applied Sciences* 9.2, p. 348.

James J. Kuffner, J. and M. Steven (2000). "RRT-Connect: An efficient approach to single-query path planning." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 995–1001.

Jang, K., J. Baek, S. Park, and J. Park (2022). "Motion planning for closed-chain constraints based on probabilistic roadmap with improved connectivity." In: *IEEE/ASME Transactions on Mechatronics* 27.4, pp. 2035–2043.

Janson, L., E. Schmerling, A. Clark, and M. Pavone (2015). "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions." In: *The International Journal of Robotics Research (IJRR)* 34.7, pp. 883–921.

Jin, L., S. Li, J. Yu, and J. He (2018). "Robot manipulator control using neural networks: A survey." In: *Neurocomputing* 285, pp. 23–34.

Johnson, J. J., L. Li, F. Liu, A. H. Qureshi, and M. C. Yip (2020). "Dynamically constrained motion planning networks for non-holonomic robots." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6937–6943.

Kalakrishnan, M., S. Chitta, E. Theodorou, P. Pastor, and S. Schaal (2011). "STOMP: Stochastic trajectory optimization for motion planning." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4569–4574.

Kappler, D., L. Chang, M. Przybylski, N. Pollard, T. Asfour, and R. Dillmann (2010). "Representation of pre-grasp strategies for object manipulation." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 617–624.

Kappler, D., L. Y. Chang, N. S. Pollard, T. Asfour, and R. Dillmann (2012). "Templates for pre-grasp sliding interactions." In: *Robotics and Autonomous Systems* 60.3, pp. 411–423.

Kavraki, L., P. Svestka, J.-C. Latombe, and M. Overmars (1996). "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." In: *IEEE Transactions on Robotics and Automation* 12.4, pp. 566–580.

Kingston, Z., M. Moll, and L. E. Kavraki (2018). "Sampling-based methods for motion planning with constraints." In: *Annual Review of Control, Robotics, and Autonomous Systems* 1, pp. 159–185.

Klamt, T., M. Schwarz, C. Lenz, L. Baccelliere, D. Buongiorno, T. Cichon, A. DiGuardo, D. Droeschel, M. Gabardi, M. Kamedula, N. Kashiri, A. Laurenzi, D. Leonardis, L. Muratore, D. Pavlichenko, A. S. Periyasamy, D. Rodriguez, M. Solazzi, A. Frisoli, M. Gustmann, J. Roßmann, U. Süss, N. G. Tsagarakis, and S. Behnke (2020). "Remote mobile manipulation with the centauro robot: Full-body telepresence and autonomous operator assistance." In: *Journal of Field Robotics (JFR)* 37.5, pp. 889–919.

Kumar, V., D. Hoeller, B. Sundaralingam, J. Tremblay, and S. Birchfield (2021). "Joint space control via deep reinforcement learning." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3619–3626.

Kuntz Alan adn Bowen, C. and R. Alterovitz (2020). "Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization." In: *Robotics Research*, pp. 929–945.

Kunz, T. and M. Stilman (2014). "Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3713–3719.

Lavalle, S. M. (1998). "Rapidly-exploring random trees: A new tool for path planning." In: *The annual research report* 98.11.

Levine, S., C. Finn, T. Darrell, and P. Abbeel (2016). "End-to-end training of deep visuomotor policies." In: *Journal of Machine Learning Research (JMLR)* 17.1, pp. 1334–1373.

Li, L., Y. Miao, A. H. Qureshi, and M. C. Yip (2021). "MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints." In: *IEEE Robotics and Automation Letters (RA-L)* 6.3, pp. 4496–4503.

Li, Q., J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig (2017). "Deep neural networks for improved, impromptu trajectory tracking of quadrotors." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5183–5189.

Li, X., W. Shang, and S. Cong (2024). "Offline reinforcement learning of robotic control using deep kinematics and dynamics." In: *IEEE/ASME Transactions on Mechatronics* 29.4, pp. 2428–2439.

Li, Y., Y. Wang, G. Wang, and T. Shi (2024). "A multi-policy framework for manipulator trajectory tracking based on feature extraction and rl compensation." In: *International Conference on Automation, Robotics and Applications (ICARA)*, pp. 191–195.

Li, Z., X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath (2021). "Reinforcement learning for robust parameterized locomotion control of bipedal robots." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2811–2817.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2015). "Continuous control with deep reinforcement learning." In: *CoRR* abs/1509.02971.

Lin, F.-J., S.-G. Chen, and I.-F. Sun (2017). "Intelligent sliding-mode position control using recurrent wavelet fuzzy neural network for electrical power steering system." In: *International Journal of Fuzzy Systems* 19, pp. 1344–1361.

Longman, R. (2000). "Iterative learning control and repetitive control for engineering practice." In: *International Journal of Control* 73, pp. 930–954.

Lu, C.-H. (2011). "Wavelet fuzzy neural networks for identification and predictive control of dynamic systems." In: *IEEE Transactions on Industrial Electronics* 58, pp. 3046–3058.

Mahler, J., S. Krishnan, M. Laskey, S. Sen, A. Murali, B. Kehoe, S. Patil, J. Wang, M. Franklin, P. Abbeel, and K. Goldberg (2014). "Learning accurate kinematic control of cable-driven surgical robots using data cleaning and gaussian process regression." In: *IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 532–539.

Makoviichuk, D. and V. Makoviychuk (2021). *rl-games: A high-performance framework for reinforcement learning*. https://github.com/Denys88/rl_games.

Makoviychuk, V., L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State (2021). "Isaac Gym: High performance GPU-based physics simulation for robot learning." In: *preprint arXiv:2108.10470*.

Mandikal, P. and K. Grauman (2020). "Learning dexterous grasping with object-centric visual affordances." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6169–6176.

– (2022). "DexVIP: Learning dexterous grasping with human hand pose priors from video." In: *Conference on Robot Learning (CoRL)*.

Mayne, D. (1966). "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems." In: *International Journal of Control (IJC)* 3, pp. 85–95.

Meijer, J., Q. Lei, and M. Wisse (2017). "An empirical study of single-query motion planning for grasp execution." In: *IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1234–1241.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning." In: *International Conference on Machine Learning (ICML)*. Vol. 48, pp. 1928–1937.

Morse, K., N. Das, Y. Lin, A. Wang, A. Rai, and F. Meier (2020). "Learning state-dependent losses for inverse dynamics learning." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5261–5268.

Mosbach, M. and S. Behnke (2022). "Efficient representations of object geometry for reinforcement learning of interactive grasping policies." In: *IEEE International Conference on Robotic Computing (IRC)*, pp. 156–163.

Muhayyuddin, M. Moll, L. Kavraki, and J. Rosell (2018). "Randomized physics-based motion planning for grasping in cluttered and uncertain environments." In: *IEEE Robotics and Automation Letters (RA-L)* 3.2, pp. 712–719.

Orthey, A., B. Frész, and M. Toussaint (2020). "Motion planning explorer: visualizing local minima using a local-minima tree." In: *IEEE Robotics and Automation Letters (RA-L)* 5.2, pp. 346–353.

Palleschi, A., F. Angelini, C. Gabellieri, D. W. Park, L. Pallottino, A. Bicchi, and M. Garabini (2023). "Grasp It Like a Pro 2.0: A data-driven approach exploiting basic shape decomposition and human data for grasping unknown objects." In: *IEEE Transactions on Robotics* 39.5, pp. 4016–4036.

Pane, Y., S. Nageshrao, J. Kober, and R. Babuska (2019). "Reinforcement learning based compensation methods for robot manipulators." In: *Engineering Applications of Artificial Intelligence* 78, pp. 236–247.

Patan, K., M. Patan, and D. Kowalów (2017). "Neural networks in design of iterative learning control for nonlinear systems." In: *IFAC-PapersOnLine* 50, pp. 13402–13407.

Pavlichenko, D. (2016). "Efficient stochastic multicriteria arm trajectory optimization." MA thesis. Bonn, Germany: University of Bonn. URL: https://www.ais.uni-bonn.de/theses/Dmytro_Pavlichenko_Master_Thesis_12_2016.pdf.

Pavlichenko, D. and S. Behnke (2017). "Efficient stochastic multicriteria arm trajectory optimization." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4018–4025.

– (2021). "Flexible-joint manipulator trajectory tracking with learned two-stage model employing one-step future prediction." In: *IEEE International Conference on Robotic Computing (IRC)*.

– (2022a). "Flexible-joint manipulator trajectory tracking with two-stage learned model utilizing a hardwired forward dynamics prediction." In: *International Journal of Semantic Computing (IJSC)* 16.03, pp. 403–423.

– (2022b). "Real-robot deep reinforcement learning: improving trajectory tracking of flexible-joint manipulator with reference correction." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2671–2677.

– (2023). "Deep reinforcement learning of dexterous pre-grasp manipulation for human-like functional categorical grasping." In: *IEEE International Conference on Automation Science and Engineering (CASE)*.

– (2025). "Dexterous pre-grasp manipulation for human-like functional categorical grasping: Deep reinforcement learning and grasp representations." In: *IEEE Transactions on Automation Science and Engineering (T-ASE)*.

Pavlichenko, D., D. Rodriguez, C. Lenz, M. Schwarz, and S. Behnke (2019). "Autonomous bimanual functional regrasping of novel object class instances." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 351–358.

Pavlichenko, D., D. Rodriguez, M. Schwarz, C. Lenz, A. S. Periyasamy, and S. Behnke (2018). "Autonomous dual-arm manipulation of familiar objects." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*.

Pereida, K., D. Kooijman, R. Duivenvoorden, and A. P. Schoellig (2018). "Transfer learning for high-precision trajectory tracking through L1 adaptive feedback and iterative learning." In: *International Journal of Adaptive Control and Signal Processing* 33, pp. 388–409.

Perez, A., S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter (2011). "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4307–4313.

Pradhan, S. K. and B. Subudhi (2012). "Real-time adaptive control of a flexible manipulator using reinforcement learning." In: *IEEE Transactions on Automation Science and Engineering* 9.2, pp. 237–249.

Qiao, J., X. Meng, and W. Li (2018). "An incremental neuronal-activity-based RBF neural network for nonlinear system modeling." In: *Neurocomputing* 302, pp. 1–11.

Qin, Y., B. Huang, Z.-H. Yin, H. Su, and X. Wang (2022). "DexPoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation." In: *Conference on Robot Learning (CoRL)*.

Qureshi, A. H., M. J. Bency, and M. C. Yip (2019). "Motion planning networks." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2118–2124.

Qureshi, A. H., J. Dong, A. Choe, and M. C. Yip (2020). "Neural manipulation planning on constraint manifolds." In: *IEEE Robotics and Automation Letters (RA-L)* 5, pp. 6089–6096.

Qureshi, A. H., Y. Miao, A. Simeonov, and M. C. Yip (2021). "Motion planning networks: Bridging the gap between learning-based and classical motion planners." In: *IEEE Transactions on Robotics* 37, pp. 48–66.

Qureshi, A. H. and M. C. Yip (2018). "Deeply informed neural sampling for robot motion planning." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6582–6588.

Radosavovic, I., X. Wang, L. Pinto, and J. Malik (2021). "State-only imitation learning for dexterous manipulation." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7865–7871.

Rajeswaran, A., V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine (2018). "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations." In: *Robotics: Science and Systems (RSS)*.

Ratliff, N., M. Zucker, J. A. Bagnell, and S. Srinivasa (2009). "CHOMP: Gradient optimization techniques for efficient motion planning." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 489–494.

Ren, Z., C. Zhou, S. Xin, and N. Tsagarakis (2017). "HERI hand: A quasi dexterous and powerful hand with asymmetrical finger dimensions and under actuation." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 322–328.

Riedmiller, M. A. (2012). "10 steps and some tricks to set up neural reinforcement controllers." In: *Neural Networks: Tricks of the Trade (2nd ed.)* Springer, pp. 735–757.

Rodriguez, D. and S. Behnke (2018). "Transferring category-based functional grasping skills by latent space non-rigid registration." In: *IEEE Robotics and Automation Letters (RA-L)* 3, pp. 2662–2669.

– (2021). "DeepWalk: Omnidirectional bipedal gait by deep reinforcement learning." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3033–3039.

Rodriguez, D., A. Di Guardo, A. Frisoli, and S. Behnke (2018). "Learning postural synergies for categorical grasping through shape space registration." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 270–276.

Rodriguez, D., F. Huber, and S. Behnke (2020). "Category-level 3D non-rigid registration from single-view RGB images." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10617–10624.

Rueckert, E., M. Nakatenus, S. Tosatto, and J. Peters (2017). "Learning inverse dynamics models in O(n) time with LSTM networks." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 811–816.

Rupert, L., P. Hyatt, and M. D. Killpack (2015). "Comparing model predictive control and input shaping for improved response of low-impedance robots." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 256–263.

Saveriano, M., Y. Yin, P. Falco, and D. Lee (2017). "Data-efficient control policy search using residual dynamics learning." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4709–4715.

Schulman, J., J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel (2013). "Finding locally optimal, collision-free trajectories with sequential convex optimization." In: *Robotics: Science and Systems IX*.

Schulman, J., S. Levine, P. Abbeel, M. Jordan, and P. Moritz (2015). "Trust region policy optimization." In: *International Conference on Machine Learning (ICML)*. Vol. 37, pp. 1889–1897.

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

Schuster, M. and K. K. Paliwal (1997). "Bidirectional recurrent neural networks." In: *IEEE Transactions on Signal Processing* 45, pp. 2673–2681.

Schwarz, M. and S. Behnke (2014). "Compliant robot behavior using servo actuator models identified by iterative learning control." In: *RoboCup 2013: Robot World Cup XVII*. Springer, pp. 207–218.

Shao, L., T. Migimatsu, and J. Bohg (2020). "Learning to scaffold the development of robotic manipulation skills." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5671–5677.

Sheng, L., G. Xiaojie, and Z. Lanyong (2017). "Robust adaptive backstepping sliding mode control for six-phase permanent magnet synchronous motor using recurrent wavelet fuzzy neural network." In: *IEEE Access* 5, pp. 14502–14515.

Shi, W., K. Wang, C. Zhao, and M. Tian (2022). "Obstacle avoidance path planning for the dual-arm robot based on an improved RRT algorithm." In: *Applied Sciences* 12.8, pp. 87–97.

Steffens, R., M. Nieuwenhuisen, and S. Behnke (2016). "Continuous motion planning for service robots with multiresolution in time." In: *International Conference on Intelligent Autonomous Systems (IAS)*, pp. 203–215.

Şucan, I. A. and L. E. Kavraki (2010). "Kinodynamic motion planning by interior-exterior cell exploration." In: *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*. Springer, pp. 449–464.

Şucan, I. A., M. Moll, and L. E. Kavraki (2012). "The Open Motion Planning Library." In: *IEEE Robotics & Automation Magazine* 19.4, pp. 72–82.

Sun, Z., K. Yuan, W. Hu, C. Yang, and Z. Li (2020). "Learning pregrasp manipulation of objects from ungraspable poses." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9917–9923.

Szynkiewicz, W. and J. Błaszczyk (2011). "Optimization-based approach to path planning for closed chain robot systems." In: *International Journal of Applied Mathematics and Computer Science* 21.4, pp. 659–670.

Talebi, H. A., R. V. Patel, and K. Khorasani (1998). "Inverse dynamics control of flexible-link manipulators using neural networks." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 806–811.

Terry, J. S., L. Rupert, and M. D. Killpack (2017). "Comparison of linearized dynamic robot manipulator models for model predictive control." In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 205–212.

Vahrenkamp, N., D. Berenson, T. Asfour, J. J. Kuffner, and R. Dillmann (2009). "Humanoid motion planning for dual-arm manipulation and re-grasping tasks." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2464–2470.

Wang, F., Z. Chao, L. Huang, H. Li, and C. Zhang (2017). "Trajectory tracking control of robot manipulator based on RBF neural network and fuzzy sliding mode." In: *Cluster Computing*, pp. 5799–5809.

Wang, G., F. Manhardt, X. Liu, X. Ji, and F. Tombari (2024). "Occlusion-aware self-supervised monocular 6D object pose estimation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.3, pp. 1788–1803.

Wang, M., H. Ye, and Z. Chen (2017). "Neural learning control of flexible joint manipulator with predefined tracking performance and application to baxter robot." In: *Complexity* 2017, pp. 1–14.

Wang, Z., W. Yan, and T. Oates (2017). "Time series classification from scratch with deep neural networks: A strong baseline." In: *IEEE International Joint Conference on Neural Networks (IJCNN)*, pp. 1578–1585.

Wei, W., P. Wang, S. Wang, Y. Luo, W. Li, D. Li, Y. Huang, and H. Duan (2024). "Learning human-like functional grasping for multifinger hands from few demonstrations." In: *IEEE Transactions on Robotics* 40, pp. 3897–3916.

Wu, R., T. Zhu, W. Peng, J. Hang, and Y. Sun (2023). "Functional grasp transfer across a category of objects from only one labeled instance." In: *IEEE Robotics and Automation Letters (RA-L)* 8.5, pp. 2748–2755.

Wu, T., Y. Gan, M. Wu, J. Cheng, Y. Yang, Y. Zhu, and H. Dong (2024). "Unidexfpm: Universal dexterous functional pre-grasp manipulation via diffusion policy." In: *preprint arXiv:2403.12421*.

Xanthidis, M., N. Karapetyan, H. Damron, S. Rahman, J. Johnson, A. O'Connell, J. M. O'Kane, and I. Rekleitis (2020). "Navigation in the presence of obstacles for an agile autonomous underwater vehicle." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 892–899.

Xia, D., L. Wang, and T. Chai (2014). "Neural-network-friction compensation based energy swing-up control of pendubot." In: *IEEE Transactions on Industrial Electronics* 61, pp. 1411–1423.

Xie, D. and N. Amato (2004). "A kinematics-based probabilistic roadmap method for high DOF closed chain systems." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 473–478.

Xu, Z., W. Huang, Z. Li, L. Hu, and P. Lu (2021). "Nonlinear nonsingular fast terminal sliding mode control using deep deterministic policy gradient." In: *Applied Sciences* 11.10, p. 4685.

Yang, C., X. Wang, L. Cheng, and H. Ma (2016). "Neural-learning-based telerobot control with guaranteed performance." In: *IEEE Transactions on Cybernetics* 47.10, pp. 3148–3159.

Yang, D., T. Tosun, B. Eisner, V. Isler, and D. Lee (2021). "Robotic grasping through combined image-based grasp proposal and 3D reconstruction." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6350–6356.

Yang, Y., D. Huang, and X. Dong (2019). "Enhanced neural network control of lower limb rehabilitation exoskeleton by add-on repetitive learning." In: *Neurocomputing* 323, pp. 256–264.

Yiming, J., Y. Chenguang, N. Jing, L. Guang, L. Yanan, and Z. Junpei (2017). "A brief review of neural networks based learning and control and their applications for robots." In: *Complexity*, pp. 1–14.

Zeng, W., S. Wang, R. Liao, Y. Chen, B. Yang, and R. Urtasun (2020). "DSDNet: Deep structured self-driving network." In: *European Conference on Computer Vision (ECCV)*, pp. 156–172.

Zhang, G., J. Wang, P. Yang, and S. Guo (2021). "Iterative learning sliding mode control for output-constrained upper-limb exoskeleton with non-repetitive tasks." In: *Applied Mathematical Modelling* 97, pp. 366–380.

Zhang, Y., J. Hang, T. Zhu, X. Lin, R. Wu, W. Peng, D. Tian, and Y. Sun (2023). "FunctionalGrasp: Learning functional grasp for robots via semantic hand-object representation." In: *IEEE Robotics and Automation Letters (RA-L)* 8.5, pp. 3094–3101.

Zhao, J., C.-M. Lin, and F. Chao (2019). "Wavelet fuzzy brain emotional learning control system design for MIMO uncertain nonlinear systems." In: *Frontiers in Neuroscience* 12, p. 918.

Zhou, L., H. Wang, Z. Zhang, Z. Liu, F. E. Tay, and M. H. Ang (2024). "You only scan once: A dynamic scene reconstruction pipeline for 6-DoF robotic grasping of novel objects." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13891–13897.

Zhou, W.-M. and D. Held (2022). "Learning to grasp the ungraspable with emergent extrinsic dexterity." In: *Conference on Robot Learning (CoRL)*.

Zhu, H., A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar (2019). "Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost." In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3651–3657.

Zhu, T., R. Wu, J. Hang, X. Lin, and Y. Sun (2023). "Toward human-like grasp: Functional grasp by dexterous robotic hand via object-hand semantic representation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.10, pp. 12521–12534.

Zhu, T., R. Wu, X. Lin, and Y. Sun (2021). "Toward human-like grasp: Dexterous grasping via semantic representation of object-hand." In: *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15721–15731.

Zhu, Y., Z. Wang, J. Merel, A. A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, and N. Heess (2018). "Reinforcement and imitation learning for diverse visuomotor skills." In: *Robotics: Science and Systems (RSS)*.

Zuo, W. and L. Cai (2010). "A new iterative learning controller using variable structure Fourier neural network." In: *IEEE Transactions on Systems, Man, and Cybernetics* 40, pp. 458–468.