



Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik

Towards Assistive Teleoperation and Its Application to Pouring Liquids

Edwin Babaians

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Bernhard U. Seeber

Prüfer*innen der Dissertation: 1. Prof. Dr.-Ing. Eckehard Steinbach
2. Prof. Dr. Cristina Piazza

Die Dissertation wurde am 07.06.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 05.09.2022 angenommen.

Abstract

In this dissertation, we investigate assistive teleoperation and apply it to liquid handling and pouring, demonstrating reliable and safe teleoperation. Assistive teleoperation heavily depends on autonomy sharing, which we cover in four major topics.

The first topic emphasizes manipulator motion planning. Teleoperation of robotic manipulators necessitates proper motion control and planning. Online motion control, while active in obstacle avoidance, often leads to unsafe motion. In contrast, offline motion planning produces precise and secure trajectories for complex manipulation tasks. We introduce a real-time nonlinear model predictive control-based motion planner (NMPC-MP) for teleoperated manipulation. Our model accommodates a complicated environment with dynamic obstacles, differing from traditional NMPC-based models. Real-time planning becomes feasible through our multi-threaded NMPC-MP. Utilizing the Kinova[®] Movo platform, we assess our methodology in both simulated and real-world environments. When juxtaposed with state-of-the-art approaches (e.g., RRT-Connect, CHOMP, and STOMP), our NMPC-MP reveals notable enhancements in real-time motion planning. This planner was employed in a dual manipulator setup during real-world tests. Experimental results illustrate that planners can accurately track the teleoperator's active goals while evading self-collisions and obstructions.

The second topic addresses shared autonomy amidst network discontinuity. Although wireless communication networks offer reduced latency and augmented transmission rates, the potential for network instability or packet drop persists. Real-time teleoperation necessitates seamless data transmission. Shared autonomy (SA) provides a solution. If remote data becomes inaccessible, the controller can persevere, drawing from previously observed models. Nonetheless, the spatial divergence between human and robot trajectories induces evident fluctuations, which pose challenges for teleoperation applications. We, therefore, suggest an innovative skill refinement technique to modify earlier trained skills and counteract unexpected, undesired motions during the control takeover phase. In our approach, we integrate the Hidden Semi-Markov Model (HSMM) with the Linear Quadratic Tracker (LQT) to comprehend and forecast user intentions. Subsequently, we use the Coherent Point Drift (CPD) to enhance the executable trajectory. We evaluate our strategy in simulations and real-world settings, specifically for 2D English letter drawing and 3D robot-assisted feeding scenarios. Results from the Kinova[®] Movo platform demonstrate that our refinement technique formulates stable trajectories and curtails inconsistencies during control switches.

Our third topic zeroes in on robot-liquid interaction. By harnessing a digital twin of the

manipulator and an avant-garde real-time liquid simulator, we explore the feasibility of remote precision in liquid pouring via complete autonomy. When broaching liquid pouring tasks in robotics, the issues of diversity and trainability often surface. A deficiency in ample datasets curtails the efficacy of robotic learning, especially in liquid pouring tasks. However, simulated liquid datasets, crafted in a virtual milieu, can bolster dataset diversity, presenting myriad simulation possibilities that cater to robot learning prerequisites. We dub our simulator the "Realistic Robotic Simulator for Pouring Liquids (RRS-PL)".

The dissertation culminates with a methodological approach for robot training in liquid pouring. Liquids present an intricate puzzle for robots, mainly due to their obliviousness to the multifaceted fluid dynamics and behavior of these liquids. For real-time engagements, a universal pouring policy remains elusive. We present "PourNet" as a significant contribution to this domain—a generalized remedy for pouring diverse liquids into varied containers. PourNet is an eclectic planner amalgamating deep reinforcement learning for end-effector planning and Nonlinear Model Predictive Control for joint planning. We train the pouring agent in this simulation backdrop using Proximal Policy Optimization (PPO) as the learning algorithm, with force/torque feedback as the dominant perception modality. Strategic selections of the state space, action space, and reward functions enable a direct sim-to-real skills transfer, obviating the need for auxiliary training. We integrate a curiosity-driven reward system, motivating agents to incessantly expand their internal knowledge repertoire. Concurrently, we randomize liquid parameters via a curriculum-centric learning approach. In simulations, PourNet trumps its competitors by a margin of 4.9g deviation for water-like substances and 9.2g for honey-like mediums. In real-world trials using the Kinova[®] Movo Platform, PourNet boasts an average pouring discrepancy of 2.3g for dish soap with a pioneering pouring container. For water, the deviation averages at 5.5g. All encompassing experiments and accompanying visuals are accessible at: <http://www.5g-munich.de>.

Kurzfassung

In dieser Dissertation haben wir Assistenzfunktionen für die Teleoperation untersucht und hinsichtlich einer zuverlässigen und sicheren Teleoperation für das Handhaben/Ausgießen von Flüssigkeiten erweitert. Die geteilte Autonomie gilt als eine der kritischsten Aspekte unterstützten Teleoperation. Das erste Thema der Dissertation ist die Bewegungsplanung von Manipulatoren. Die Teleoperation von Roboter-Manipulatoren hängt von der Bewegungssteuerung und -planung ab. Die Verwendung von Online-Bewegungssteuerung ist eine Herausforderung für die aktive Vermeidung von Hindernissen und führt zu unsicheren Bewegungen. Im Gegensatz dazu generiert die Offline-Bewegungsplanung präzise und sichere Trajektorien für komplexe Manipulationen. Wir präsentieren einen nicht-linearen modellprädiktiven steuerungsbasierten Bewegungsplaner (NMPC-MP) in Echtzeit für die teleoperierte Manipulation. Unser Modell berücksichtigt im Gegensatz zu herkömmlichen NMPC-basierten Modellen eine komplexe Umgebung mit dynamischen Hindernissen. Echtzeit-Planung ist mit unserem Multi-Threaded NMPC-MP möglich. Unter Verwendung der Kinova[®] Movo-Plattform evaluieren wir unseren Ansatz sowohl in einer simulierten als auch in einer realen Umgebung. Der Vergleich von NMPC-MP mit modernen Ansätzen (z. B. RRT-Connect, CHOMP und STOMP) zeigt deutliche Verbesserungen bei der Echtzeit-Bewegungsplanung. Dieser Planer wurde in realen Tests auf einen Dual-Manipulator-Aufbau angewendet. Experimentelle Ergebnisse zeigen, dass Planer die aktiven Ziele des Teleoperators genau verfolgen und dabei Selbstkollisionen und Hindernisse vermeiden können. Als Teil des zweiten Themas der Dissertation diskutieren wir geteilte Autonomie in Gegenwart von Kommunikationsstörungen. Drahtlose Kommunikationsnetzwerke bieten eine geringere Latenz und höhere Übertragungsraten. Das Risiko von Netzwerkinstabilität oder Paketverlust besteht immer noch, obwohl dies viele neue Teleoperationsanwendungen ermöglicht. Teleoperation in Echtzeit erfordert eine unterbrechungsfreie Datenübertragung. Shared Autonomy (SA) mildert dieses Problem. Wenn die Ferndaten nicht verfügbar sind, kann der Controller basierend auf den zuvor beobachteten Modellen fortfahren. Aufgrund der räumlichen Lücke zwischen den Bewegungsbahnen von Mensch und Roboter treten unbestreitbare Schwankungen auf, die Probleme für Teleoperationsanwendungen verursachen. Daher schlagen wir eine neuartige Strategie zur Verfeinerung von Fähigkeiten vor, um zuvor trainierte Fähigkeiten zu modifizieren und die plötzlichen unerwünschten Bewegungen abzumildern, die während der Kontrollübernahmephase auftreten. Zu diesem Zweck umfasst unser Ansatz die kombinierte Anwendung des Hidden Semi-Markov-Modells (HSMM) und des Linear Quadratic Tracker (LQT), um die Absichten

des Benutzers zu lernen und vorherzusagen, und die anschließende Nutzung von Coherent Point Drift (CPD), um die ausführbare Trajektorie zu verfeinern. Wir testen unsere Methode sowohl in der Simulation als auch in der realen Welt für das Zeichnen von englischen Buchstaben in 2D und robotergestützte Fütterungsszenarien in 3D. Unsere experimentellen Ergebnisse mit der Kinova[®] Movo-Plattform zeigen, dass der vorgeschlagene Verfeinerungsansatz eine stabile Trajektorie erzeugt und die Inkonsistenz der Steuerungsumschaltung mildert. Das dritte Thema konzentriert sich auf die Roboter-Flüssigkeits-Interaktion. Durch die Verwendung eines digitalen Zwillings des Manipulators und eines neuartigen Echtzeit-Flüssigkeitssimulators untersuchen wir die Möglichkeit des präzisen Flüssigkeitsgießens aus der Ferne mit vollständiger Autonomie. Für die Entwicklung von Flüssigkeitsgießaufgaben in der Robotik sind Diversität und Trainierbarkeit zu großen Herausforderungen geworden. Der Mangel an ausreichenden Datensätzen schränkt die Forschungseffektivität des Roboterlernens ein, insbesondere im verwandten Bereich der Aufgaben zum Gießen von Flüssigkeiten. In der virtuellen Umgebung generierte simulierte Datensätze können die Vielfalt der Daten erheblich erhöhen und eine Reihe von Simulationsmöglichkeiten bieten, um die Anforderungen des Roboterlernens zu erfüllen. Realistic Robotic Simulator for Pouring Liquids (RRS-PL) ist der Name unseres Simulators. Die Dissertation endet mit einer Methode zum Trainieren von Robotern zum Ausgießen von Flüssigkeiten. Flüssigkeiten sind eine der herausforderndsten Aufgaben für Roboter, da sie sich der komplexen Fluidodynamik und des Verhaltens von Flüssigkeiten nicht bewusst sind. Für Echtzeitanwendungen ist eine generische Pouring-Policy nicht möglich. Als wichtigen Beitrag zu diesem Thema schlagen wir PourNet als allgemeine Lösung zum Einfüllen verschiedener Flüssigkeiten in Behälter vor. PourNet ist ein hybrider Planer, der Deep Reinforcement Learning für die Endeffektor-Planung und Nonlinear Model Predictive Control für die gemeinsame Planung verwendet. Das Gießmittel wird in dieser Simulationsumgebung unter Verwendung von Proximal Policy Optimization (PPO) als Lernalgorithmus und Kraft-/Drehmoment-Feedback als primärem Wahrnehmungssystem trainiert. Durch die effektive Wahl des Zustandsraums, des Aktionsraums und der Belohnungsfunktionen ermöglichen wir einen direkten Sim-to-Real-Transfer der erlernten Fähigkeiten ohne zusätzliches Training. Ein auf Neugier basierendes Belohnungssystem gibt Agenten einen Anreiz, ihr Wissen intern zu erweitern. Darüber hinaus werden die Flüssigkeitsparameter durch ein Curriculum-basiertes Lernverfahren randomisiert. In der Simulation übertrifft PourNet den Stand der Technik um durchschnittlich 4,9 g Abweichung für wasserähnliche und 9,2 g Abweichung für honigähnliche Flüssigkeiten. Im realen Szenario mit Kinova Movo Plattform erreicht PourNet eine durchschnittliche Ausgießabweichung von 2,3 g für Spülmittel bei Verwendung eines neuartigen Ausgießbehälters. Die für Wasser gemessene durchschnittliche Gießabweichung beträgt 5,5 g.

Acknowledgements

The work presented in this dissertation was carried out as a member of the academic staff at the Chair of Media Technology (LMT) at the Technical University of Munich. Over the past years, many people have supported me, both personally and professionally.

First of all, I would like to express my deep gratitude to my supervisor Prof. Dr.-Ing. Eckehard Steinbach. It has been an honor to be part of his team. He provided me with unwavering support and valuable remarks that were instrumental in shaping my final dissertation. His professional approach and enthusiasm for my research were both contagious and inspiring.

Furthermore, I would like to thank Prof. Dr. Cristina Piazza for agreeing to serve as the second examiner and Prof. Dr.-Ing. Bernhard U. Seeber for chairing the thesis committee.

I extend my thanks to our project partner, the Bavarian Ministry of Economic Affairs, Regional Development, and Energy. Their genuine support and funding made our collaboration both intriguing and fruitful.

I am also grateful to my past and current colleagues at LMT. This group has been a source of friendships as well as invaluable advice and collaboration. Beyond professional assistance, they enriched my time at LMT personally. I am especially grateful to Dr. Xiao Xu for jump-starting my research with his extensive knowledge and experience. My thanks extend to all my colleagues at LMT and the Chair of Communication Networks (LKN), but I feel compelled to spotlight a few: Dr. Rahul Chaudhari, Jingyi Xu, Basak Gülecyüz, Serkut Ayvasik, Nemanja Deric, Hasan Furkan Kaynar, Martin Piccolrovazzi, and Tamay Aykut. Their insights and discussions were crucial to my research.

Further gratitude goes to all my students, particularly Siqu Hu, Dong Yang, and Tapan Sharma, who made significant contributions to our publications. I also appreciate the steadfast administrative support from Dr. Martin Maier, Simon Krapf, Marta Giunta, and Etienne Mayer. Special thanks are reserved for my mentor, Prof. Dr.-Ing. Wolfgang Kellerer, who offered consistent spiritual support and exceptional mentorship throughout my Ph.D.

Lastly, my heartfelt thanks go to my family. Special mentions to my parents, Karmen Gibreil and Emil Babaians, my partner Iryna Savchuk, and my closest friends Sahand Sharifzadeh and Mojtaba Leox Karimi. Their unwavering belief in me and their constant encouragement have been my pillars of strength.

Edwin Babaians, Munich, May 15, 2022

Contents

Notation	xiii
1 Introduction	1
1.1 Thesis Outline	5
2 Background and Related Work	7
2.1 Redundant Robots	7
2.2 Forward Kinematics and Jacobian Matrix	7
2.2.1 Resolved motion rate control	9
2.3 IK with the Order of Priority	9
2.3.1 General Formulation of Prioritized Subtasks	9
2.3.2 The Damped Pseudo-inverse	11
2.3.3 Obstacle Avoidance by Potential Function	11
2.3.4 Obstacle Avoidance as Prioritized Task	12
2.4 Offline Motion Planning	13
2.4.1 Sampling-based motion planners	13
2.4.2 Optimization-based Motion Planners	13
2.4.2.1 CHOMP	14
2.4.2.2 STOMP	15
2.5 Introduction to MPC	16
2.5.1 Models	17
2.6 Hidden Semi-Markov Model	18
2.6.1 Discussion about other State-of-the-art Approaches	19
2.6.2 Linear Quadratic Tracker	20
2.6.2.1 Continuous-time LQT	22
2.7 Point Set Registration	23
2.7.1 Coherent Point Drift Algorithm	25
2.8 Reinforcement Learning	27
2.8.1 Proximal Policy Optimization	28
2.9 Liquid Simulation	29
2.9.1 Position Based Fluids	30
2.9.2 Enforcing Incompressibility	30
2.9.3 Tensile Instability	32

2.9.4	Vorticity Confinement and Viscosity	33
2.10	Related Work	33
2.11	Chapter Summary	37
3	Experimental Setup and Datasets	39
3.1	Movo Telepresence Platform	39
3.2	Upgrading Movo’s Hardware	40
3.2.1	Setup of a New Jaco2 Robot Model with Hand-e Gripper	40
3.2.2	Force/Torque Sensor and Filtering	41
3.2.3	Low-Pass Filter	41
3.2.4	Installation of the Real-sense camera	42
3.2.5	Gravity Compensation	43
3.3	Upgrading Movo’s Software	44
3.3.1	Shared Autonomy Interface (SAI)	44
3.3.2	NVIDIA FLEX	46
3.3.3	Digital Twin and PhysX Implementation	47
3.3.4	Tablet Interface	47
3.4	Datasets	48
3.4.1	YCB Standard Model Dataset	48
3.4.2	Liquid Profile Dataset	48
3.4.2.1	Liquids	48
3.4.2.2	Density and Viscosity	49
3.5	Chapter Summary	53
4	Nonlinear Model Predictive Control for Real-time Motion Planning	55
4.0.1	Linear MPC regulator	56
4.1	Forward kinematics and Jacobin for robot in real world	60
4.1.1	Forward kinematics	60
4.1.2	Orientation representation	62
4.1.3	Jacobian	64
4.2	Model predictive control for motion planning	66
4.2.1	Optimization problem formulation	67
4.2.2	Objective function	69
4.2.3	Constraints	71
4.2.4	Sequential quadratic programming method	73
4.2.5	Real-time scheme and parallelization	75
4.3	Experiments and results	77
4.3.1	Simulations	77
4.3.2	Comparison scenarios	81
4.3.3	Experiments on the real Movo robot	83
4.4	Adapting NMPC-MP for Franka	84
4.4.1	Forward Kinematic Calculation	84
4.5	Simulated Franka	85

4.5.1	Reconfiguration of cylinders	85
4.5.2	Optimization with limits	86
4.6	Real Franka	88
4.7	Chapter Summary	90
5	Real-time Skill Refinement for Shared Autonomy in Manipulator Teleoperation	91
5.1	Problem Statement	92
5.2	System Architecture	93
5.3	Technical Details on Skill-CPD	93
5.3.1	Hidden Semi-Markov Model (HSMM)	93
5.3.2	Linear Quadratic Tracker (LQT)	94
5.3.3	Coherent Point Drift (CPD) Algorithm	95
5.3.4	Skill Registration	98
5.3.4.1	Trajectory Sampling and Step Indicator	98
5.3.4.2	Adaptive Registration and Switching Strategy	98
5.3.4.3	Reverse Takeover Strategy	99
5.4	Experimental Evaluation	99
5.4.1	Experimental Setup	99
5.4.2	Task(i): 2D English Letter Experiments in Simulation	100
5.4.3	Task(ii): 2D English Letter Experiments in Reality	101
5.4.4	Task(iii): 3D robot-assisted Feeding in Reality	101
5.4.5	Discussion	102
5.5	Chapter Summary	103
6	Liquid Pouring Through Curriculum and Curiosity-based Reinforcement Learning	105
6.1	Liquid Property Awareness	106
6.2	Operating Environment Awareness	107
6.3	Pouring Container Geometry Awareness	108
6.4	Problem Statement	109
6.5	Technical Details on PourNet	109
6.5.1	Average Pouring Error	109
6.5.2	Proximal Policy Optimization	109
6.5.3	Action Handler	110
6.5.4	Reward Shaping	110
6.5.5	Intrinsic Curiosity Module	110
6.5.6	Curriculum Learning	111
6.5.6.1	Designing Pouring Curriculum	112
6.5.7	End-to-end Technical Specifications	113
6.5.8	PourNet's Architecture	114
6.5.8.1	LSTM-based PourNet	115
6.5.8.2	Feed-forward neural network-based PourNet	115
6.6	Experimental Evaluation	115
6.6.1	Simulation	115

6.6.1.1	Simulation Setup	115
6.6.1.2	Training	116
6.6.1.3	Discussion	116
6.6.2	Real Experiments	117
6.6.2.1	Platform Setup	117
6.6.2.2	Weight Measurement	117
6.6.2.3	Experimental Containers and Liquids	118
6.6.2.4	Pouring Experiments	118
6.6.2.5	Discussion	120
6.7	Chapter Summary	120
7	Conclusion and Future work	123
7.1	Summary	123
7.2	Limitations	124
7.3	Future Work	124
	Bibliography	125
	List of Figures	135
	List of Tables	139

Notation

Abbreviations

Abbreviation	Description	Definition
ADL	Activities of daily living	page 1
HRI	Human Robot Interaction	page 2
CPD	Coherent Point Drift	page 2
RL	Reinforcement Learning	page 2
IK	Inverse Kinematics	page 7
FK	Forward Kinematics	page 7
DOF	Degree of Freedom	page 8
HSMM	Hidden Semi-Markov Models	page 18
HMM	Hidden Markov Models	page 18
LQT	Linear Quadratic Tracker	page 20
GMM	Gaussian Mixture Model	page 20
SAC	Soft Actor Critic	page 28
SA	Shared Autonomy	page 34
SC	Shared Control	page 34
2D	Two-dimensional	page 35
3D	Three-dimensional	page 35
PID	Proportional–integral–derivative	page 35
ROS	Robotic Operating System	page 39
API	Application Programming Interface	page 39
URDF	Unified Robot Description Format	page 40
IMU	Inertial Measurement Unit	page 42
EE	End Effector (of the robot arm)	page 43
SAI	Shared Autonomy Interface	page 44
GUI	Graphical User Interface	page 45
TGS	Temporal Gauss-Seidel	page 47
NMPC	Nonlinear Model Predictive Control	page 55
FCL	Flexible Collision Library	page 80
PPO	Proximal Policy Optimization	page 109
RRS-PL	Realistic Robotic Simulator for Pouring Liquids	page 115
FNN	Fully Connected Neural Network	page 115
LSTM	Long short-term memory	page 118
GAIL	Generative Adversarial Imitation Learning	page 124

Scalars and vectors

x	scalar
\boldsymbol{x}	vector
\boldsymbol{X}	matrix
$ x $	absolute value of scalar x
$\ \boldsymbol{x}\ $	Euclidean norm of vector \boldsymbol{x}

Subscripts and superscripts

\bar{x}	mean of x
\hat{x}	estimated/predicted value of x

Symbols

$J(q)$	Jacobian matrix of q
--------	------------------------

Chapter 1

Introduction



Figure 1.1: Dual hand liquid pouring. One of the basic requirements of a dual arm motion planning system is self-awareness. In a sense, each manipulator is a dynamic obstacle to the other. [1]

In the last century, healthcare and lifestyle improvements have made people live longer. Despite longevity being a significant achievement of modern society, it poses challenges in terms of caring for an elderly population. There is a supply and demand issue for elder care. The number of caregivers is not increasing as the elderly population increases. A relative lack of caregivers keeps elderly care costs high and places a heavy burden on families and caregivers. Robotics can address the supply and demand issue in elder care. Elderly care costs will be drastically reduced when medical robots are used. Medical robotics can be used to help the elderly in many ways, including fetching food and water. As people get older, their physical capabilities decrease: not only mobility decreases, but also sight, precision, and strength. Therefore, they need assistance with most activities of daily living (ADL). Performing daily tasks by themselves is no longer feasible for elderly people due to their lack of coordination. Among the ADLs that service robots would encounter, tasks that involve liquids deserve special attention. The use of robots in activities that involve liquids, on the other hand, has not been the focus of attention until recent years. Household activities are not the only ones in which we encounter liquids, since a lot of medicine comes in liquid form, such as pills, cough syrup, and vaccines. Therefore, pouring liquids without spilling and controlling precise quantities have become important research topics.



Figure 1.2: Image (a) illustrates an example application for rigid object manipulation [2]. An example of liquid handling and feeding is shown in image (b). Image (c) illustrates precise liquid pouring. Here is an example of medicine in the form of syrup [3].

Table 1.1: Summarizing levels of autonomy in this dissertation.

Thesis Chapter	Chapter 4	Chapter 5	Chapter 6
Arbitration	Blend	Binary	Only Robot
Methodology	NMPC	HSMM/LQT/CPD	PPO/RL
HRI Level	high	high	low
Environment	dynamic	dynamic	dynamic
Task Complexity	simple	moderate	extreme
Elderly Care Task	rigid object manipulation	handling liquids	pouring liquids
Input Device	Sigma7/HTC Vive	Ph. Omni/HTC Vive	Tablet
Mode	Shared Autonomy	Shared Autonomy	Autonomous

Using robots to provide care for the elderly is one of the most active areas of research. Teleoperation offers a promising opportunity for assisting as many elderly with limited care givers as possible. Teleoperation is an aspect of human-robot interaction (HRI) in which the nurse (care provider) communicates remotely with a robot. The concept of autonomy is a critical component of HRI that differs considerably between platforms. Various levels of robot autonomy, from teleoperation to fully autonomous systems, impact the way humans and robots interact with one another. According to the level of autonomy within HRI, we can divide teleoperation into three major categories. The first type of control is direct control, in which the robot follows the movements of the teleoperator directly. A second way of controlling the robots is through the shared control process. This is where the robots correct their own behaviors based on local sensor feedback. The third way is through the supervisory control process where the robot performs the actions independently.

There are tasks which robots are currently not able to perform independently due to improper regulation or tasks that require human-level dexterity. These tasks include remote food feeding, medical examinations, injections, and general nurse duties. Although robots are capable of performing some tasks independently, teleoperators are not able to execute them remotely due to the limited setting or lack of transparent sensory feedback. For instance, indoor navigation in dynamic environments and picking up and placing objects accurately and efficiently are examples of these tasks.

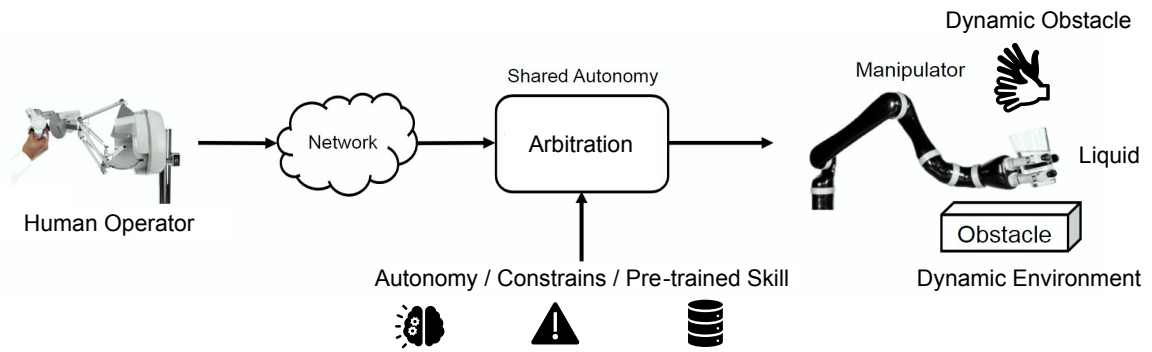


Figure 1.3: Assistive teleportation in dynamic environments.

Table 1.1 summarizes the HRI levels in teleoperation and related methodologies which are covered in this dissertation.

The first topic in this dissertation is motion planning in dynamic environments. This is also important in dual arm systems as the one manipulator sees the other one a dynamic obstacle. Figure 1.1 shows our experimental setup for shared autonomy, consisting of an operator interface and HTC[®] VR HMD and the Movo platform as the telerobot with two arm-based manipulators. A common problem in teleoperating a robot, is the lack of dexterity in the human operator's input. Typically, only the end-effector's posture is under the direct control of a human operator. Other parts of the manipulator only serve to take the end-effector to the goal posture. Therefore, they usually have redundancy in autonomy and can not be directly controlled. This may result in dangerous situations. Therefore, this redundancy should be exploited to ensure obstacle and self-collision avoidance. Besides, the manipulator also needs to have an autonomous reaction when the teleoperation commands are cut or delayed due to unfavorable network conditions. This combination of an autonomous system and a human operator that controls a robot, is called shared autonomy.

For the second topic of this dissertation, we adopt a more general way to mitigate network latency in teleoperation by leveraging machine learning and shared autonomy [4]. To do this, our system is able to perform direct teleoperation until network instability is detected. The intention of the human teleoperator is recognized at the same time. When the communication network is affected and thus becomes unstable, our system will switch to the complete autonomy mode. The performance of reproduction on the robot side could be also affected by varying environmental situations. Therefore, we are motivated to develop a new skill refinement method to improve the performance of task reproduction. In order to fit the reproduced trajectory and the demonstrated trajectory better, the skill refinement in our work is viewed as a point set registration problem. The goal of point set registration is to assign correspondences between two point sets and to recover the transformation that maps one point set to the other [5]. The Coherent Point Drift (CPD) algorithm is applied in our system due to its more robust and accurate performance compared with other methods. The CPD algorithm is a probabilistic method that converts the alignment of two point sets to a probability density estimation problem, where one point set represents the Gaussian Mixture Model (GMM) centroids, and the other one represents the data points [5]. With our skill

refinement in task reproduction, the experimental results reveal that our system improves the performance of the remote manipulation task in teleoperation by mitigating the effect of imprecise movements, especially the intersection part of the partial demonstration and the reproduction.

For the third topic in this dissertation independent liquid pouring has been investigated. With the advent of automation in our industries to the applications in our homes, pouring liquids precisely and efficiently is becoming an indispensable skill for robotics. In this dissertation, a robust policy is developed called "PourNet", which can perform precision pouring actions for any liquid type, provided that the liquid parameters are available. We consider precision robotic pouring as a "Sparse Reward Reinforcement Learning Problem". The approach for precision pouring can be made better by incorporating a large number of experiences while working with a multitude of liquid types. This ensures that the precision pouring policy is representative of many liquid types which may behave differently in their pouring dynamics because of their different properties like viscosity, surface tension, etc. Since acquiring a large number of experiences from the real world can be time and cost expensive, this thesis discusses the ability to gain precision pouring skill in a pure simulation environment. The thesis further considers that a policy trained on domain randomized liquid properties in the simulation can be transferred to a real robot for planning pouring actions while minimizing pouring errors. Figure 1.4 summarizes the major topics of the dissertation.

The key contributions of the work presented in this manuscript can be summarized as follows:

- Proposal of a motion planner based on nonlinear model predictive control.
- Implementation of warm start and multi-threading mechanisms, which accelerate the motion planning and enables it to be applied in real-time for dynamic obstacles.
- Design of the novel online skill refinement approach using the CPD algorithm.
- Implementation of a shared control switching and reverse takeover strategy with minimum displacement.
- Setup robust and representative pouring scenario in simulation using Unity3D and Nvidia Flex.
- Introduce an internal reward based on curiosity regarding the liquid pouring policy. This makes an agent capable of growing its knowledge itself. Furthermore, it provides a delicate balance between "exploration" and "exploitation" of the emerging precision pouring policy.
- Implement Curriculum Learning for liquid parameter randomization. This enables the pouring policy's generalization to many liquid types by gaining experience, with increasing difficulty as the lessons in curriculum progress.

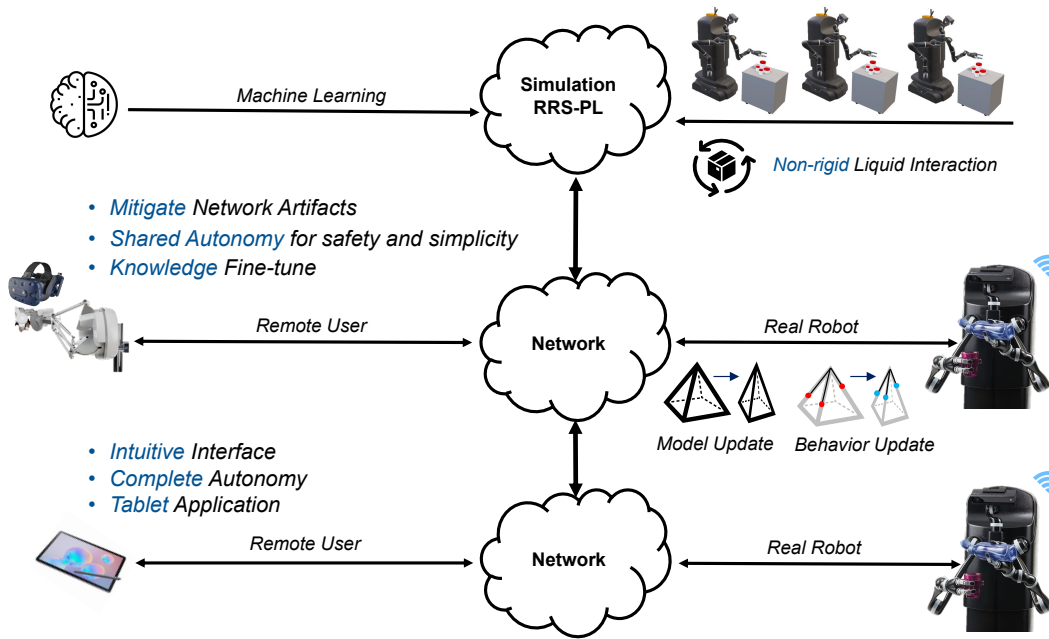


Figure 1.4: Dissertation big picture.











Problem(s)	Current solution	Our improvement(s)	Mode
 Static obstacles Dynamic obstacles	Model predictive control (MPC)	Warm start Multi-thread Using on bimanual robot	 +  MPC
 Network impairments	Intention recognition Shared autonomy HSMM + LQT	Minimum displacement in control switching	 Or  +  ML + MPC
 Bimanual pouring	No solution!	Achieving up to 10 g accuracy Using RL	 +  ML + MPC

Figure 1.5: Overview of research questions and subsequent improvements.

1.1 Thesis Outline

Chapter 2 introduces the theoretical background needed to understand the proposed concepts and the decisions made within the scope of this work. The proposed methodology is confronted with related work. The main advantages are discussed and compared to the state-of-the-art.

The experimental framework is described in **Chapter 3**. In order to gain a deeper understanding of the methodologies, we examine the performed upgrade of MOVO's grippers and head in relation to its hardware specifications. An additional presentation illustrates and discusses the software architecture, shared autonomy interface, liquid pouring simulator, and two datasets for network interaction study from a motion generation perspective

and simulated liquid profiles from the policy training perspective to facilitate reproducible validation.

Chapter 4 introduces the model predictive control approach and establishes a generic, algebraic formulation of the continuous motion planner for a redundant manipulator.

Chapter 5 analyzes the effects of the network impairments on teleoperation in dynamic environments. This section describes the skill refinement methodology as a non-rigid point set registration problem. Deep Reinforcement learning-based approach to design a hybrid motion planner is described in **Chapter 6**. The proposed deep reinforcement learning paradigm is compared to prior art and evaluated by different liquid profiles on both simulation and real experiments.

Chapter 7 concludes this work by summarizing the most distinct outcomes of this manuscript, discussing their limitations, and briefly broaching some potential conceptual improvements for future work.

Parts of this manuscript have been published in [6], [1], [7], [8], [9]. Contribution to other publications are [10], [11], [12].

Chapter 2

Background and Related Work

2.1 Redundant Robots

The main purpose of adopting redundancy in robot systems is to improve reliability and safety. For example, in most industrial robots, end-effectors need to be arbitrarily positioned in a three-dimensional workspace, and usually, six actuators are mounted in order to perform the task. And when tasks are defined in a six-dimensional workspace for position and orientation, a robot manipulator with seven or more joint actuators is used for the tasks. In this case, the robot is said to have kinematic redundancy. Redundancy is useful for not only the reliability but also the dexterity of the manipulator.

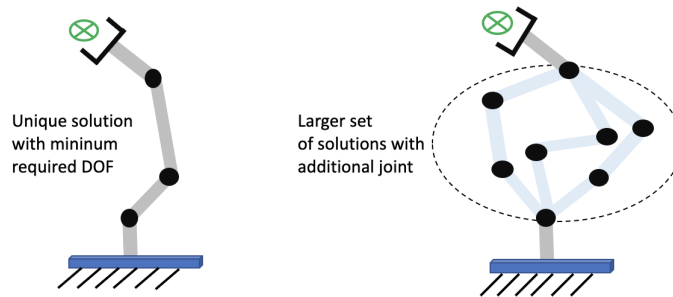


Figure 2.1: This figures illustrates the kinematic redundancy of a simple interconnected kinematic chain: thanks to one additional joint, the kinematics solver can find a large set of solutions.

2.2 Forward Kinematics and Jacobian Matrix

The Kinematics of a robot consists of Forward Kinematic (FK) and Inverse Kinematic (IK). FK is the computation needed to obtain the cartesian position and orientation of the end-effector given joint angles. And IK is to solve the joint angles from the given cartesian position and orientation of the end-effector. While FK can be represented by a closed-form nonlinear algebra computation, there is no analytical closed-form solution for IK. Besides, for a redundant robot, the number of solutions for a given cartesian position can be infinite. Thus, in Nakamura's control scheme [13], directly calculating the joint angles from cartesian position via

IK is avoided. Instead, the Jacobian matrix of the FK is used to calculate joint velocities. The Jacobian matrix represents the linear relationship between the cartesian velocities of the end-effector and the joint velocities. And the Jacobian matrix itself is nonlinearly and uniquely determined by the joint angles. Based on these facts, a resolved motion rate control can be implemented to control the end-effector to move to the desired position.

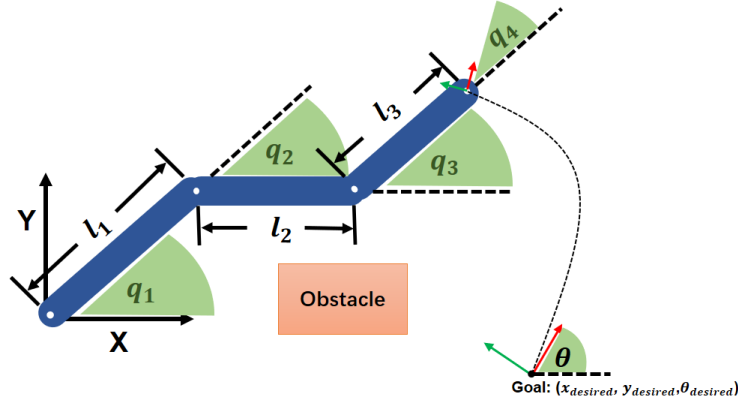


Figure 2.2: Simulated planar robot manipulator

Since the structure of the four Degrees of Freedom (DOF) planar robot is simple, the corresponding FK, and Jacobian matrix can be analytically derived in an intuitive way:

$$\mathbf{X}(\mathbf{q}) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} l_1 c_1 + l_2 c_{12} + l_3 c_{123} \\ l_1 s_1 + l_2 s_{12} + l_3 s_{123} \\ q_1 + q_2 + q_3 + q_4 \end{bmatrix} \quad (2.1)$$

$$\mathbf{J}(\mathbf{q}) = \frac{d\mathbf{X}}{d\mathbf{q}} = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} - l_3 s_{123} & -l_2 s_{12} - l_3 s_{123} & -l_3 s_{123} & 0 \\ l_1 c_1 + l_2 c_{12} + l_3 c_{123} & l_2 c_{12} + l_3 c_{123} & l_3 c_{123} & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (2.2)$$

where $\mathbf{X} \in R^m$ represents the position of the end-effector and $\mathbf{J} \in R^{m \times n}$ the Jacobian Matrix, $\mathbf{q} \in R^n$ is the joint angles vector, namely $[q_1, q_2, q_3, q_4]^T$. m and n represent the dimension of cartesian space and the number of robot joints respectively, here in the simulation of the planar robot, m is equal to 3 and n is 4. l_1 to l_3 are the lengths of the links. c_1 is for the abbreviation of $\cos(q_1)$ and similarly s_{123} for $\sin(q_1 + q_2 + q_3)$.

Differentiating both sides of Eq 2.1 with respect to time leads to:

$$\dot{\mathbf{X}} = \frac{d\mathbf{X}}{dt} = \frac{d\mathbf{X}}{d\mathbf{q}} \cdot \frac{d\mathbf{q}}{dt} = \frac{d\mathbf{X}}{d\mathbf{q}} \cdot \dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \quad (2.3)$$

Substituting Eq 2.2 into Eq 2.3 results in:

$$\dot{\mathbf{X}} = \mathbf{J}(\mathbf{q}) \cdot \dot{\mathbf{q}} \quad (2.4)$$

where a linear relationship between the velocity in cartesian space and the joint velocities is formulated.

2.2.1 Resolved motion rate control

Given the desired velocities in cartesian space, the joint velocities can be obtained by solving Eq 2.4. However, for the $\mathbf{J}(\mathbf{q}) \in R^{m \times n}$ of a redundant robot, where $n > m$, solving Eq 2.4 for $\dot{\mathbf{q}}$ yields infinitely many solutions. In order to make the solution unique, one good measurement is to select the solution that minimizes $\dot{\mathbf{q}}$. In this way, the problem is converted to the following:

$$\min \|\dot{\mathbf{q}}\|^2, \quad \text{s.t.} \quad \dot{\mathbf{X}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \quad (2.5)$$

Using Lagrange multipliers, we have:

$$L(\dot{\mathbf{q}}, \boldsymbol{\lambda}) = \|\dot{\mathbf{q}}\|^2 + \boldsymbol{\lambda}^T (\dot{\mathbf{X}} - \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}) \quad (2.6)$$

Differentiating Eq 2.6 with respect to $\dot{\mathbf{q}}$ yields:

$$\frac{\partial L}{\partial \dot{\mathbf{q}}} = 2\dot{\mathbf{q}} - \mathbf{J}(\mathbf{q})^T \boldsymbol{\lambda} \quad (2.7)$$

Letting $\frac{\partial L}{\partial \dot{\mathbf{q}}}$ be 0 and multiplying both sides by $\mathbf{J}(\mathbf{q})$:

$$2\mathbf{J}(\mathbf{q})\dot{\mathbf{q}} - \mathbf{J}(\mathbf{q})\mathbf{J}(\mathbf{q})^T \boldsymbol{\lambda} = 0 \quad (2.8)$$

Substituting Eq 2.4 into Eq 2.8 and solve for $\boldsymbol{\lambda}$:

$$\begin{aligned} 2\dot{\mathbf{X}} &= \mathbf{J}(\mathbf{q})\mathbf{J}(\mathbf{q})^T \boldsymbol{\lambda} \\ \boldsymbol{\lambda} &= 2(\mathbf{J}(\mathbf{q})\mathbf{J}(\mathbf{q})^T)^{-1} \dot{\mathbf{X}} \end{aligned} \quad (2.9)$$

Now substituting Eq 2.9 into Eq 2.7:

$$\begin{aligned} \dot{\mathbf{q}} &= \mathbf{J}^\# \dot{\mathbf{X}} \\ \mathbf{J}^\# &= \mathbf{J}(\mathbf{q})^T (\mathbf{J}(\mathbf{q})\mathbf{J}(\mathbf{q})^T)^{-1} \end{aligned} \quad (2.10)$$

where $\mathbf{J}^\#$ represents the pseudo-inverse of $\mathbf{J}(\mathbf{q}) \in R^{m \times n}$ when $\mathbf{J}(\mathbf{q})$ is under-determined, i.e. $m < n$.

2.3 IK with the Order of Priority

2.3.1 General Formulation of Prioritized Subtasks

A task can be divided into multiple subtasks with the order of priority. A subtask is defined by a manipulation variable, $\mathbf{s}_i \in R^{m_i}$, where i specifies the order of the subtask and m_i the

dimension of the manipulation variable for this subtask. Then the relationship between the manipulation variable and the joint angles can be represented by:

$$\mathbf{s}_i = \mathbf{f}_i(\mathbf{q}) \quad (2.11)$$

and the differential relationship is expressed by:

$$\dot{\mathbf{s}}_i = \mathbf{J}_i(\mathbf{q})\dot{\mathbf{q}} \quad (2.12)$$

As discussed in the previous section, for the Jacobian matrix $\mathbf{J}_i \in R^{m_i \times n}$, if $m_i < n$, solving Eq 2.11 yields infinite many solutions for $\dot{\mathbf{q}}$. The set of this infinite variety of solutions can be analytically derived via the pseudo-inverse of \mathbf{J}_i :

$$\dot{\mathbf{q}} = \mathbf{J}_i^\#(\mathbf{q})\dot{\mathbf{s}}_i + \{\mathbf{I}_n - \mathbf{J}_i^\#(\mathbf{q})\mathbf{J}_i(\mathbf{q})\}\mathbf{v} \quad (2.13)$$

where $\mathbf{I}_n \in R^{n \times n}$ is an identity matrix and matrix $\{\mathbf{I}_n - \mathbf{J}_i^\#(\mathbf{q})\mathbf{J}_i(\mathbf{q})\}$ represents a projection of an arbitrary vector $\mathbf{v} \in R^n$ onto the nullspace of \mathbf{J}_i . Any motion in the nullspace doesn't affect the motion in the task space. And $\mathbf{v} = \mathbf{0}$ indicates the solution with the minimum norm.

Now consider a task consisting of two subtasks. The first subtask is fulfilled by using Eq 2.13. Substitute Eq 2.13 for the first subtask into Eq 2.11 for the second subtask, in order to find a vector \mathbf{v} whose projection onto the nullspace of the Jacobian matrix for the first task helps fulfill the second subtask and doesn't affect fulfilling the first subtask. Then the following equation is obtained:

$$\dot{\mathbf{s}}_2 - \mathbf{J}_2\mathbf{J}_1^\# \dot{\mathbf{s}}_1 = \mathbf{J}_2(\mathbf{I}_n - \mathbf{J}_1^\# \mathbf{J}_1)\mathbf{v} \quad (2.14)$$

If the solution of \mathbf{v} for Eq 2.14 exists, the second subtask can be achieved. However, such solution doesn't always exist. In that case, a \mathbf{v} is only obtained to minimize $\|\dot{\mathbf{s}}_2 - \mathbf{J}_2\dot{\mathbf{q}}\|$. Eq 2.14 is of the same form as Eq 2.11 and can be solved for \mathbf{v} in the same way:

$$\begin{aligned} \mathbf{v} &= \hat{\mathbf{J}}_2^\# (\dot{\mathbf{s}}_2 - \mathbf{J}_2\mathbf{J}_1^\# \dot{\mathbf{s}}_1) + (\mathbf{I}_n - \hat{\mathbf{J}}_2^\# \hat{\mathbf{J}}_2)\mathbf{w} \\ \hat{\mathbf{J}}_2 &\doteq \mathbf{J}_2(\mathbf{I}_n - \mathbf{J}_1^\# \mathbf{J}_1) \end{aligned} \quad (2.15)$$

Substituting Eq 2.15 into Eq 2.13 for first subtask, solution of $\dot{\mathbf{q}}$ is obtained:

$$\begin{aligned} \dot{\mathbf{q}} &= \mathbf{J}_1^\# \dot{\mathbf{s}}_1 + (\mathbf{I}_n - \mathbf{J}_1^\# \mathbf{J}_1)\hat{\mathbf{J}}_2^\# (\dot{\mathbf{s}}_2 - \mathbf{J}_2\mathbf{J}_1^\# \dot{\mathbf{s}}_1) \\ &\quad + (\mathbf{I}_n - \mathbf{J}_1^\# \mathbf{J}_1)(\mathbf{I}_n - \hat{\mathbf{J}}_2^\# \hat{\mathbf{J}}_2)\mathbf{w} \end{aligned} \quad (2.16)$$

Eq 2.16 can be simplified to the following:

$$\dot{\mathbf{q}} = \mathbf{J}_1^\# \dot{\mathbf{s}}_1 + \hat{\mathbf{J}}_2^\# (\dot{\mathbf{s}}_2 - \mathbf{J}_2\mathbf{J}_1^\# \dot{\mathbf{s}}_1) + (\mathbf{I}_n - \mathbf{J}_1^\# \mathbf{J}_1)(\mathbf{I}_n - \hat{\mathbf{J}}_2^\# \hat{\mathbf{J}}_2)\mathbf{w} \quad (2.17)$$

Eq 2.17 indicates the inverse kinematics for prioritized subtasks \mathbf{s}_1 and \mathbf{s}_2 .

2.3.2 The Damped Pseudo-inverse

Eq 2.17 used the pseudo-inverse of the Jacobian matrices. However, near a singular configuration of the robot for a task, some elements of its Jacobian matrix can be close to zero, which yields elements close to infinity in its pseudo-inverse and causes the problem of discontinuity. In order to address this problem, the damped pseudo-inverse is used. The core of the damped pseudo-inverse is to find the $\dot{\mathbf{q}}$ that minimizes the following:

$$E = \left\| \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} - \dot{\mathbf{X}} \right\|^2 + \rho^2 \|\dot{\mathbf{q}}\|^2 \quad (2.18)$$

where $\rho \ll 1$ is the damping factor. The solution to this problem is:

$$\begin{aligned} \dot{\mathbf{q}} &= \mathbf{J}_\rho^\# \dot{\mathbf{X}} \\ \mathbf{J}_\rho^\# &= \mathbf{J}^T (\mathbf{J}\mathbf{J}^T + \rho^2 \mathbf{I}_m)^{-1} \end{aligned} \quad (2.19)$$

$\mathbf{J}_\rho^\#$ is called the damped pseudo-inverse of \mathbf{J} . As $\rho \rightarrow 0$, $\mathbf{J}_\rho^\#$ is identical to \mathbf{J} .

Given the singular value decomposition of \mathbf{J} :

$$\mathbf{J} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (2.20)$$

the following is derived:

$$\begin{aligned} \mathbf{J}_\rho^\# &= \mathbf{J}^T (\mathbf{J}\mathbf{J}^T + \rho^2 \mathbf{I}_m)^{-1} = \mathbf{V}\mathbf{\Sigma}^T \mathbf{U}^T (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T + \rho^2 \mathbf{I}_m)^{-1} \\ &= \mathbf{V}\mathbf{\Sigma}^T \mathbf{U}^T [\mathbf{U} (\mathbf{\Sigma}\mathbf{\Sigma}^T + \rho^2 \mathbf{I}_m) \mathbf{U}^T]^{-1} \\ &= \mathbf{V}\mathbf{\Sigma}^T \mathbf{U}^T \mathbf{U} (\mathbf{\Sigma}\mathbf{\Sigma}^T + \rho^2 \mathbf{I}_m)^{-1} \mathbf{U}^T \\ &= \mathbf{V}\mathbf{\Sigma}^T (\mathbf{\Sigma}\mathbf{\Sigma}^T + \rho^2 \mathbf{I}_m)^{-1} \mathbf{U}^T \\ &= \mathbf{V}\mathbf{\Sigma}_\rho \mathbf{U}^T \end{aligned} \quad (2.21)$$

$$\mathbf{\Sigma}_\rho = \begin{bmatrix} \tilde{\sigma}_1 & 0 & \cdots & 0 \\ 0 & \tilde{\sigma}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \tilde{\sigma}_m \\ 0 & 0 & \cdots & 0 \end{bmatrix} \quad \text{where } \tilde{\sigma}_i = \frac{\sigma_i}{\sigma_i^2 + \rho^2} \quad (2.22)$$

Therefore, by applying the modification of Eq 2.22 during the singular value decomposition of \mathbf{J} , its damped pseudo-inverse is obtained.

2.3.3 Obstacle Avoidance by Potential Function

Based on Khatib's usage of the artificial potential function in the obstacle avoidance problem[14], Nakamura proposed an artificial potential functions specified for defining a task of

obstacle avoidance in his prioritized task framework[15]. The potential function is defined by:

$$P_O = k_O \sum_{i=1}^6 (C_O(\mathbf{p}_i) - 1)^{-1} \quad (2.23)$$

where P_O gives the potential of the planar robot due to the obstacle, \mathbf{p}_i s represent six critical points on the robot, including three mobile joints and three centroids of the three robot links, as shown in Figure 2.3.

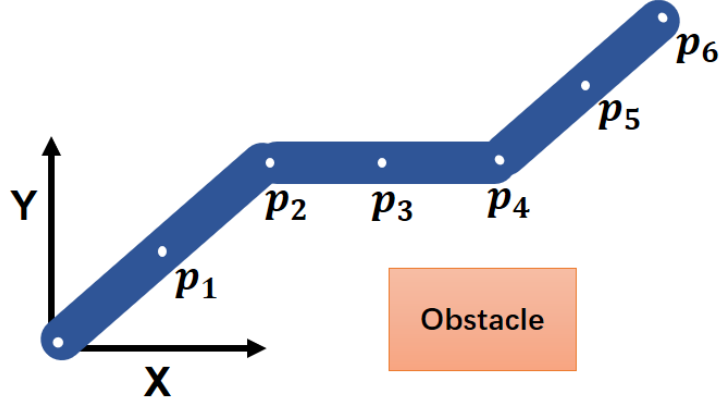


Figure 2.3: Critical points on the robot

When all of these six points are out of the dangerous range of the obstacle, it is assumed that the whole robot is safe from the obstacle. $C_O(\mathbf{p}_i)$ is used to approximate the contour of the obstacle. For a rectangular obstacle, $C_o(\mathbf{p}_i)$ is given as follows[14]:

$$C_O(\mathbf{p}_i) = \left(\frac{x_i - 60}{20}\right)^8 + \left(\frac{y_i - 15}{15}\right)^8 \quad (2.24)$$

The exponent of the two addend terms can be set accordingly, the higher it is, the more accurate the approximation of the contour of the rectangular obstacle is.

2.3.4 Obstacle Avoidance as Prioritized Task

Now consider a task of reaching the goal while the whole robot is avoiding an obstacle. The task is divided into a subtask of reaching the goal with first priority order and a subtask of avoiding obstacle with second priority order. Two subtasks can be defined as follows:

$$\begin{aligned} s_1 &= \|\mathbf{X}_{desired} - \mathbf{f}(\mathbf{q})\|^2 \\ s_2 &= P_O \end{aligned} \quad (2.25)$$

where $\mathbf{X}_{desired} \in R^3$ is the goal for the end-effector, $\mathbf{f}(\mathbf{q})$ is the FK of the end-effector, and P_O is the scalar potential of the robot due to the obstacle. The Jacobian matrices $\mathbf{J}_1 \in R^3$ and $\mathbf{J}_2 \in R^3$ for subtasks s_1 and s_2 can be analytically derived. And the solution for realizing the task can be obtained by applying Eq 2.17.

2.4 Offline Motion Planning

As opposed to the online instantaneous control, which calculates a control policy for the current time instant and executes it immediately, offline motion planners calculate a trajectory from the current time instant to a certain future time beforehand and drive the robot to follow the calculated trajectory. Basically, motion planning is to search for a continuous trajectory from an initial state \mathbf{q}_{init} to a goal state \mathbf{q}_{goal} or region \mathcal{Q}_{goal} in a configuration space \mathcal{C} . Each $\mathbf{q} \in \mathcal{C}$ specifies a state for the robot. And for scenarios with obstacles, an obstacle region $\mathcal{C}_{obs} \subset \mathcal{C}$ should be avoided. However, in most problems, it is difficult or even impossible to have an explicit representation of \mathcal{C}_{obs} . One can only use a collision detection method to check whether a configuration \mathbf{q} is in \mathcal{C}_{obs} . Motion planners are supposed to search for a trajectory that entirely lies in \mathcal{C}_{free} , the complement of \mathcal{C}_{obs} . Generally speaking, offline motion planners can be divided into two types: sampling-based planners and optimization-based planners. Sampling-based planners are widely used as they are easier to implement and can spare high computation costs in high-dimensional spaces that have complex constraints and thus are adaptable in various applications. The key idea behind sampling-based planners is to randomly sample valid configurations between initial and goal states and chain them in \mathcal{C}_{free} to form a feasible trajectory. On the other hand, optimization-based planners find optimal trajectories via solving a constrained optimization problem. They refrain from rough motions at the cost of more computation.

2.4.1 Sampling-based motion planners

Rapidly-exploring Random Trees (RRT) method[16] and Probabilistic RoadMaps (PRM) method[17] are the two most influential sampling-based planners. Many other planners are their extensions and improvements. These two methods are different from the way how they construct the graph connecting the configuration points. The roadmap generated from PRM covers the whole free configuration space, this roadmap can be reused for other motion planning problems in the same configuration space, this is referred to as a multi-query planning method, while RRT constructs a new graph for every motion planning problem and is called a single-query planning method.

2.4.2 Optimization-based Motion Planners

Sampling-based planners can be surprisingly good at finding a feasible path in a complicated environment with densely positioned obstacles. In this case, finding a path is much more important than focusing on the quality of the path. However, in many scenarios, where the environment has relatively simple structure and obstacles are sparsely located, heuristics used by RRT and PRM is then unnecessary and generated paths are unreasonable although they reach the goal position. RRT* and PRM* realize some optimization and smoothing operations on the trajectory, but these operations are only confined to choosing connections between fixed sampled vertices. Therefore, optimization-based planners are needed, either as a post-processing for the generated trajectory from sampling-based planners or as a standalone motion planner.

2.4.2.1 CHOMP

CHOMP is a trajectory optimization method based on covariant gradient descent. Two goals are realized by CHOMP[18]: smoothness and obstacle avoidance of the trajectory between start and goal configurations $q_{init}, q_{goal} \in R^m$. Here, only discrete trajectory is considered and a trajectory between q_{init} and q_{goal} is then defined by q_1, \dots, q_n . Velocities and accelerations at waypoints can be approximated by finite difference. The problem is modeled as a problem of minimizing the cost of a trajectory. This cost describes how near the robot is to the obstacles plus the norm of the dynamical quantities of the robot which determines the smoothness of the trajectory. The cost is then written as:

$$c(\xi) = c_{prior}(\xi) + c_{obs}(\xi) \quad (2.26)$$

where ξ describes a trajectory, c_{obs} indicates the cost of being close to obstacles and c_{prior} is a sum of squared derivatives. calculation of c_{prior} via finite finite difference is as follows:

$$c_{prior}(\xi) = \frac{1}{2} \sum_{d=1}^D \omega_d \|K_d \xi + e_d\|^2 \quad (2.27)$$

where $K_d \in R^{(n+1) \cdot m \times n \cdot m}$ s are finite difference matrices with $d = 1 \dots D$ and $e_d \in R^{(n+1) \cdot m}$ s are constant vectors that are used to approximate derivatives at the starting and ending configurations. Eq. 2.27 can be rewritten to quadratic form:

$$\begin{aligned} c_{prior}(\xi) &= \frac{1}{2} \sum_{d=1}^D \omega_d (K_d \xi + e_d)^T (K_d \xi + e_d) \\ &= \frac{1}{2} \sum_{d=1}^D \omega_d [\xi^T K_d^T K_d \xi + 2e_d^T K_d \xi + e_d^T e_d] \\ &= \frac{1}{2} \xi^T \left[\omega_d \sum_{d=1}^D K_d^T K_d \right] \xi + \left[\omega_d \sum_{d=1}^D e_d^T K_d \right] \xi + \frac{1}{2} \omega_d \sum_{d=1}^D e_d^T e_d \\ &= \frac{1}{2} \xi^T A \xi + b^T \xi + c \end{aligned} \quad (2.28)$$

The prior cost is then determined by constant matrices $A = \omega_d \sum_{d=1}^D K_d^T K_d$, $b = \omega_d \sum_{d=1}^D e_d^T K_d$ and $c = \frac{1}{2} \omega_d \sum_{d=1}^D e_d^T e_d$.

At k -th iteration, the value of the objective function $c(\xi)$ can be approximated in the small region around current trajectory point ξ_k by first-order Taylor expansion:

$$c(\xi) \approx c(\xi_k) + \nabla c(\xi_k)^T (\xi - \xi_k) \quad (2.29)$$

A minimization procedure is then conducted on Eq. 2.29 to find the optimal ξ at current iteration, with a regularization on ξ to retain the smoothness of the trajectory. Then the trajectory point at next iteration ξ_{k+1} is then given by:

$$\xi_{k+1} = \underset{\xi}{\operatorname{argmin}} \left\{ c(\xi_k) + \nabla c(\xi_k)^T (\xi - \xi_k) + \frac{\lambda}{2} \|\xi - \xi_k\|_M^2 \right\} \quad (2.30)$$

According to Ratliff[18], the regularization term is a norm taken with respect to the Riemannian metric M and this metric M is set to the matrix A in Eq. 2.28. The reason for this operation is that, under the metric A , the regularization term can be large if the trajectory at the current iteration is not smooth, which leads to a minimization mainly on this term during the optimization. In this way, the trajectory will keep as smooth as possible after each optimization. Solving Eq. 2.30 yields then the following update rule for ξ .

$$\xi_{k+1} = \xi_k - \frac{1}{\lambda} A^{-1} \nabla c(\xi_k) \quad (2.31)$$

Without loss of generality, assuming c_{obs} is convex and considering in a fairly small region around a local optimum, covariant gradient descent based on update rule 2.31 is said to be $O(n)$ times faster to converge as opposed to standard gradient descent[18], where n is the number of the waypoints in a trajectory. Besides, covariant gradient descent spares the calculation of the Hessian matrix, which is a large computation overhead but necessary for traditional gradient descent methods.

Definition of obstacle cost c_{obs} can be an arbitrary potential function, as long as it achieves reasonable obstacle avoidance during the minimization of its value. It should be close to zero when the robot is away from obstacles but increases rapidly when the robot is near the obstacles. For example, a smooth cost function can be defined as follows:

$$c(d) = \begin{cases} e^{-d}, & \text{if } x \geq 0 \\ -d + 1, & \text{otherwise} \end{cases} \quad (2.32)$$

The deficiency of CHOMP is that it needs to solve a inverse kinematics from a cartesian goal first, which would slow down the procedure. Besides, same as Nakamura's instantaneous control, there is still a risk of collision. Because obstacle avoidance is only achieved by minimizing obstacle cost, when the cost is not optimized down to a safe level, requirement for avoiding obstacle is broken.

2.4.2.2 STOMP

Optimization-based planners like CHOMP are prone to getting stuck in local minima and STOMP[19] overcomes this disadvantage. STOMP updates the candidate trajectory to a lower cost based on the combination of multiple generated noisy trajectories between given start and goal configurations. This stochastic nature of the method helps jump out of local minima. Another advantage for STOMP is that it does not need a gradient of the cost function to perform updates, while for CHOMP, smoothness and differentiability are necessary for its cost function to yield gradient information. Furthermore, the Hessian of the defined cost function needs to be positive definite for CHOMP to guarantee convergence and STOMP does not have this limitation.

Given $\xi_0 \in R^{N-m}$, an initial guess on the trajectory with N waypoints between $q_{init}, q_{goal} \in R^m$, STOMP formulates the following optimization problem:

$$\min_{\hat{\xi}} \mathbf{E} \left[\sum_{i=1}^N c(\hat{\xi}_i) + \frac{1}{2} \hat{\xi}^T R \hat{\xi} \right] \quad (2.33)$$

where $\hat{\xi} = \mathcal{N}(\xi, \Sigma)$ denotes a normal distribution with mean ξ and covariance matrix Σ , under which current stochastic trajectory candidates are sampled. Cost function $c(\hat{\xi}_i)$ can be defined to fulfill certain task. Matrix R is chosen, so that $\hat{\xi}^T R \hat{\xi}$ represents the sum of the squared dynamical quantities of the robot.

Taking the derivative of Eq. 2.33 and setting it to zero results in:

$$\begin{aligned} \nabla_{\hat{\xi}} \left(\mathbf{E} \left[\sum_{i=1}^N c(\hat{\xi}_i) + \frac{1}{2} \hat{\xi}^T R \hat{\xi} \right] \right) &\doteq 0, \\ \mathbf{E}(\hat{\xi}) &= -R^{-1} \mathbf{E} \left(\nabla_{\hat{\xi}} \left[\sum_{i=1}^N q(\hat{\xi}_i) \right] \right) = -R^{-1} \delta \hat{\xi}_G \\ \delta \hat{\xi}_G &= \mathbf{E} \left(\nabla_{\hat{\xi}} \left[\sum_{i=1}^N q(\hat{\xi}_i) \right] \right) \end{aligned} \quad (2.34)$$

where $\delta \hat{\xi}_G$ denotes the estimated gradient, and according to [19], this term can be formulated as follows:

$$\delta \hat{\xi}_G = \int \exp \left(-\frac{1}{\lambda} S(\xi) \right) \delta \xi d(\delta \xi) \quad (2.35)$$

This equation is the expectation of $\delta \xi$ under the probability distribution of $\exp(-\frac{1}{\lambda} S(\xi))$ where $S(\xi) = \sum_{i=1}^N c(\xi_i)$ indicates the accumulated cost along the trajectory ξ . For discrete case where K noisy trajectories are sampled and their costs are calculated, $\delta \hat{\xi}_G$ is then given by the following equation:

$$[\delta \hat{\xi}_G]_i = \sum_{k=1}^K \frac{\exp(-\frac{1}{\lambda} S(\hat{\xi}_{k,i}))}{\sum_{l=1}^K \exp(-\frac{1}{\lambda} S(\hat{\xi}_{l,i}))} \delta \xi_{k,i} \quad (2.36)$$

And STOMP update rule for current trajectory is $\xi^+ = \xi + M \delta \hat{\xi}$, the multiplication with $M = R^{-1}$ realizes the smoothness of the trajectory update. Updated trajectory is a convex combination of the sampled noisy trajectories from the last iteration.

2.5 Introduction to MPC

The fundamental concept of MPC is to use a dynamic model of the system to predict the future states of the system, and optimize one or multiple future state dependent metrics to produce a sequence of optimal control moves. However, only the control move at the first time step is executed, control moves after that are discarded. At coming to the next time step, the system state may be different from what MPC has predicted due to unmeasured disturbance. It is not a good choice to stick to executing the planned control move at the second time step from the solution sequence generated by the previous MPC, a new MPC is formulated according to the current system state and solved to generate new control moves. Figure 2.4 illustrates the concept of MPC. It is assumed that system states can be exactly measured for the robot manipulator system considered in this thesis. System states are joint angles and they can be precisely captured.

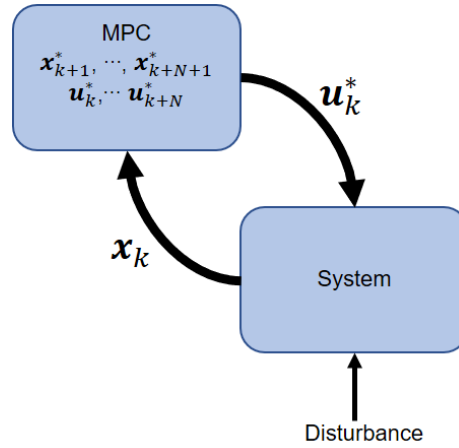


Figure 2.4: Concept of MPC.

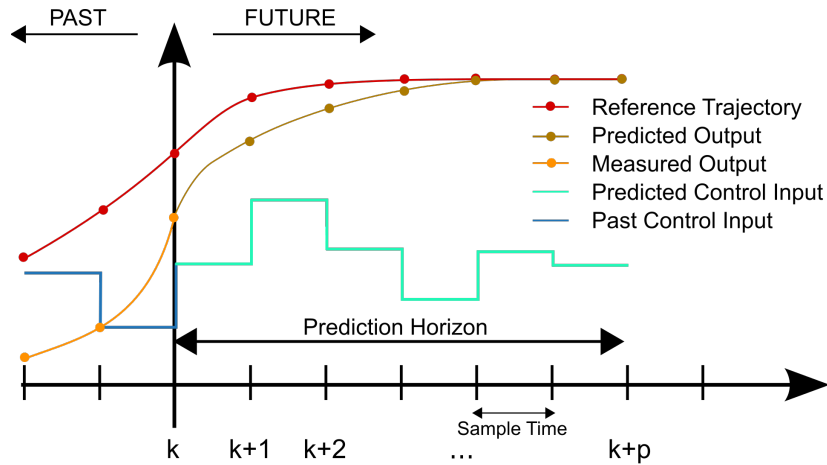


Figure 2.5: Prediction horizon in MPC [20].

2.5.1 Models

The model of the system is important for the prediction of MPC Figure 2.5. General continuous time variant systems can be described by:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k), t), \\ \mathbf{y}(k) &= \mathbf{h}(\mathbf{x}(k), t). \end{aligned} \quad (2.37)$$

where $\mathbf{x} \in R^n$ denotes the system states, $\mathbf{u} \in R^m$ the system inputs and $\mathbf{y} \in R^q$ the system outputs. Although natural systems are all continuous, discretized systems can well describe the original system as long as the sampling frequency is high enough. Discretization leads to a reduction in computation overhead and acceleration of the process. But it is worth mentioning that, there are novel optimal control schemes designed for continuous systems. For example, Nakamura and Hanafusa proposed an optimal redundancy control for a robot manipulator [13], by applying Pontryagin's Maximum Principle. The optimization problem is solved to generate an optimal continuous control function $\mathbf{u}(t)$ for the continuous system.

This dissertation focuses on discretized systems. Furthermore, linear time invariant (LTI) systems are used for conciser explanation of the MPC concept in this section.

Until now, we have discussed the basic fundamentals of robot motion planning and model predictive control. In Chapter 4, we will expand on this topic and relate it to non-linear model predictive control. Now we will move on to understand the basics of intention recognition in teleoperation using Markov Chains [21].

2.6 Hidden Semi-Markov Model

Hidden Markov Models (HMM) contain the spatial and temporal information by augmenting a Gaussian Mixture Model (GMM) with latent states that sequentially evolve over time in the demonstrations [22]. As an extension to the Hidden Markov Model (HMM), a Hidden Semi-Markov Model (HSMM) is traditionally defined by allowing the underlying process to be a semi-Markov chain with a variable duration/elapsed time for each state [23], i.e., the duration time d of each given state is also explicitly defined in HSMM in addition to the defined parameters in HMM. The state duration denotes a random integer variable that assumes a value in the set $\{1, 2, \dots, D\}$. The value indicates the number of observations produced in the given state i before the transition to the next state $i + 1$ [22].

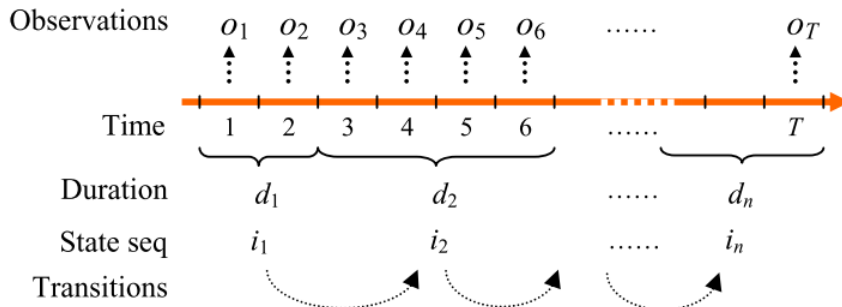


Figure 2.6: General HSMM [23] ©2010 Elsevier.

A general HSMM is illustrated in Figure 2.6, i_0 and d_0 are the initial state and duration, according to the transition probability a_{i_0, i_1} , the first state i_1 with duration d_1 are selected. i_1 lasts for $d_1 = 2$ time units in this instance and produces two observations o_1 and o_2 with the emission probability Π_1 . The model continues to develop with this rule.

In this dissertation, we use the HSMMs to segment the motion of the desired task, i.e., the letter trajectories are clustered according to the different datapoints in different states. Let $\{\xi_t\}_{t=1}^T$ denote the sequence of observations with datapoints $\xi_t \in \mathbb{R}^N$ collected while demonstrating the letter drawing task. The observation sequence is associated with a hidden state sequence $\{z_t\}_{t=1}^T$ with $z_t \in \{1, \dots, K\}$ belonging to a set of K cluster indices. The HSMM model is parameterized by $\theta = \{\Pi_i, \{a_{i,j}\}_{j=1}^K, \mu_i, \Sigma_i, \mu_i^S, \Sigma_i^S\}_{i=1}^K$. The transition matrix $\mathbf{a} \in \mathbb{R}^{K \times K}$ with $a_{i,j} \triangleq P(z_t = j | z_{t-1} = i)$ denotes the transition from one state i at time $t - 1$ to another state j at time t . Π_i is the initial state probability, the output distribution of state i is described by a multivariate Gaussian with parameters $\{\mu_i, \Sigma_i\}$, and the parameters

$\{\mu_i^S, \Sigma_i^S\}$ represent the mean and the standard deviation of staying s consecutive steps in state i estimated by a Gaussian $\mathcal{N}(s|\mu_i^S, \Sigma_i^S)$ [24].

2.6.1 Discussion about other State-of-the-art Approaches

As described above, we carry out the motion segmentation of desired letter trajectories using the HSMM and predict the intention of the human teleoperator in the letter drawing task by calculating the highest probabilities among 26 letters. However, there are also other state-of-art approaches to reach the same goal. For instance, Gaussian Mixture Model (GMM) and Hidden Markov Model (HMM) can be applied to segment the motions in the encoding step, k-nearest neighbors (KNN) algorithm, deep neural networks, and other machine learning methods can be also used to recognize the intention of the human teleoperator in recent years. A discussion about these state-of-art approaches is introduced as follows.

A GMM is a probabilistic clustering model to encode the local structure in the data for classification or regression and represented as a weighted sum of Gaussian component densities [25]. Given a set of T observations $\{\xi_t\}_{t=1}^T$ with $\xi_t \in \mathbb{R}^D$, the probability density function \mathcal{P} with K mixture components can be computed as [22]:

$$\mathcal{P}(\xi_t|\theta) = \sum_{i=1}^K \pi_i \mathcal{N}(\xi_t|\mu_i, \Sigma_i), \quad (2.38)$$

where $\mathcal{N}(\mu_i, \Sigma_i)$ denotes the multivariate Gaussian distribution with parameters $\theta = \{\pi_i, \mu_i, \Sigma_i\}_{i=1}^K$ which indicates a set of prior π_i , mean μ_i , and covariance matrix Σ_i and are estimated in the density function. For the case of motion segmentation, we focus mostly on clustering the given data to encode their local structure based on the variance in the demonstration, and the overall behavior of the given data interested us not. The log-likelihood function of the GMM is expressed as:

$$\mathcal{L}(\theta|\xi) = \sum_{t=1}^T \log\left(\sum_{i=1}^K \pi_i \mathcal{N}(\xi_t|\mu_i, \Sigma_i)\right). \quad (2.39)$$

For the initial set of parameters θ^{old} , the auxiliary function of GMM takes form:

$$\begin{aligned} \mathcal{Q}(\theta, \theta^{old}) &= \mathbb{E}\left\{\sum_{t=1}^T \log \mathcal{P}(\xi_t, z_t|\theta) | \xi_t, \theta^{old}\right\} \\ &\approx \frac{1}{2} \sum_{t=1}^T \sum_{i=1}^K h_{t,i}^{\theta^{old}} (\log \pi_i^2 - \log |\Sigma_i| - (\xi_t - \mu_i)^\top \Sigma_i^{-1} (\xi_t - \mu_i) - D \log(2\pi)), \end{aligned} \quad (2.40)$$

where $\{z_t\}_{t=1}^T$ denotes the unobserved labels that are assumed to be independent realizations of a random variable $z_t \in \{1, \dots, K\}$. $h_{t,i}^{\theta^{old}} = p(z_t = i | \xi_t, \theta^{old})$ represents the probability that the data point ξ_t belongs to i -th Gaussian component. In this case, the model can be trained to encode the desired motions using EM algorithms, in which we set the derivatives of the auxiliary function $\mathcal{Q}(\theta, \theta^{old})$ with respect to the model parameters equal to zero. The

E-step and M-step proceed iteratively until the likelihood function in Eq. 2.39 converges to a local optimum:

E-step:

$$h_{t,j} = \frac{\pi_j \mathcal{N}(\xi_t | \mu_j, \Sigma_j)}{\sum_{k=1}^K \pi_k \mathcal{N}(\xi_t | \mu_k, \Sigma_k)}. \quad (2.41)$$

M-step:

$$\pi_i \leftarrow \frac{\sum_{t=1}^T h_{t,i}}{T}, \quad (2.42)$$

$$\mu_i \leftarrow \frac{\sum_{t=1}^T h_{t,i} \xi_t}{\sum_{t=1}^T h_{t,i}}, \quad (2.43)$$

$$\Sigma_i \leftarrow \frac{\sum_{t=1}^T h_{t,i} (\xi_t - \mu_i)(\xi_t - \mu_i)^T}{\sum_{t=1}^T h_{t,i}}, \quad (2.44)$$

However, GMM can only encode the structure of motion and does not model the transition between the different states (see Fig. 2.7). Hence, the performance of GMM is worse than the one of HMM or HSMM. A graphical representation of the difference of encoding among GMM, HMM, and HSMM is illustrated in Fig. 2.7, as an extension of the HMM, HSMM models also the state duration probabilities as Gaussian distribution when the transition probabilities among states are kept. Therefore, HSMM performs better than the HMM due to the better models movements with longer state duration time. A comparison for use in robotic applications can be found in [26].

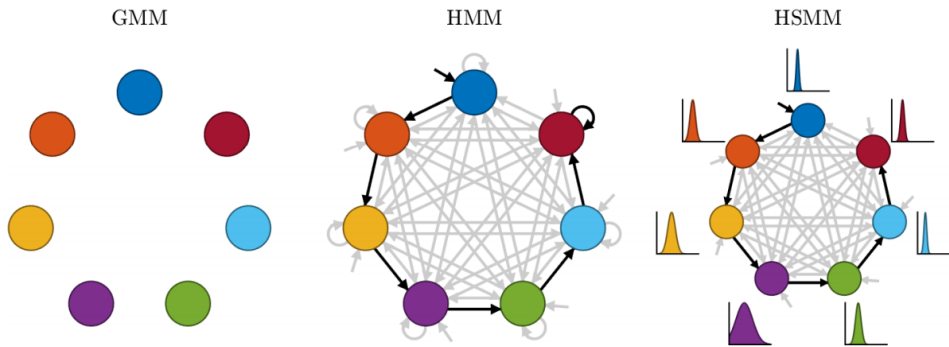


Figure 2.7: Graphical representation of a GMM, HMM, and HSMM with 7 components [22] ©2018 EPFL.

2.6.2 Linear Quadratic Tracker

In this section, we synthesize the motion of the recognized letter using a linear quadratic tracker (LQT), which is a commonly used tool from control theory to derive an optimal control policy for the robot. The step-wise desired sequences of poses $\{\mathcal{N}(\hat{\mu}_t, \hat{\Sigma}_t)\}_{t=1}^{T_p}$ are tracked

by optimizing a scalar cost function with a LQT [22]. We acquire the optimal control policy \mathbf{u}_t at each time step by minimizing the cost function over the finite time horizon T_p :

$$c_t(\boldsymbol{\xi}_t, \mathbf{u}_t) = \sum_{t=1}^{T_p} (\boldsymbol{\xi}_t - \hat{\boldsymbol{\mu}}_t)^\top \mathbf{Q}_t (\boldsymbol{\xi}_t - \hat{\boldsymbol{\mu}}_t) + \mathbf{u}_t^\top \mathbf{R}_t \mathbf{u}_t, \quad (2.45)$$

s.t. $\boldsymbol{\xi}_{t+1} = \mathbf{A}_d \boldsymbol{\xi}_t + \mathbf{B}_d \mathbf{u}_t,$

We use a linear time-invariant double integrator system to describe the discrete-time dynamical system, which is specified by \mathbf{A}_d and \mathbf{B}_d as follows [27]:

$$\begin{bmatrix} \boldsymbol{\xi}_{t+1} \\ \mathbf{x}_{t+1} \\ \mathbf{x}_{t+2} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_d \\ \mathbf{I} & \Delta t \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_t \\ \mathbf{x}_t \\ \mathbf{x}_{t+1} \end{bmatrix} + \begin{bmatrix} \mathbf{B}_d \\ \mathbf{I} \frac{1}{2} \Delta t^2 \\ \mathbf{I} \Delta t \end{bmatrix} \mathbf{u}_t. \quad (2.46)$$

By minimizing the cost function in Eq. 2.45 and setting $\mathbf{Q}_t = \hat{\boldsymbol{\Sigma}}_t^{-1} \geq 0$, $\mathbf{R}_t > 0$, the optimal control policy \mathbf{u}_t^* subject to the linear dynamics in discrete time is obtained as [27]:

$$\begin{aligned} \mathbf{u}_t^* &= -(\mathbf{R} + \mathbf{B}_d^\top \mathbf{P}_t \mathbf{B}_d)^{-1} \mathbf{B}_d^\top \mathbf{P}_t \mathbf{A}_d (\boldsymbol{\xi}_t - \hat{\boldsymbol{\mu}}_t) - (\mathbf{R} + \mathbf{B}_d^\top \mathbf{P}_t \mathbf{B}_d)^{-1} \mathbf{B}_d^\top (\mathbf{P}_t (\mathbf{A}_d \hat{\boldsymbol{\mu}}_t - \hat{\boldsymbol{\mu}}_t) + \mathbf{d}_t) \\ &= \mathbf{K}_t^{\mathcal{P}} (\hat{\boldsymbol{\mu}}_t^x - \mathbf{x}_t) + \mathbf{K}_t^{\mathcal{Y}} (\hat{\boldsymbol{\mu}}_t^{\dot{x}} - \dot{\mathbf{x}}_t) - (\mathbf{R} + \mathbf{B}_d^\top \mathbf{P}_t \mathbf{B}_d)^{-1} \mathbf{B}_d^\top (\mathbf{P}_t (\mathbf{A}_d \hat{\boldsymbol{\mu}}_t - \hat{\boldsymbol{\mu}}_t) + \mathbf{d}_t), \end{aligned} \quad (2.47)$$

where \mathbf{x} , $\dot{\mathbf{x}}$ indicate the position and velocity of the double integrator system, $\hat{\boldsymbol{\mu}}_t^x$, $\hat{\boldsymbol{\mu}}_t^{\dot{x}}$ represent the desired position and velocity to follow, and $[\mathbf{K}_t^{\mathcal{P}}, \mathbf{K}_t^{\mathcal{Y}}] = -(\mathbf{R} + \mathbf{B}_d^\top \mathbf{P}_t \mathbf{B}_d)^{-1} \mathbf{B}_d^\top \mathbf{P}_t \mathbf{A}_d$ denote the full stiffness and damping matrices for the feedback term. $\mathbf{u}_t^{FF} = (\mathbf{R} + \mathbf{B}_d^\top \mathbf{P}_t \mathbf{B}_d)^{-1} \mathbf{B}_d^\top (\mathbf{P}_t (\mathbf{A}_d \hat{\boldsymbol{\mu}}_t - \hat{\boldsymbol{\mu}}_t) + \mathbf{d}_t)$ is the feedforward term. By solving the Riccati differential equation and linear differential equation backwards in discrete time, we acquire \mathbf{P}_t and \mathbf{d}_t with terminal conditions respectively:

$$\mathbf{P}_{t-1} = \mathbf{Q}_t - \mathbf{A}_d^\top (\mathbf{P}_t \mathbf{B}_d (\mathbf{R} + \mathbf{B}_d^\top \mathbf{P}_t \mathbf{B}_d)^{-1} \mathbf{B}_d^\top \mathbf{P}_t - \mathbf{P}_t) \mathbf{A}_d, \quad \mathbf{P}_{T_p} = \mathbf{Q}_{T_p}, \quad (2.48)$$

$$\mathbf{d}_{t-1} = (\mathbf{A}_d^\top - \mathbf{A}_d^\top \mathbf{P}_t \mathbf{B}_d (\mathbf{R} + \mathbf{B}_d^\top \mathbf{P}_t \mathbf{B}_d)^{-1} \mathbf{B}_d^\top) (\mathbf{P}_t (\mathbf{A}_d \hat{\boldsymbol{\mu}}_t - \hat{\boldsymbol{\mu}}_{t+1}) + \mathbf{d}_t), \quad \mathbf{d}_{T_p} = \mathbf{0}. \quad (2.49)$$

We described the formulations with finite horizon case, and for the infinite horizon case with $T_p \rightarrow \infty$ and the desired pose $\hat{\boldsymbol{\mu}}_t = \hat{\boldsymbol{\mu}}_{t_0}$, the feedforward term \mathbf{u}_t^{FF} is set to zero and $\mathbf{P}_{t-1} = \mathbf{P}_t = \mathbf{P}$ is obtained by calculating the steady-state solution of eigenvalue decomposition of the discrete algebraic Riccati equation (DARE) in Eq. 2.48. Then we define the symplectic matrix [27]:

$$\mathbf{H}_b = \begin{bmatrix} \mathbf{A}_d + \mathbf{B}_d \mathbf{R}^{-1} \mathbf{B}_d^\top (\mathbf{A}_d^{-1})^\top \mathbf{Q} & \mathbf{B}_d \mathbf{R}^{-1} \mathbf{B}_d^\top (\mathbf{A}_d^{-1})^\top \\ -(\mathbf{A}_d^{-1})^\top \mathbf{Q} & (\mathbf{A}_d^{-1})^\top \end{bmatrix}. \quad (2.50)$$

Let $\begin{bmatrix} \mathbf{V}_1^\top & \mathbf{V}_{21}^\top \end{bmatrix}$ be the corresponding subspace of \mathbf{H}_b , thus the solution of DARE indicates $\mathbf{P} = \mathbf{V}_{21}\mathbf{V}_1^{-1}$ and the optimal control policy is expressed as:

$$\mathbf{u}_t^* = -(\mathbf{R} + \mathbf{B}_d^\top \mathbf{P} \mathbf{B}_d)^{-1} \mathbf{B}_d^\top \mathbf{P} \mathbf{A}_d (\boldsymbol{\xi}_t - \hat{\boldsymbol{\mu}}_t). \quad (2.51)$$

After computing the optimal control policy, we can reproduce the desired trajectory according to the recognized letter. In this dissertation, the discrete-time LQT based on [27] is applied after encoding with HSMM. Fig. 2.8 shows the results of applying discrete LQT from an HSMM encoding of the demonstrations of the example letters A and C. The blue lines shown in Fig. 2.8 represent respectively the reproduced positions of the trajectory of the letter A and the reproduced velocity of the trajectory of the letter A. The lines with other colors in the figures are the trajectories of the letters of 10 observed demonstrations while encoding with the HSMM. We can clearly see the good performance of trajectory reproduction using discrete-time LQT.

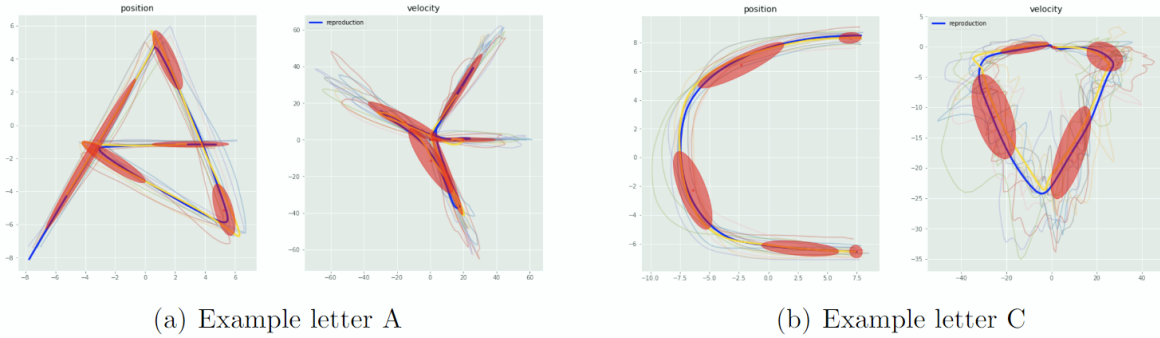


Figure 2.8: Trajectory reproduction of two example letters (from Chapter 5 experiments).

2.6.2.1 Continuous-time LQT

However, the continuous-time LQT can also be used to follow the desired pose $\mathcal{N}(\hat{\boldsymbol{\mu}}_t, \hat{\boldsymbol{\Sigma}}_t)$ at time t based on [28]. Same with the discrete-time LQT, a linear time-invariant double integrator system is considered to describe the dynamics of the linear system. In this case, the double integrator is defined as:

$$\begin{bmatrix} \dot{\boldsymbol{\xi}}_t \\ \ddot{\boldsymbol{\xi}}_t \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_t \\ \dot{\boldsymbol{\xi}}_t \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \mathbf{u}_t. \quad (2.52)$$

where datapoint $\boldsymbol{\xi}_t = \begin{bmatrix} \mathbf{x}_t^\top & \dot{\mathbf{x}}_t^\top \end{bmatrix}^\top$, $\hat{\boldsymbol{\mu}}_t = \begin{bmatrix} (\hat{\boldsymbol{\mu}}_t^x)^\top & (\hat{\boldsymbol{\mu}}_t^{\dot{x}})^\top \end{bmatrix}^\top$. Therefore, by minimizing the Hamilton-Jacobi-Bellman equation [22], the optimal control policy \mathbf{u}_t^* that minimizes the cost function in Eq. 2.45 can be computed as:

$$\begin{aligned} \mathbf{u}_t^* &= -\mathbf{R}_t^{-1} \mathbf{B}_d^\top \mathbf{P}_t (\boldsymbol{\xi}_t - \hat{\boldsymbol{\mu}}_t) + \mathbf{R}_t^{-1} \mathbf{B}_d^\top \mathbf{d}_t \\ &= \mathbf{K}_t^{\mathcal{P}} (\hat{\boldsymbol{\mu}}_t^x - \mathbf{x}_t) + \mathbf{K}_t^{\mathcal{V}} (\hat{\boldsymbol{\mu}}_t^x - \dot{\mathbf{x}}_t) + \mathbf{R}_t^{-1} \mathbf{B}_d^\top \mathbf{d}_t, \end{aligned} \quad (2.53)$$

where $[\mathbf{K}_t^{\mathcal{P}}, \mathbf{K}_t^{\mathcal{V}}] = \mathbf{R}_t^{-1} \mathbf{B}_d^\top \mathbf{P}_t$ denote the full stiffness and damping matrices, $\mathbf{u}_t^{FF} = \mathbf{R}_t^{-1} \mathbf{B}_d^\top \mathbf{d}_t$ indicates the feedforward term. By setting the terminal conditions $\mathbf{P}_{T_p} = \mathbf{0}$ and $\mathbf{d}_{T_p} = \mathbf{0}$, \mathbf{P}_t and \mathbf{d}_t can be solved respectively by:

$$-\dot{\mathbf{P}}_t = \mathbf{A}_d^\top \mathbf{P}_t + \mathbf{P}_t \mathbf{A}_d - \mathbf{P}_t \mathbf{B}_d \mathbf{R}_t^{-1} \mathbf{B}_d^\top \mathbf{P}_t + \mathbf{Q}_t, \quad (2.54)$$

$$-\dot{\mathbf{d}}_t = \mathbf{A}_d^\top \mathbf{d}_t - \mathbf{P}_t \mathbf{B}_d \mathbf{R}_t^{-1} \mathbf{B}_d^\top \mathbf{d}_t + \mathbf{P}_t \hat{\boldsymbol{\mu}}_t - \mathbf{P}_t \mathbf{A}_d \hat{\boldsymbol{\mu}}_t. \quad (2.55)$$

In the case of infinite horizon with $T_p \rightarrow \infty$ and $\mathbf{Q}_t = \mathbf{Q}$ in Eq. 2.45, the feedforward term \mathbf{u}_t^{FF} is set to zero. $\mathbf{P}_{t-1} = \mathbf{P}_t = \mathbf{P}$ can be calculated by minimizing the continuous algebraic Riccati equation (CARE):

$$\mathbf{A}_d^\top \mathbf{P} + \mathbf{P} \mathbf{A}_d - \mathbf{P} \mathbf{B}_d \mathbf{R}^{-1} \mathbf{B}_d^\top \mathbf{P} + \mathbf{Q} = \mathbf{0}. \quad (2.56)$$

In order to solve CARE, the Hamiltonian matrix is defined as:

$$\mathbf{H}_a = \begin{bmatrix} \mathbf{A}_d & -\mathbf{B}_d \mathbf{R}^{-1} \mathbf{B}_d^\top \\ -\mathbf{Q} & -\mathbf{A}_d^\top \end{bmatrix}. \quad (2.57)$$

The solution denotes $\mathbf{P} = \mathbf{V}_{21} \mathbf{V}_1^{-1}$ by solving the eigenvalue decomposition of the Hamiltonian matrix \mathbf{H}_a as follows:

$$\mathbf{H}_a = \mathbf{V} \begin{bmatrix} \lambda_1 & \mathbf{0} \\ \mathbf{0} & \lambda_2 \end{bmatrix} \mathbf{V}^\top, \quad \mathbf{V} = \begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_{12} \\ \mathbf{V}_{21} & \mathbf{V}_2 \end{bmatrix}. \quad (2.58)$$

The optimal control policy $\boldsymbol{\mu}_t^*$ can be obtained as:

$$\mathbf{u}_t^* = -\mathbf{R}^{-1} \mathbf{B}_d^\top \mathbf{P} (\boldsymbol{\xi}_t - \hat{\boldsymbol{\mu}}_t). \quad (2.59)$$

Both discrete and continuous-time LQT can be applied to reproduce the desired trajectory. However, numerically stable results are acquired for a wide range of weight values \mathbf{R} in the discrete-time LQT [22]. Thus, we used the discrete-time LQT in this dissertation.

2.7 Point Set Registration

In this section, we describe the basic fundamentals of the point set registration problem in order to give you a better understanding of the problem in chapter 5. We will then use this fundamental knowledge for skill refinement in manipulator teleoperation using shared autonomy.

The registration strategy we used in chapter 5 was applied to a letter dataset as an example. The skill refinement is converted into a point set registration problem, which is frequently encountered in numerous applications of computer vision, medical image analysis, etc. A letter trajectory in our defined task can be considered as a point set and the points in a point set represent features extracted from the corresponding letter trajectory. The objective of point set registration is to assign correspondences between two point sets and to recover the transformation that maps one point set to the other [5]. Therefore, the partial demonstrated trajectory and the reproduced trajectory can be treated as the two point sets, whose correspondences and the transformation need to be assigned in order to improve the sub-optimal performance due to the unmatched between the partial demonstration and the reproduction.

An example of a point set registration problem is shown in Fig. 2.9. The point set registration methods usually fall into two categories: rigid and non-rigid. The rigid registration of two point sets yields a rigid transformation that the distance between any two points in the two point sets is preserved. Thus, the typical rigid transformation comprises translation and rotation. In contrast, anisotropic scaling and skews are also allowed in non-rigid transformation which is usually associated with nonlinear transformation. The non-rigid registration problem is normally more difficult to solve because the true underlying model for non-rigid transformation is often unknown and hard to find. However, the non-rigid transformation still plays a very important role in many real-world tasks.

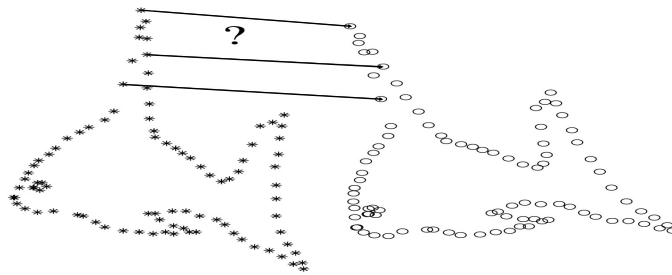


Figure 2.9: Non-rigid point set registration problem [5] ©2010 IEEE.

Solving the practical point set registration problem is challenging due to multiple factors, including the large dimensionality of point sets, high computational complexity, unpredictable noise, and outliers in the real world. Therefore, a reliable point set registration algorithm should be able to: (1) accurately model the transformation with tractable computational complexity; (2) handle possible high dimensional point sets; (3) become robust to noise, outliers, and even missing points because of the imperfect image acquisition and feature extraction [5]. Hence, the coherent point drift (CPD) algorithm is applied in our system due to its robust and accurate performance for both rigid and non-rigid transforms. In the following, we describe mainly the CPD algorithm and give then a discussion about other state-of-art methods for point set registration problem.

2.7.1 Coherent Point Drift Algorithm

Coherent point drift (CPD) is known as a robust probabilistic multidimensional point set registration algorithm for both rigid and non-rigid transforms [5]. Herein the alignment of two point sets is viewed as a probability density estimation problem. The one point set indicates the Gaussian Mixture Model (GMM) centroids and the second point set represents the data points that are fit by maximizing the likelihood. The Expectation-Maximization (EM) algorithm is applied for the optimization and the correspondence between the two point sets can be inferred by calculating the maximum of the posterior probability of the Gaussian mixture components. In order to preserve the topological structure of the point set, the GMM centroids are forced to move coherently as a group, which is also the core of this method. An example of affine point registration using CPD algorithm is illustrated in Fig. 2.10. As shown in Fig. 2.10, the two fish point sets are independent before using CPD algorithm. The result after using CPD algorithm is shown in Fig. 2.10, the blue fish point set is registered to the red one with a perfect correspondence.

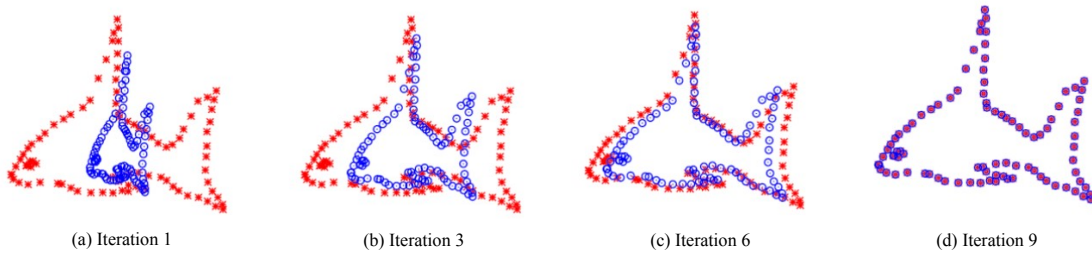


Figure 2.10: Affine registration using CPD [29] ©2010 IEEE.

According to [5], the CPD algorithm for affine and non-rigid point set registration case is derived as follows. Firstly, we give the following notations that are used during the derivation:

- The reference point set $\mathbf{X}_{N \times D} = (\mathbf{x}_1, \dots, \mathbf{x}_N)^\top$ is expressed as a $N \times D$ matrix, where D denotes the dimension of the point set,
- The template point set $\mathbf{Y}_{M \times D} = (\mathbf{y}_1, \dots, \mathbf{y}_M)^\top$ is expressed as a $M \times D$ matrix,
- $\mathcal{T}(\mathbf{Y}, \theta)$ indicates the transformation applied to \mathbf{Y} , where θ denotes a set of the transformation parameters,
- \mathbf{I} represents the identity matrix,
- $\text{col}(\mathbf{1})$ denotes the column vector of all ones,
- $\mathbf{d}(\mathbf{a})$ is the diagonal matrix formed from the vector \mathbf{a} .

Consider two point set \mathbf{X} and \mathbf{Y} , the objective is to align \mathbf{Y} with \mathbf{X} . The points in \mathbf{Y} are treated as the GMM centroids and the points in \mathbf{X} are viewed as the data points generated by the GMM. The GMM probability density function $p(\mathbf{x})$ is expressed as:

$$p(\mathbf{x}) = \sum_{m=1}^{M+1} P(m)p(\mathbf{x}|m), \quad (2.60)$$

where

$$p(\mathbf{x}|m) = \frac{1}{(2\pi\sigma^2)^{D/2}} \exp -\frac{\|\mathbf{x} - \mathbf{y}_m\|_2}{2\sigma^2}. \quad (2.61)$$

In order to account for noise and outliers that could appear in the point sets, an additional uniform distribution $p(\mathbf{x}|M+1) = \frac{1}{N}$ is also considered in the mixture model. The membership probabilities $P(m) = \frac{1}{M}$ and the isotropic covariances σ^2 are equal for all GMM components $m = 1, \dots, M$. We denote the weight of the uniform distribution as w , where $0 \leq w \leq 1$. Therefore, the mixture model can be expressed as:

$$p(\mathbf{x}) = w \frac{1}{N} + (1-w) \sum_{m=1}^M \frac{1}{M} p(\mathbf{x}|m). \quad (2.62)$$

The GMM centroid locations are reparameterized by a set of parameters θ that can be estimated by maximizing the likelihood. It can be done equivalently by minimizing the negative log-likelihood function:

$$E(\theta, \sigma^2) = - \sum_{n=1}^N \log \sum_{m=1}^{M+1} P(m)p(\mathbf{x}_n|m). \quad (2.63)$$

where we make the assumption that the data is independent and identically distributed. The correspondence probability between two points \mathbf{x}_n in point set \mathbf{X} and \mathbf{y}_m in point set \mathbf{Y} is defined as the posterior probability of the GMM centroid given the data point:

$$P(m|\mathbf{x}_n) = \frac{P(m)p(\mathbf{x}_n|m)}{p(\mathbf{x}_n)}. \quad (2.64)$$

To find the optimal θ and σ^2 , Expectation-Maximization (EM) algorithm is applied, which consists of two steps. The first step is the expectation or E-step, the values of the "old" parameter is guessed and a posteriori probability distributions $P^{old}(m|\mathbf{x}_n)$ of mixture components is calculated using the Bayes' theorem. In the second step, i.e., the maximization or M-step, the "new" parameter values are then found by minimizing the expectation of the complete negative log-likelihood function which is called also as cost function:

$$Q(\theta, \sigma^2) = - \sum_{n=1}^N \sum_{m=1}^{M+1} P^{old}(m|\mathbf{x}_n) \log(P^{new}(m)p^{new}(\mathbf{x}_n|m)). \quad (2.65)$$

We can see that the cost function in Eq. 2.65 is actually also an upper bound of the negative log-likelihood function in Eq. 5.6. Therefore, it's necessary to decrease the negative log-likelihood function E in Eq. 5.6 in order to minimize the cost function Q , unless it is already at a local minimum [5]. By alternating between the E- and M-step, the EM algorithm converges to the optimum. Ignoring the constants independent of θ and σ^2 , the cost function can then be expressed as:

$$Q(\theta, \sigma^2) = \frac{1}{2\sigma^2} \sum_{n=1}^N \sum_{m=1}^M P^{old}(m|\mathbf{x}_n) \|\mathbf{x}_n - \mathcal{T}(\mathbf{y}_m, \theta)\|^2 + \frac{N_P D}{2} \log \sigma^2, \quad (2.66)$$

where

$$N_P = \sum_{n=1}^N \sum_{m=1}^M P^{old}(m|\mathbf{x}_n) \leq N. \quad (2.67)$$

with $N_P = N$ only if $w = 0$. The posterior probabilities of GMM components $P^{old}(m|\mathbf{x}_n)$ can be computed with the help of the previous parameter values:

$$P^{old}(m|\mathbf{x}_n) = \frac{\exp\left(-\frac{1}{2} \left\| \frac{\mathbf{x}_n - \mathcal{T}(\mathbf{y}_m, \theta^{old})}{\sigma^{old}} \right\|^2\right)}{\sum_{k=1}^M \exp\left(-\frac{1}{2} \left\| \frac{\mathbf{x}_n - \mathcal{T}(\mathbf{y}_k, \theta^{old})}{\sigma^{old}} \right\|^2\right) + (2\pi\sigma^2)^{\frac{D}{2}} \frac{w}{1-w} \frac{M}{N}}, \quad (2.68)$$

Then we describe the rigid, affine, and non-rigid point set registration with different transformation \mathcal{T} separately.

2.8 Reinforcement Learning

The purpose of this section is to introduce the principles of Reinforcement Learning (RL), which will later be applied to a liquid pouring problem in Chapter 6.

RL as an Artificial Intelligence approach was possibly described by Waltz and Fu in 1965 in their paper "A heuristic approach to reinforcement learning control systems" [30] for the first time. However, Rich Sutton [31] is often credited with the popularity of RL through his exceptional work on Temporal Difference Learning [32] and Policy Gradient Methods [33], which today forms the fundamentals of the RL in the domain of Artificial Intelligence. He is for his contributions towards RL, often regarded as one of the founding fathers of the domain.

In a nutshell, RL is the domain of Machine Learning, where an intelligent agent acts in its environment, observing the states in order to maximize the notion of "cumulative reward" [31]. Reinforcement learning unlike Supervised Learning does not need labelled data to be present. Likewise, the sub-optimal actions need not be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory in the environment) and exploitation (of current knowledge the policy has). Particularly in the context of precision pouring, a RL-based scenario can be broken down to have the following components:

- **States:** This comprises of liquid features, measurements from the force/torque feedback and pouring container's translation and rotational positions, object's geometrical features and weight of liquid in both pouring and target container at a given time-step in the scene. The states form the basis of observation that an intelligent RL agent perceives in it's environment. Based on the observations made, an agent should be in a position to exploit it's policy to generate the suitable actions for the next time frame.

- **Actions:** This comprises of translation and rotational speed control actions that an agent perform in order to precisely pour the liquid contents into the target container. An action causes change in the states in the scene i.e. in terms of measurements, weight of liquid transferred, etc. Based on the actions that the agent takes, it receives a reward from the environment which can either be reinforcing in nature or negative to avoid the sequence of actions taken.
- **Rewards:** Each action is associated with a small negative reward to motivate the agent to complete it's task as quickly as possible. However, the overall task completion is a "Sparse Reward" problem, i.e. an agent can only get a final large positive reinforcing reward, if the sequence of actions till the task completion results in pouring deviation within some tolerable limit. If the pouring deviation is large, the agent can not get a positive reinforcing reward. The reward shaping hence, is one of the most important component of the RL-based applications.
- **Environment:** The RL agent works in an environment where it can observe the states, make actions and receive rewards from the environment for the actions, resulting in changing states. It is possible to have a real or virtual working environment. In this dissertation, the RL-based precision pouring problem is considered in a simulation-based environment where an agent comes up with a robust pouring policy by working with a multitude of liquids. Replicating the same behavior for training in the real world would be time and cost expensive. However, it is possible to transfer a simulation-driven policy in the real-world environment. In this dissertation, a simulation-to-real transfer of the pouring policy has been demonstrated.

2.8.1 Proximal Policy Optimization

The RL algorithm considered in this dissertation is based on Proximal Policy Optimization (PPO) [34]. For the pouring liquids problem, we will discuss in the chapter 6 that the PPO observed a better average pouring error when compared to the Soft-Actor-Critical (SAC) [35] based RL algorithm. In Table 2.1, all major algorithms for RL optimization are compared. For robot control training, the state space and action space must be continuous. In simulation, on-policy methods are more efficient as the state space is generated from scratch each time, allowing for a better exploration of the agent's behavior. PPO, TRPO and A3C are good in continuous actions and multi-processed problems. Training is faster in A3C but the convergence is better in PPO while TRPO struggles at some points.

The PPO is one of the policy gradient method-based approaches which alternates between sampling data through interaction with the environment and optimizing a "surrogate" objective function using stochastic gradient ascent. This approach sets PPO apart from the other policy gradient method approaches, where PPO is capable of multiple epochs for mini-batch updates. On the contrary, the traditional policy gradient methods are only capable of one gradient update per data sample. PPO is simpler to implement, exhibits better empirical sample complexity and is more general.

Some of the salient features of PPO are:

Table 2.1: Comparison of reinforcement learning algorithms.

Algorithm	Description	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Either	Discrete	Discrete	Sample-means
Q-learning	State-action-reward-state	Off-policy	Discrete	Discrete	Q-value
SARSA	State-action-reward-state-action	On-policy	Discrete	Discrete	Q-value
Q-learning - Lambda	State-action-reward-state with eligibility traces	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State-action-reward-state-action with eligibility traces	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	On-policy	Continuous	Continuous	Advantage
NAF	Q-Learning with Normalized Advantage Functions	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor-Critic	Off-policy	Continuous	Continuous	Advantage

1. PPO is On-policy algorithm, i.e. PPO-based policy evaluates and improves the same policy which is being used to select actions based on the inputs. PPO thus do not require an experience buffer which can be looked on to for observation-action pair as seen in the traditional algorithms like Soft Actor Critic [35], etc. This makes PPO more stable but data inefficient, as a large number of representative experience data is required in absence of an experience buffer to lookup to.
2. PPO-based policy is suited both for *Continuous domain* and *Discrete* actions.
3. PPO-based policy uses entropy regularization unlike traditional approaches involving addition of entropy for the objective maximization.

PPO ensures robustness of policy by optimizing a novel loss function called the *Clipped Surrogate Objective* [34]. This ensures that the sudden catastrophic drops in performance is avoided. The Clipped Surrogate Objective is described below:

$$L^{CLIP}(\theta) = \hat{t}[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad (2.69)$$

Here, expectation is calculated over a minimum of two terms, i.e., normal policy gradient objective and a clipped policy gradient objective. The key component comes from the second term where a normal policy gradient objective is truncated with a clipping operation between $1-\epsilon$ and $1+\epsilon$, epsilon being the hyperparameter. Epsilon influences how rapidly the policy can evolve during training. Epsilon also corresponds to the acceptable threshold of divergence between the old and new policies during gradient ascent update.

2.9 Liquid Simulation

In this section we explain more details about the liquid simulation background which later in chapter 6 will be utilized for the pouring task in the RRS-PL simulator. For many years,

computer graphics researchers have been studying the modeling and simulation of complicated fluid effects. The dynamics of complicated fluids, such as those with high viscosity or nonlinear strain-stress relationships, have attracted the interest of various researchers, as these fluids exhibit a variety of unique effects and behaviors.

2.9.1 Position Based Fluids

Enforcing incompressibility is critical for realism in fluid simulation, but it is also computationally expensive. Although related work has improved efficiency, real-time applications still require time-steps that are impractical. An iterative density solver is integrated into the Position Based Dynamics framework. By formulating and solving a set of positional constraints that enforce constant density, Position Based Dynamics allows similar incompressibility and convergence to modern smoothed particle hydrodynamic (SPH) solvers, but inherits the stability of the geometric, position-based dynamics method, allowing large time steps suitable for real-time applications. An artificial pressure term is used to improve particle distribution, create surface tension, and reduce traditional SPH's neighborhood requirements. Furthermore, vorticity confinement will be incorporated as a velocity postprocess to address the issue of energy loss.

Smoothed Particle Hydrodynamics (SPH) is a well-known particle-based method for fluid simulation [36]. It has a number of appealing features, including mass conservation, Lagrangian discretization (which is especially useful in games where the simulation domain isn't known ahead of time), and conceptual simplicity. However, due to the unstructured character of the model, SPH is susceptible to density fluctuations caused by neighborhood defects, and enforcing incompressibility is costly. Robustness is a fundamental concern in interactive environments: the simulation must smoothly tolerate degenerate conditions. Position Based Fluids method is chosen because of its unwaveringly steady time integration and robustness, which has made it popular among game and film developers. This solution allows users to exchange incompressibility for performance while remaining stable by addressing particle shortage at free surfaces and handling significant density errors. Figure 2.11 illustrates the underlying particle simulation of an example liquid rendering. Moreover, it illustrates particle clumping and artificial pressure.

2.9.2 Enforcing Incompressibility

PBD solves a system of non-linear constraints with one constraint per particle to enforce constant density. Each constraint is determined by the particle's position as well as the positions of its neighbors, which we refer to as $\mathbf{p}_1, \dots, \mathbf{p}_n$. The density limitation on the i th particle is defined using an equation of state according to [38]:

$$C_i(\mathbf{p}_1, \dots, \mathbf{p}_n) = \frac{\rho_i}{\rho_0} - 1 \quad (2.70)$$

where ρ_0 is the rest density and ρ_i is given by the standard SPH density estimator:

$$\rho_i = \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h) \quad (2.71)$$

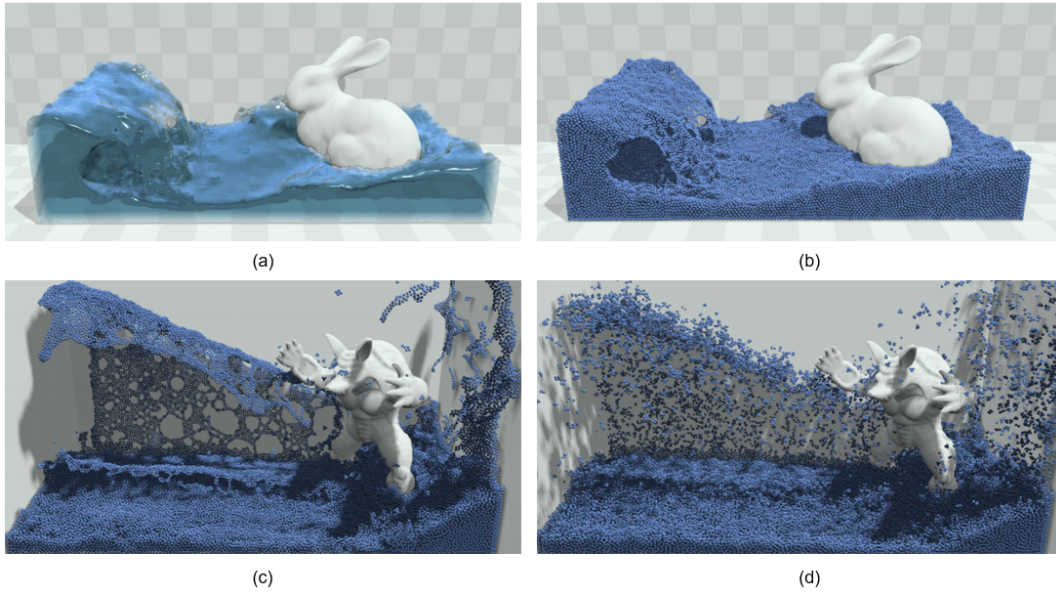


Figure 2.11: (a) Real-time rendered fluid ellipsoid splatting. (b) Underlying simulation particles. (c) Particle clumping due to neighbor deficiencies. (d) with artificial pressure term. [37] ©2013 ACM.

It has been assumed that all particles have the same mass and remove this term from subsequent equations. As in [39], the Poly6 kernel for density estimation and the Spiky kernel for gradient calculation has been implemented.

The SPH formula for the gradient of a function specified on the particles is given in [40]. The gradient of the constraint function (1) with regard to a particle k can be calculated as follows:

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_0} \sum_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) \quad (2.72)$$

Which has two different cases based on whether k is a neighboring particle or not:

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_0} \begin{cases} \sum_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) & \text{if } k = i \\ -\nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) & \text{if } k = j \end{cases} \quad (2.73)$$

Plugging this into the equation of a series of Newton steps along the constraint gradient and solving for λ gives

$$\lambda_i = -\frac{C_i(\mathbf{p}_1, \dots, \mathbf{p}_n)}{\sum_k |\nabla_{\mathbf{p}_k} C_i|^2} \quad (2.74)$$

which is the same for all particles in the constraint. The denominator in equation (9) creates instability when particles are close to splitting because the constraint function (1) is non-linear and has a vanishing gradient at the smoothing kernel border. This can be solved similarly to Predictive-corrective incompressible SPH (PCISPH) [41] by computing a cautious corrective scale in advance based on a reference particle configuration with a filled neighborhood.

Constraint force mixing (CFM) [Smith 2006] is another option for regularizing the constraint. CFM softens the constraint by reintroducing some of the constraint force into the constraint function; in the instance of PBD, this alters the equation of a series of Newton steps along the constraint gradient to:

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla C^T \nabla C \lambda + \varepsilon \lambda = 0 \quad (2.75)$$

Where ε is a user specified relaxation parameter that is constant over the simulation. The scaling factor is now:

$$\lambda_i = -\frac{C_i(\mathbf{p}_1, \dots, \mathbf{p}_n)}{\sum_k |\nabla_{\mathbf{p}_k} C_i|^2 + \varepsilon} \quad (2.76)$$

and the total position update \mathbf{p}_i including corrections from neighbor particles density constraint λ_j is:

$$\Delta\mathbf{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(\mathbf{p}_i - \mathbf{p}_j, h) \quad (2.77)$$

2.9.3 Tensile Instability

For fluid simulation methods that use SPH interpolation technology to calculate density, usually 30-40 neighbor particles are needed to make the density evaluation result tend to be static. In the case of insufficient neighbor particles, the density of the fluid calculated by formula (3.9) will be lower than the static density, which will cause the pressure to be negative, and the original pressure between the particles will become attractive, causing the particles to produce unrealistic conditions. Cohesion, this is the concrete manifestation of the Tensile Instability problem of SPH in fluid simulation, which leads to the result that the simulation of the fluid surface feels unreal.

[3] uses an artificial repulsive force calculation model. When the fluid particles are too close, the repulsive force will separate them, thereby avoiding particle aggregation. When the pressure of the fluid particles becomes negative, replacing the pressure with the repulsive force can effectively eliminate the Tensile Instability problem of the SPH method and prevent the unnatural attraction between particles caused by the negative pressure. PBF also adopted a similar method, adding a repulsive term s_{corr} on the basis of formula (3.8):

$$\Delta p_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j + s_{corr}) \nabla W(p_i - p_j, h) \quad (2.78)$$

$$s_{corr} = -k \left(\frac{W(p_i - p_j, h)}{W(\Delta q, h)} \right)^n \quad (2.79)$$

Here, Δq means a point at a fixed distance from the particle Δq , usually $0.1h \dots 0.3h$. In addition, k can be regarded as a surface tension parameter (because s_{corr} can produce an effect similar to surface tension, so that the particles on the surface of the fluid are evenly distributed), here are the values $k = 0.1$ and $n = 4$.

As shown in Figure(3.3) and Figure(3.4)

2.9.4 Vorticity Confinement and Viscosity

The PBD method usually introduces additional damping, resulting in energy loss of the entire system, which will cause some vortices that should have disappeared quickly. Similar to [4], PBF reinjects energy into the system through vorticity confinement:

$$f_i^{\text{vorticity}} = \epsilon (N \times \omega_i) \quad (2.80)$$

which

$$N = \frac{\eta}{|\eta|}, \quad \eta = \nabla |\omega|_i \quad (2.81)$$

And the curl formula of the particle is:

$$\omega_i = \nabla \times v = \sum_j (v_j - v_i) \times \nabla_{p_j} W(p_i - p_j, h) \quad (2.82)$$

The basic idea of Vorticity Confinement is: by adding a body force, the curl particles (which can be intuitively understood as particles that rotate faster than the surrounding particles, ω_i points to the axis of rotation of the particle i) In this way, the rotation of the particles is accelerated to maintain the rotation of the system. ϵ Used to control the intensity of Vorticity Confinement.

In the SPH fluid simulation method, Artificial Viscosity can not only increase the numerical stability of the simulation, but also eliminate nonphysical oscillations. In the Lagrangian fluid simulation method, artificial viscosity essentially produces a damping effect on the relative motion of fluid particles, so that the kinetic energy of the fluid is converted into thermal energy. As in [5], PBF uses XSPH to directly update the speed to generate damping in this way.

$$v_i^{\text{new}} = v_i + c \sum_j v_{ij} \cdot W(p_i - p_j, h) \quad (2.83)$$

2.10 Related Work

Motion planning for robot manipulators is an active research field in robotics. Motion planners can be divided into three types. The first type is online instantaneous controllers which only consider the robot configuration and environment observation at the current time instant. Nakamura et al. in [15] first described a scheme for redundant manipulators, in which task priority was introduced by using the redundancy for Inverse Kinematics (IK). Based on [15], An et al. proposed prioritized IK for multiple tasks, which allows for a smooth transition between tasks via a method called task transition control [42]. Flacco et al. presented a method that handles the saturation of the joint command in the null space [43]. Rakita proposed RelaxedIK, which incorporates IK along with obstacle, singularity, and joint bounds avoidance into a nonlinear optimization problem [44].

The second type is offline planners. They generate a trajectory of configurations for the robot joints, and the robot will strictly execute this trajectory. Offline planners can be further

Table 2.2: The state-of-the-art motion controllers and planners comparison.

Controller /Planner	Nakamura et al. [15]	RRT [16]	PRM [17]	CHOMP [18]	STOMP [19]	Schulman et al. [46]	RelaxedIK [44]	Rubagatti et al. [48]	NMPC-MP (Ours)
Real-time	✓						✓	✓	✓
Smoothness		✓	✓	✓	✓	✓		✓	✓
Dynamic obstacle									✓
Octomap/mesh obstacle		✓	✓	✓	✓	✓			✓
Hard obstacle constraint		✓	✓			✓		✓	✓
Optimization				✓	✓	✓	✓	✓	✓
Dexterity		✓	✓	✓	✓	✓		✓	✓
Robust against local minimum		✓	✓		✓				

divided into sampling-based planners and optimization-based planners. The most fundamental sampling-based planning methods are the Rapidly-exploring Random Trees (RRT) method [16] and the Probabilistic Roadmaps (PRM) method [17]. Conceptually, methods of this kind first determine the free configuration space and then generate a trajectory connecting the start and goal points in free space by sampling. Based on RRT and PRM, Karaman et al. proposed RRT* and PRM*, which improve the solution from RRT and PRM [45]. For optimization-based methods, Ratliff et al. designed the CHOMP method that uses covariant gradient techniques to improve the quality of the trajectory [18]. The STOMP method proposed in [19] addresses CHOMP’s disadvantage of getting stuck in the local minimum by generating noisy trajectories and exploring the free configuration space. Schulman et al. formulated an optimization problem for motion planning, in which the cost function is the error between the goal and the current measurement, and collisions are penalized with a hinge loss [46].

The third type of motion planners uses a combination of offline planning and online execution. Methods built on model predictive control (MPC) are typical examples, as they predict the states in a certain amount of time into the future. Only the solution in the first time step of the calculated solution will be executed. Girgin et al. utilized the rich planning redundancy and nullspace that MPC would be a natural fit to solve linear motion planning problems [47]. Using nonlinear model predictive control (NMPC) in teleoperation of a manipulator was successfully demonstrated in [48][49] [50]. Although these methods are real-time capable, they can only handle simple static obstacles. Table 2.2 shows the merits and demerits of the controllers and planners that have been introduced and sets goals that we want our NMPC based method to achieve.

Table 2.2 shows the merits and demerits of the controllers and planners that have been introduced and sets goals that we want our NMPC-MP to achieve.

The shared autonomy (SA) paradigm can be categorized into three major parts: The first is the robot inference subsystem. Several baseline works such as [4], [51] use the Hidden Semi Markov Model (HSMM) [52] to predict the operator’s intention by only using the temporal and spatial motion [53]. Aronson et al. improved the prediction by leveraging gaze eye-tracking [54], and vision-based approaches try to improve the classification using video feeds in remote surgery [55] using neural networks. In [56], the intention prediction prob-

lem in teleoperation is converted into an inverse reinforcement learning (IRL) problem. In addition, the use of a Layered Hidden Markov Model (LHMM) to model human skills are indicated in [57]. [58] presented a combination of HMM and virtual fixtures for the recognition of user motions. [59] performed the recognition of task segments using hierarchically clustered Hidden Markov Models and increased the robot’s joint manipulation skills with incremental learning. [60] proposed the Intention-Driven Dynamics Model (IDDM) to infer the intention from observed movements using Bayes’ theorem.

The second part is motion reproduction. Dynamic motion primitives (DMPs) [61], the combination of Hidden Markov Model (HMM) and Gaussian Mixture Regression (GMR) [62], HSMM and LQT [4] have been used for motion segmentation and synthesis for robotics. In our work, we adopt motion reproduction using HSMM and LQT from [51] and [4].

The third subsystem is the control switching strategy known as arbitration. This can be modeled as a non-linear function that blends the human and autonomous control input signals [63], [64]. This thesis targets discrete switching from direct teleoperation to autonomous control in this subsystem of shared-autonomy. As stated above, adaptation is the most desirable feature when a robot operates in a dynamically changing environment. For high latency teleoperation (> 1 s) such as deep space missions, [65] proposed physics simulation and scene representation to adapt and bridge the gap between autonomous control and direct teleoperation. In the shared control (SC) paradigm, virtual fixtures have been applied in many manipulation tasks by limiting the movement at the remote side into restricted regions along desired paths [58] [66]. This thesis introduces a novel online skill refinement approach using the Coherent Point Drift (CPD) algorithm to minimize the spatial displacement for control switching. In Table 2.3 we summarize the state-of-the-art solutions and compare them with our approach (named Skill-CPD in the following) from different perspectives.

Table 2.3: Comparison of the state-of-the-art shared autonomy-based teleoperation.

Criteria	Tanwani et al. [51]	Tian et al. [4]	Skill-CPD (Ours)
2D scenarios	✗	✓	✓
3D scenarios	✓	✗	✓
Min. displacement	✗	✗	✓
Rigid registration	✗	✗	✓
Non-rigid registration	✗	✗	✓

As a result of research related to precision pouring, the approaches can be divided into two categories, classical closed-loop control approaches and machine learning-based methods.

Closed-loop control approaches do not require data-driven methods and use mechanisms such as Proportional-Integral-Derivative (PID) controllers. For instance, Dong et al. proposed a PD controller-based approach [67]. In their approach, they propose calculating the volume of the poured liquid using the model of the target container and the height of liq-

liquid contained in it. Likewise, for controlling actions of pouring, they propose using a (PD) controller, using the pouring container's angular speed as a process variable and poured volume as a control variable. A prominent drawback observed was the inability to factor in the effect of surface tension of the water. Since the PD-based mechanism is not aware of these liquid-specific variables, it cannot generalize to get the best results.

Learning from Demonstrations (LfD) is one of the popular methods to teach robots a particular skill [68], [69]. Usually, a human teacher demonstrates the skill using teleoperation. Rozo et al. proposed an innovative approach using LfD and force feedback-based perception [70]. The underlying policy is based on a Parametric Hidden Markov Model for Behavior Cloning. However, the model is blind towards the type of the liquid it works with. As a result, for each novel liquid, a new set of demonstrations would be required and hence is not generic for all liquid types.

In [71] Linag et al. proposed a Multimodal Pouring network (MP-Net) to robustly predict the liquid height by considering audio and haptics as input. Using raw audio data, a spectrogram with 257 descriptors is created. The haptic data associated with each time slice is gradually fed into the encoder module (a recurrent neural unit). In the final step, the height predictor module calculates the 1D length of the air column in the target container. Using a multimodal pouring data set including 300 recordings and force-torque measurements, the MP-Net is trained based on three types of containers. However, it suffers from the same prominent drawback, where the underlying model is blind towards the liquid properties. This is observed as a poor generalization of MP-Net to liquids of approximate similar nature like milk and fruit juices [71] which perform poorly, although for water the results are very promising.

Schenck et al. [72], [73] described a machine-vision-based approach, which uses visual feedback to perform closed-loop control for pouring liquids [74]. To detect liquids, convolutional neural networks (CNNs) are used to detect the type of container and the volume of liquid within it based on RGB and thermal images [75]. The CNN-based output is restricted to merely 10 classes of volume. This approach, hence, is not completely representative for precise pouring. Based on the pouring model trained on a sequence of 279 pouring actions, the pouring policy tested on water observed an average 38ml deviation [74] from the target amount. Besides, the model is unaware of the liquid properties. In addition, the scenarios involving occlusion of camera feedback would greatly impact the performance.

Chau et al. [76] presented a deep reinforcement learning-based approach to learn a policy for pouring using Deep Deterministic Policy Gradients (DDPG) [77]. In their work, the training experiences were collected using a state-of-the-art liquid simulator PreonLab [78]. Through their experiments, performed with a PR2 robot, they demonstrated a successful transfer of the learned policy to a real robot. Despite the absence of force sensors, the task involved pouring liquids using a robot at specific heights while avoiding spills and collisions. Using force/torque sensors can on contrary provide means for better precision pouring control. In Table 2.4 we summarize the state-of-the-art solutions and compare them with our approach (named PourNet in the following) from different perspectives. We will explain our method in chapter 6.

Table 2.4: PourNet compared to the other state-of-the-art approaches.

Evaluation Criteria	Closed-loop Control [67]	LfD [70]	MPNet [71]	Vision CNN [74]	DDPG [76]	PourNet
Liquid Properties Awareness	✗	✗	✗	✗	✓	✓
Container Geometry Awareness	✓	✗	✗	✗	✗	✓
Ability to control multiple actions	✗	✓	✓	✗	✓	✓
Generalization to novel liquids	✗	✗	✗	✗	✓	✓

2.11 Chapter Summary

This chapter introduced the most relevant background needed to better comprehend the reasoning discussed and the concepts proposed in this work.

The first part addressed the basics of robot kinematics and the basics of motion planning and a short introduction for model predictive control. State-of-the-art technologies that approach this issue were presented and thoroughly debated. Intention recognition in teleoperation was discussed next where we adapted the basic knowledge and extended it using the novel non-rigid registration method. At the end the concept of reinforcement learning was mentioned which enabled us to train the pouring skill in simulation using trial and errors with the PPO algorithm. Later in this dissertation, we will expand on each in the subsequent chapters (4,5,6).

Chapter 3

Experimental Setup and Datasets

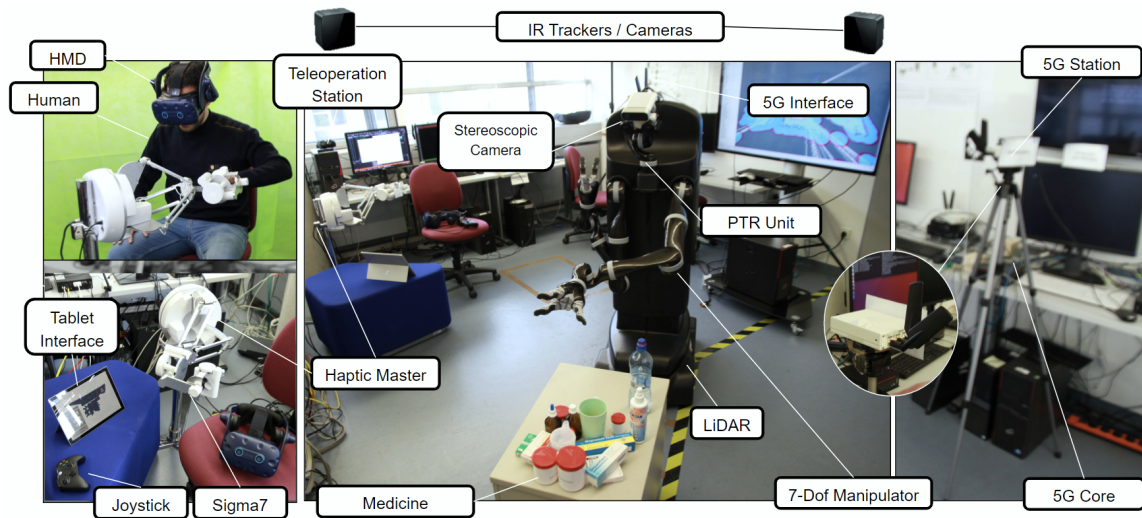


Figure 3.1: 5G Research Hub Munich. The image illustrates the room setup with different pieces of equipment.

This chapter briefly revisits the structural basics of the Movo telepresence platform that was used to test the algorithms presented in this thesis.

3.1 Movo Telepresence Platform

Kinova Movo is a mobile manipulation platform designed to aid in the research and experimentation of mobile robotics. Movo is fully designed and developed using the Robotic Operating System (ROS) and offers an application programming interface (API) that provides researchers the most advanced functionality in all areas of robotics. Its unique combination of performance, scalability, modularity and openness allows one to configure a robot for application specific needs. [Figure 3.1](#) illustrates the experimental room in which Movo was located.

3.2 Upgrading Movo's Hardware

In order to meet the project requirements and increase the robot's functionality for assistive teleoperation we upgraded the robot in two major steps. First we exchanged the old head with a new stereoscopic head and second we installed industrial grade parallel grippers as well as Force/Torque (FT) sensors to the manipulators. Figure 3.2 shows the differences between the original version and the upgraded version.

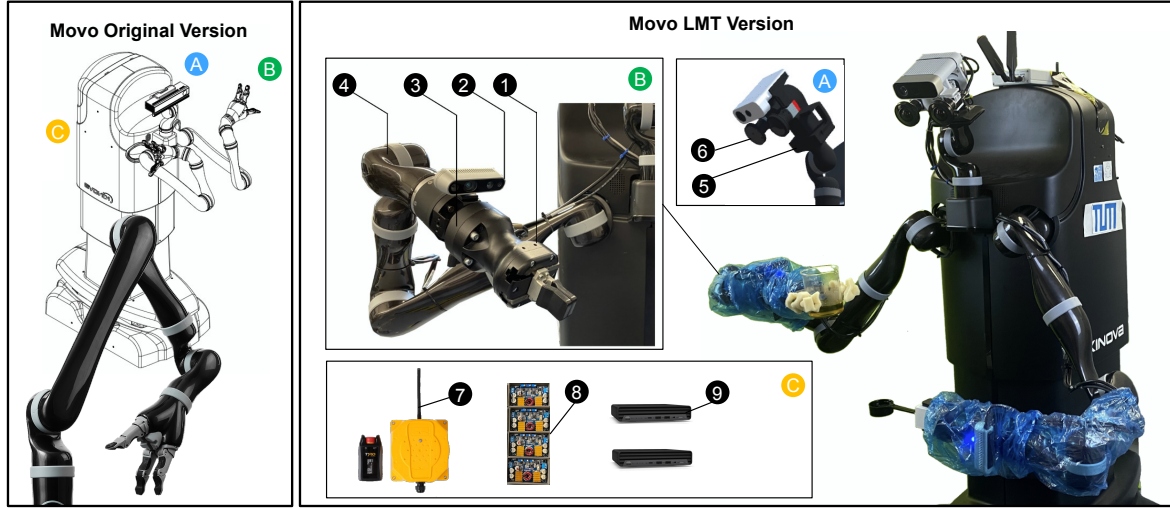


Figure 3.2: Movo platform, LMT version. 1) Robotiq® Hand-e gripper 2) Intel® Real-sense camera 3) Botasys® F/T sensor 4) Kinova® Jaco2 manipulator 5) The roll actuator to extend the head motion with the new Robotis® Dynamixel MX-106 actuator 6) XIMEA® Stereo cameras 7) TYRO® Wireless emergency stop 8) Customized power regulators 9) Hp® embedded computers.

3.2.1 Setup of a New Jaco2 Robot Model with Hand-e Gripper

As part of the upgrade, a new Hand-e parallel gripper had to be installed in place of the three finger gripper. We assembled the hand not only mechanically, but also updated the Universal Robot Description Format (URDF) of the robot to accommodate the new hand parameters. From the previous URDF file, the eighth link with three fingers was deleted and replaced with the new gripper. The inertial matrix, mass, and meshes for the new gripper can be found on Robotiq's Github repository¹.

$$Inertial_{base} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} 1017560E-9 & 0 & 2235E-9 \\ 0 & 1028041E-9 & 0 \\ 0 & 0 & 489810E-9 \end{bmatrix} \quad (3.1)$$

For the left and right fingers (same mass 0.03804 Kg), their inertial parameters (I_l , I_r) are shown in Equation 3.2 respectively.

¹ Github page: <https://github.com/cambel/robotiq> (Commit: Feb 10, 2021)

$$I_r = I_l = \begin{bmatrix} 13567E - 9 & 1849E - 9 & 3622E - 9 \\ 0 & 15784E - 9 & 3616E - 9 \\ 0 & 0 & 7478E - 9 \end{bmatrix} \quad (3.2)$$

3.2.2 Force/Torque Sensor and Filtering

A robotic force torque sensor is a device that measures force and torque when they are applied. Through this a signal is created, measured, recorded, and used as a feedback signal in human-robot interaction. The most widely used sensor in robotics is a 6-axis force torque sensor where 3 forces and 3 torques can be measured. Static and dynamic force torque measurement devices measure strain induced from applied forces/torques using resistive, capacitive, and optical technologies. Among these, the most reliable and predictable measurements are resistive strain gauges. When compared to other technologies, these gauges are very effective when working with steel, aluminium, or titanium. In this dissertation we employed Botasys SensOne FT sensor on the wrist of the robot. The maximal reachable frame rate is 800Hz, in our experiments we worked with 100 samples per second.

Although Jaco2 has torque sensors on each joint, the gripper's control and grasping are performed in Cartesian space. Therefore, in order to increase precision and sensitivity to external forces and torques, an industry grade FT sensor was installed on the wrist between the gripper and the manipulator's final joint. The Table 3.1 provides specifications of the corresponding FT sensor with serial communication.

3.2.3 Low-Pass Filter

In signal preprocessing, a filter is a process that is responsible for cutting off frequencies to reduce background noise so that unwanted values are removed from a signal. In this dissertation we make use of low-pass filter. A low-pass filter is a filter that only allows signals below a certain cutoff frequency to pass. There are various types of filters, we will use the Butterworth low pass filter. The Butterworth filter was first described in 1930 [17] and is today one of the most common frequency domain filters. It is a type of an active low pass filter that has been used in several signal denoising applications. It is considered as an anti-aliasing filter in data converter and audio processing applications. Moreover, in some research work [9], using low pass filter has shown good results when applied on FT signals [18]. The frequency response of Butterworth filter is smooth at the frequencies of the pass-band and almost zero at the frequencies of the stop band. It is known as a "maximally flat magnitude filter". The the n th-order Butterworth filter is defined in terms of its frequency response as follows:

$$H(j\omega) = \frac{1}{\sqrt{1 + \varepsilon^2 \left(\frac{\omega}{\omega_c}\right)^{2n}}} \quad (3.3)$$

where

n : order of the filter = 2

w : passband frequency = 50

w_c : cut-off frequency = 1

The filter is characterized by two parameters: one is the cut-off frequency and one is the order of filtering which is a parameter that determines the steepness of the filter's transfer function. Fig 3.3 illustrates the filtering result after data acquisition. For DSP implementation we used the design from Dr. Bernd Porr accessible from his official Github repository².

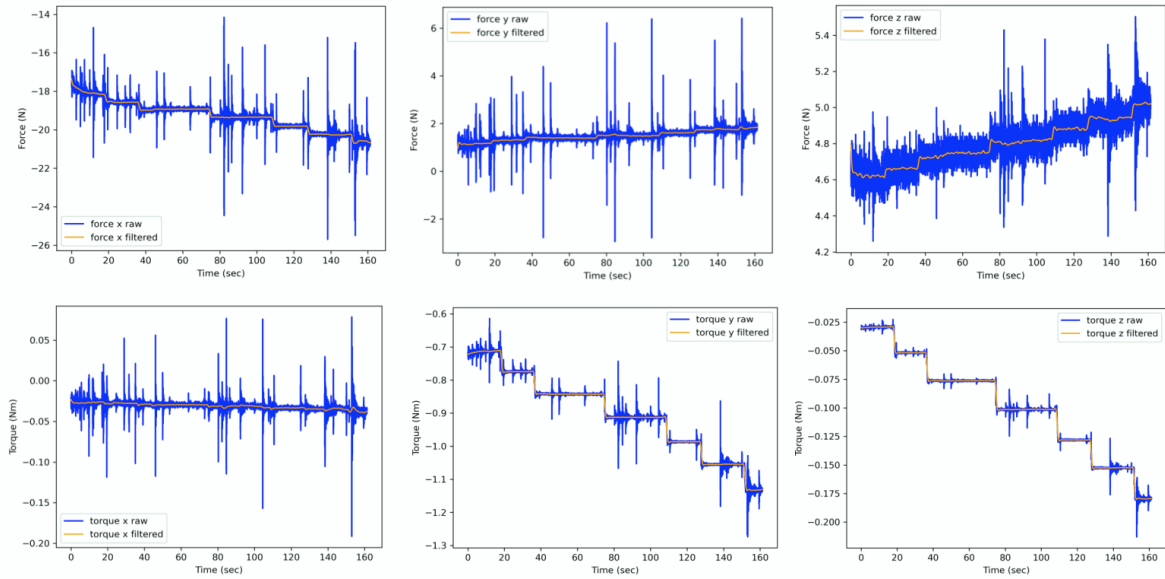


Figure 3.3: Butterworth filter result. The raw sensory data is shown in blue, and the result of a low pass filter is shown in orange.

3.2.4 Installation of the Real-sense camera

In addition to the force torque sensor which is mounted before the Hand-e gripper for gravity compensation algorithm an Intel Real-sense D435i camera is mounted on the gripper to provide (Inertial Measurement Unit) IMU data. See Figure 3.2 and Figure 3.4. The IMU has an accelerometer to measure the acceleration in $[m/s^2]$, a Gyroscope for angular velocity calculation $[rad/s]$ and its frequency is configurable (200 Hz – 400 Hz). The important task is the calibration of the IMU and the gravity compensation, in order to receive external torque signal without disturbances due to temperature and transient current, plus the gravity induced torques/forces should be compensated after the incorporating of the FT sensor and the Hand-e gripper in the Jaco2 robot arm.

The calibration parameters of the IMU include intrinsic and extrinsic parameters. In the intrinsic side, there are the scale factor (S_x, S_y, S_z) , the Bias (a_{bias}) and the Off-axis terms $(C_{xy}, C_{yx}, C_{xz}, C_{zx}, C_{yz}, C_{zy})$ of the accelerometer and the W_{bials} term for the Gyroscope.

² Github page: <https://github.com/berndporr/iir1> (Commit: Dec 27, 2021)

The following transformation equations represent the mapping between raw data output and real measurements by modelling the inaccuracies and temperature-dependent drift of the sensor.

$$\mathbf{a}_{true} = \begin{bmatrix} S_x & C_{xy} & C_{xz} \\ C_{yx} & S_y & C_{yz} \\ C_{zx} & C_{zy} & S_z \end{bmatrix} \mathbf{a}_{raw} - \mathbf{a}_{bias} \quad (3.4)$$

$$\mathbf{w}_{true} = \mathbf{w}_{raw} - \mathbf{w}_{bias} \quad (3.5)$$



Figure 3.4: The new Hand-e gripper and the 6 axis force torque sensor and the Intel Real-sense camera.

Table 3.1: Serial SensONE Force Torque Sensor from BOTA system.

Specifications	SensONE Serial
Ranega (F_{xy}, F_z, M_{xy}, M_z)	500 N, 1200 N, 15Nm, 12Nm
Overload (F_{xy}, F_z, M_{xy}, M_z)	2500 N, 4500 N, 35Nm, 40Nm
Noise Free Resolution (100 Hz)	0.15 N, 0.15 N, 0.005Nm, 0.002Nm
Weight	220 g
Size (DxL)	70 × 35 mm
Communication	USB, RS422
Sampling Rate (Max.)	800 Hz
IMU	external (Real sense camera)
Ingress protection	dustproof and water resistance
Operating temperature	0 – 55 Celsius
Power supply	5 V, 1.0 W

3.2.5 Gravity Compensation

We used the calibration algorithm from [79] for gravity compensation of the gripper and our calibration software³ ran only once to guide the robot arm (end effector) to different positions (these positions may be randomly determined or manipulated manually). The FT sensor and

³ Github page: <https://github.com/kth-ros-pkg/force-torque-tools> (Commit: May 7, 2021)

accelerometer signals stored in these poses (Cartesian x, y, z and Quaternion for the orientation) are then used to compute a least square estimation of the Bias term of the force torque sensor, the mass of the gripper and the location of the center of mass of the attached gripper. These estimations are stored in a configuration file. The compensation algorithm then uses the stored estimated parameters from the calibration phase and filters the raw data of the FT sensor afterwards. The output signal from the force torque sensor should be only sensitive to external forces or torques, otherwise no zeros signals are expected in output.

The center of mass as well as the mass and the inertial matrix of the used Hand-e gripper with Real-sense camera D435i and FT-sensor are shown in Table 3.2.

Table 3.2: Parameters of the new gripper.

Product	Center of Mass [mm]			Total Mass [g]	Total Length [cm]
FT-sensor+camera+Hand-e	-1.1	0.8	78.6	1435.0	16.0

Figure 3.5 illustrates the output readings of a calibrated torque sensor used in a peg-in-the-hole application. When there was no contact with the surface, the force-torque readings were zero. In motion, the maximum noise error was 2N for force and 0.04Nm for torque. In idle, the maximum noise error was 0.5N for force and 0.01Nm for torque. In both motion and idle conditions, this allows us to obtain accurate measurements. In idle mode, we were able to achieve an accuracy of ± 2 grams.

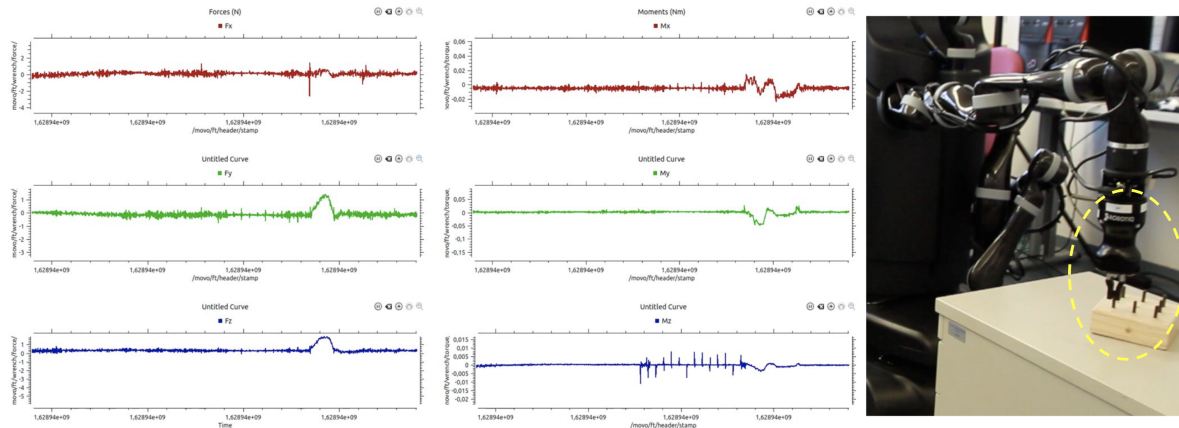


Figure 3.5: Illustrations of a peg-in-the-hole application and the results of the experiment.

3.3 Upgrading Movo's Software

3.3.1 Shared Autonomy Interface (SAI)

Our goal was to be able to change the input control to the robot quickly and intuitively for different experiments, so we developed a GUI interface using Microsoft .NET. The GUI serves as a network multiplexer between different external input devices and is the central monitoring and control system for the robot. Specifically, it bridges with ROS using a high

performance bridge that utilizes the ZMQ network SDK to implement a pub/sub paradigm with TCP/IP sockets and Google protobuf serializers and deserializers to encapsulate messages to the robot and vice versa. Several external interfaces are compatible with the shared autonomy interface, including the HTC Vive, Xbox Joystick, Sigma7, Phantom Omni, Tablet interface, and standard mouse and keyboard inputs. Figure 3.6 illustrates the GUI interface and Figure 3.7 shows the software architecture of different components in the experimental setup.

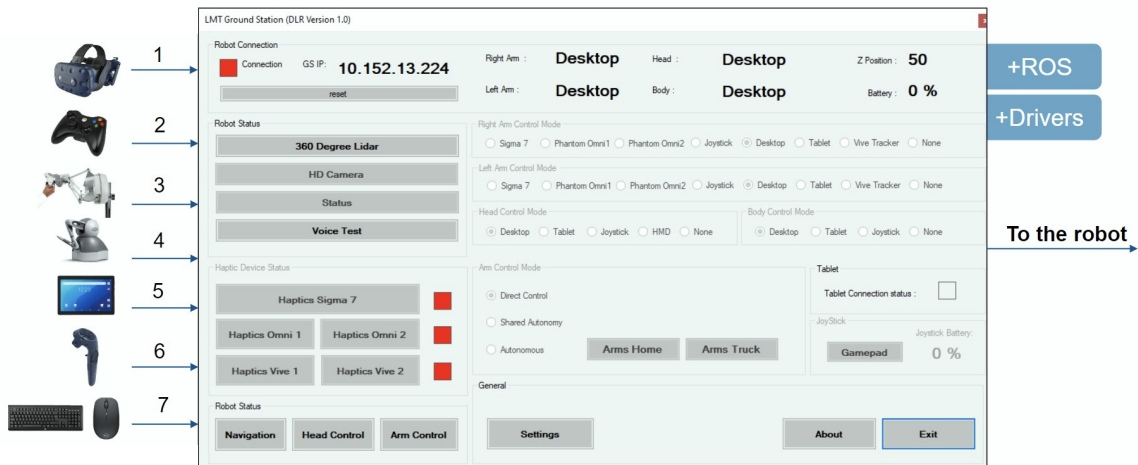


Figure 3.6: LMT's shared autonomy interface, the graphical user interface (GUI) software.

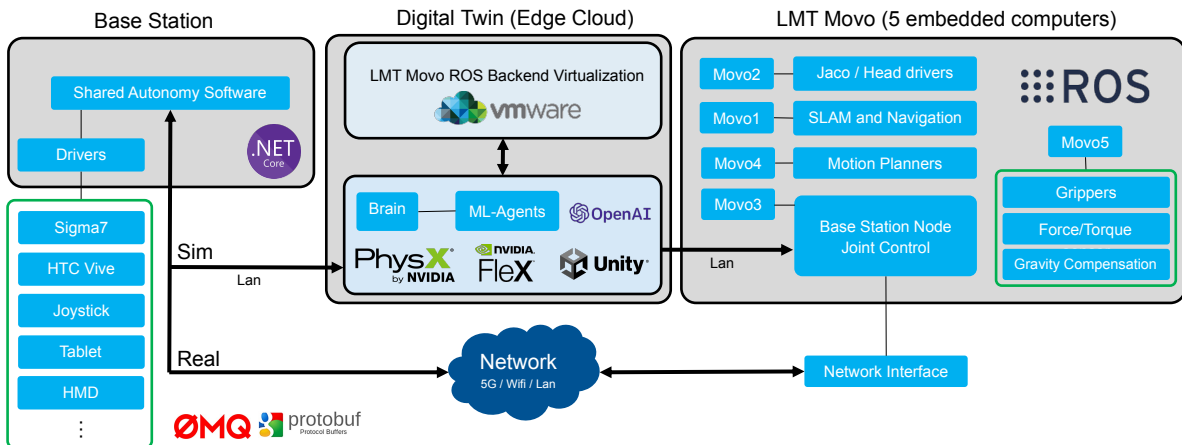


Figure 3.7: Software architecture

Figure 3.7 depicts the software architecture of the setup. There are two major concepts represented here. First of all, there is the teleoperation interface and the shared autonomy interface which are located in the base station computer in order to be able to control the robot remotely. Wireless, LAN, or 5G networks may be used as the network. The second concept is the digital twin of the robot which later we introduce it as realistic robotic simulator for pouring liquids. Our dissertation aims to minimize the simulation-to-reality gap by providing the same robot back end as a virtual computer. We used Unity3D in conjunction with NVIDIA's Physics to simulate the robot. In addition, we used the ML-agent OpenAI interface of Unity3D

to train skills to our robot using simulation. All other robot software is located on different computers (Movo1-5). ROS is responsible for the interprocess communication. Our custom bridge was used [80] to connect ROS to the .Net environment, particularly Unity3D.

3.3.2 NVIDIA FLEX

The theoretical basis of fluid simulation in Nvidia FleX is PBF and the treatment methods of surface tension, viscosity, cohesion and adhesion that are unique to fluid simulation. Compared with the PCISPH method, PBF is more stable, allows large time steps, and is more suitable for the simulation of game physics engine fluids. In addition, both the SPH-based method and the PBD-based method are very suitable for parallelization,

Algorithm 1 describes the simulation loop. It's similar to the original Position Based Dynamics update, except that instead of using sequential Gauss-Seidel iteration, each constraint is solved independently in a Jacobi iteration. As part of the constraint solving process, collisions between solids have been detected.

Each solver iteration, particle neighborhoods have been recomputed once per step and recalculate distance and constraint values. When a particle separates from its initial set of neighbors, this optimization can result in density underestimates. This can cause serious problems in PCISPH because once a particle is isolated, each iteration lowers its pressure. Large erroneous pressure forces are applied if it comes back into contact on a subsequent iteration. Because this algorithm only considers current particle positions (not accumulated pressure), this problem does not arise.

Algorithm 1 Simulation Loop

```

1: for all particles  $i$  do
2:   apply forces  $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$ 
3:   predict position  $\mathbf{x}_i^* \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$ 
4: end for
5: for all particles  $i$  do
6:   find neighboring particles  $N_i(\mathbf{x}_i^*)$ 
7: end for
8: while  $iter < solverIterations$  do
9:   for all particles  $i$  do
10:    calculate  $\lambda_i$ 
11:   end for
12:   for all particles  $i$  do
13:    calculate  $\Delta \mathbf{p}_i$ 
14:    perform collision detection and response
15:   end for
16:   for all particles  $i$  do
17:    update position  $\mathbf{x}_i^* \leftarrow \mathbf{x}_i^* + \Delta \mathbf{p}_i$ 
18:   end for
19: end while
20: for all particles  $i$  do
21:   update velocity  $\mathbf{v}_i \leftarrow \frac{1}{\Delta t} (\mathbf{x}_i^* - \mathbf{x}_i)$ 
22:   apply vorticity confinement and XSPH viscosity
23:   update position  $\mathbf{x}_i \leftarrow \mathbf{x}_i^*$ 
24: end for

```

Figure 3.8: PBF Algorithm[37]. ©2013 ACM.

3.3.3 Digital Twin and PhysX Implementation

An articulation is a set of bodies arranged in a logical tree. The parent-child link in this tree reflects that the bodies have their relative motion constrained. Articulations are solved by a Featherstone solver that works in reduced coordinates - that is each body has relative coordinates to its parent but only along the unlocked degrees of freedom. This guarantees there is no unwanted stretch. The Movo's digital twin was developed using Unity3D engine and the upgraded PhysX 4.1 technology for articulation bodies. PhysX 4.1 introduced the new Temporal Gauss-Seidel (TGS) solver which has improved joint drive accuracy and non-linear rigid body solver with high-mass ratios. In chapter 6, we used this simulator to test the robot in the loop scenarios. We plan to release the public repository of our simulator in the near future⁴. You can find more technical details on implementation here [81].



Figure 3.9: Photo realistic high definition rendering pipeline in Unity3D on the left and sensor outputs on the right. All the robot sensors such as IMU, LiDAR, cameras, depth cameras, transformations, and joint force and position feedback are provided to the ROS side through RRS.

3.3.4 Tablet Interface

During the project, a tablet interface based on the Unity3D engine has been developed in order to test the robot navigation and general functionality. It is directly connected to the shared autonomy interface, and can send and receive commands from human operators as well as status information from the robot. The purpose of this interface is to perform remote assistive teleportation in complete autonomy with the robot. With this interface, liquid pouring as well as autonomous navigation has been tested.

⁴ Github page: <https://github.com/cxdxcd/RRS> (Commit: June 03, 2022)

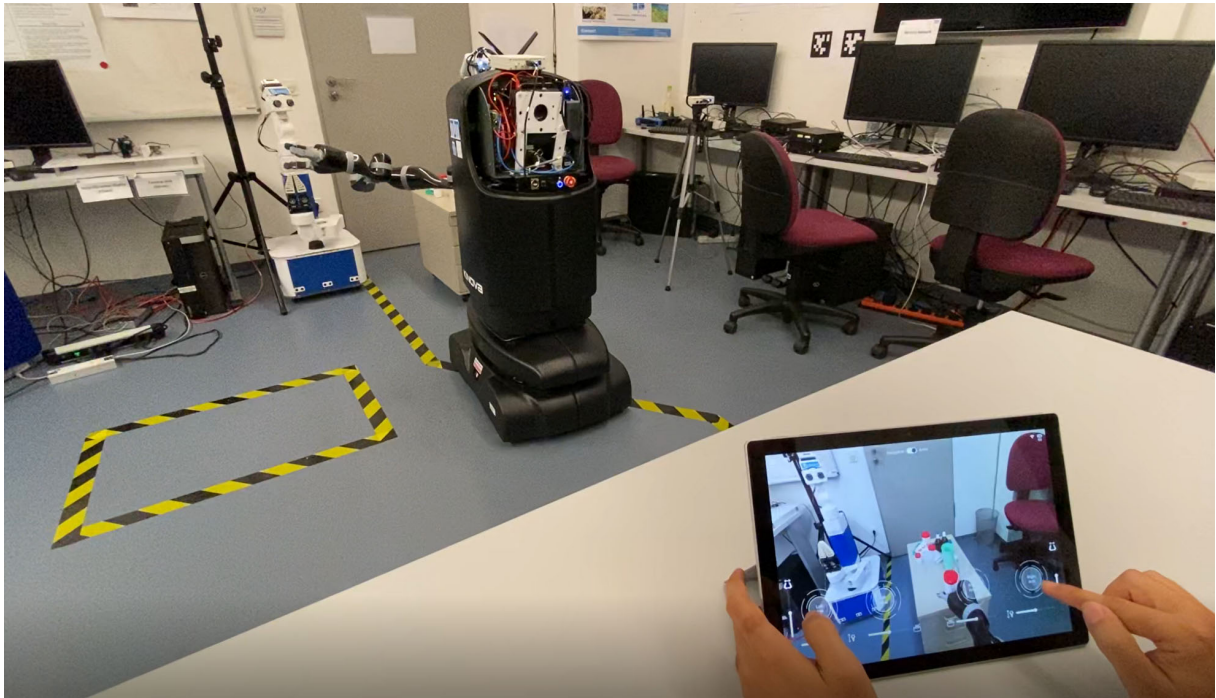


Figure 3.10: Operating Movo directly from the app using a tablet

3.4 Datasets

3.4.1 YCB Standard Model Dataset

The YCB Object and Model Set [82] is intended to aid in robotic manipulation benchmarking. The set includes everyday objects with varying shapes, sizes, textures, weight, and rigidity, as well as some commonly used manipulation tests. The physical objects are distributed to any research group that registers on the website. They hope that the ready availability of physical objects will allow the community of manipulation researchers to more easily compare approaches and continuously evolve benchmarking tests as the field matures. The set is linked to a model database, which contains mesh models and high-resolution RGB-D scans of the objects, allowing them to be easily integrated into manipulation and planning software platforms.

3.4.2 Liquid Profile Dataset

3.4.2.1 Liquids

Several liquids were used for experimentation. Due to the experiments involve fluid dynamics and viscosity, 10 different liquids were selected in order to have the broadest liquid properties covered such as their appearance, transparency and mainly diverse levels of viscosity.



Figure 3.11: YCB Object Set [82]. ©2015 IEEE.

3.4.2.2 Density and Viscosity

Although in the industry the liquids' parameters such as viscosity are measured with highly precise instruments, looking at the qualitative effects of different liquids and their attributes does not require highly sophisticated technology. First of all, we need to obtain the respective densities of the 10 liquids presented. For this, we proceeded as follows: using the measuring glasses and the scale we poured each liquid four times and measured its weight at 50 ml, 100 ml, 150 ml and 200 ml, respectively.

Then we calculate the mean of the four values and using equation 3.1 we obtain the density ρ , where m is the mass and V the volume from the respective liquid.

$$\rho = \frac{m}{V} \quad (3.6)$$

After obtaining the density from each experimental liquid we elaborate Table 3.3 and present the results.

Table 3.3: Density calculation from liquids.

Liquids	50 ml [g]	100 ml [g]	150 ml [g]	200 ml [g]	Mean [g]	Density [g/ml]
Apple Juice	46	98	148	196	49	0.98 ± 0.14
Washing soap	50	102	156	212	53	1.06 ± 0.15
Handgel	40	82	124	162	40.5	0.81 ± 0.12
Honey	74	156	228	300	75	1.5 ± 0.19
Rapeseed oil	40	90	136	184	46	0.92 ± 0.13
Balsamico	50	106	160	210	52.5	1.05 ± 0.15
Joghurt	64	110	166	216	54	1.08 ± 0.15
Hand soap	52	102	154	200	50	1 ± 0.14
Shampoo	52	102	154	200	50	1 ± 0.14
Milk	48	100	150	198	49.5	0.99 ± 0.14

Name	Image	Name	Image
Joghurt Drink Erdbeer Bananen		Honey - Wald Honig	
Milk - Frischen Alpen Milch		Apple juice	
Dishwashing soap - Spülmittel Aloe Vera		Rapeseed oil - Rapskernöl	
Hygiene Handgel		Balsamic vinegar - Aceto Balsamico di Modena	
Hand Soap - Flüssigseife Milch & Honig		Shampoo	

Figure 3.12: Real liquids used for the experiments.

Subsequently we proceed to look qualitatively and quantitatively how viscosity affects solid objects and their differences. As shown in Figure 3.13, in some liquids the marble is more visible than others. Therefore, the velocity could not be measured for some dark liquids like balsamico and shampoo. In Figure 3.13 we can observe the marble position after 0.5 seconds in water, apple juice, oil, hand-gel, dish soap, hand soap, honey, respectively. Whereas the marble already came to the bottom of the cylinder filled with water, in honey the marble is not even entire submerged at the same time. Due to the marble's velocity we had to film recordings at 400 fps to determine how much time it takes the marble to travel 50 ml trough every liquid. Three trials were performed for each liquid.

Then the mean time was calculated and finally the velocity was obtained dividing the

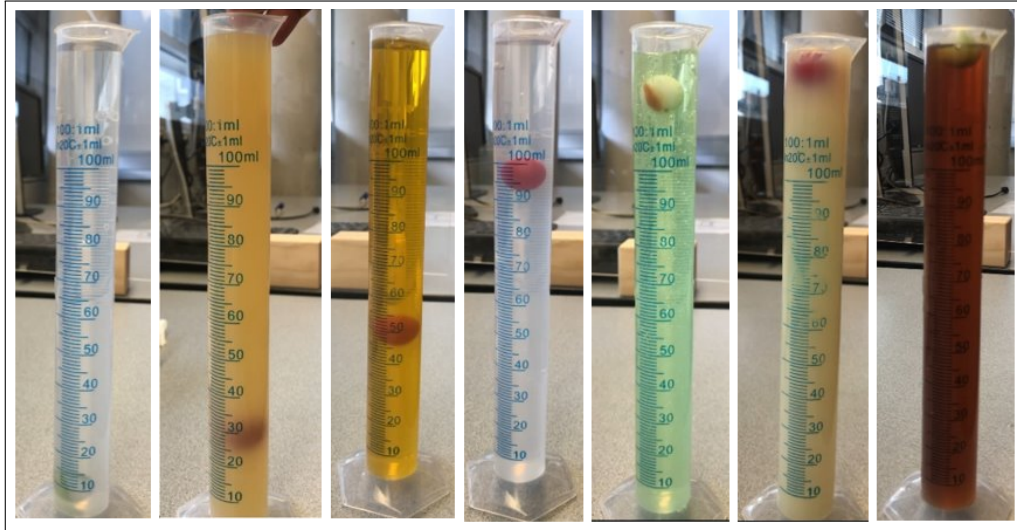


Figure 3.13: Marble position in 6 liquids after 0.5 seconds.

Table 3.4: Measurements of different liquids for obtaining their viscosity

Liquids	1. Trail [s]	2. Trail [s]	3. Trail [s]	Mean [s]	Speed [m/s]	Viscosity [cP]
Water	0.2	0.22	0.22	0.2133	0.4173	0.87
Apple	0.27	0.29	0.29	0.2833	0.3142	1.15
Oil	0.33	0.34	0.35	0.34	0.2618	1.39
Handgel	1.3	1.14	1.13	1.19	0.0748	4.85
Dish soap	3.87	3.58	3.6	3.6833	0.0242	14.99
Hand soap	9.33	9.04	9.02	9.13	0.0097	37.4
Honey	72.78	73.45	76.49	74.24	0.0012	302.35

distance travelled, i.e., 8.9 cm through the respective time. The velocities are presented below. Using Equation 3.7 we calculate viscosity μ , where r is the radius of the marble, g the gravitational acceleration, ρ_s the density of the marble, ρ_f the density of the liquid and V_s the velocity of the marble. The viscosity is obtained in Pascal seconds [Pa·s], but the most common magnitude is centipoise [cP]. Therefore we multiply by a magnitude of 3 and show the results in Table 3.4.

$$\mu = \frac{2}{9} \frac{r^2 g (\rho_s - \rho_f)}{V_s} \quad (3.7)$$

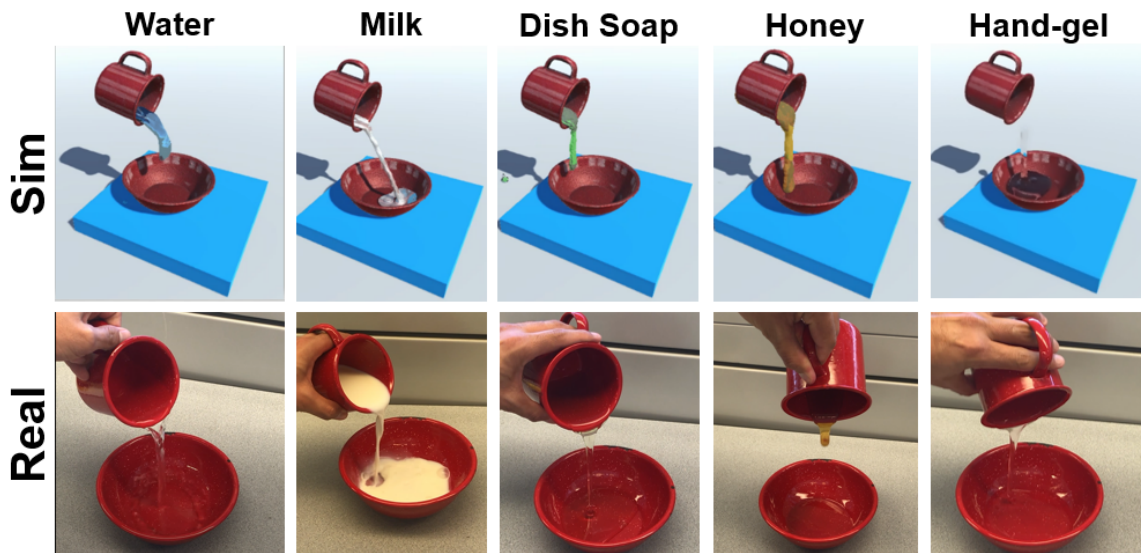
The final experiment focuses deeply in tuning the parameters of the simulator based on the appearance of each liquid in the real world while pouring and model them as similar as possible. Five different simulated liquids were compared with their real-world counterparts.

Factors such as liquid appearance and how it behaves when it finds itself in the target container can be further improved in some liquids like in water, milk and hand-gel. Despite the fact that the parameters are dependent and they will be used for more viscous liquids to keep the consistency in all liquid types we selected the best possible parameters with

Table 3.5: Liquids' parameters in RRS Simulator

Parameters	Water	Milk	Dish Soap	Honey	Ketchup	Hand-gel
Solid Rest	0.005	0.01	0.001	0.05	1	0.001
Fluid Rest	0.14	0.15	0.16	0.16	0.16	0.15
Static Friction	0.1	0.5	0.5	1	2	0.1
Dynamic Friction	0.1	0.5	0.5	1	2	0.1
Particle Friction	1	0.5	0.5	1	2	1
Adhesion	0	0	0	0.003	0.01	0
Dissipation	0	0	5	4	0.2	0.2
Damping	0	0	0.5	2	2	0
Cohesion	0.03	0.03	0.05	0.05	0.1	0.04
Surface Tension	0	0	0	0	0.01	0
Viscosity	0.3	0.5	0.8	1	0	0.5
Buoyancy	2	2.1	3	3	1	2.2

trial and error and real footage comparison manually. The tuning of some parameters does not seem to affect the appearance of the liquid such as surface tension in Table 3.5, some of them change drastically the behaviour after a small tuning like adhesion. Furthermore, it seems that some parameters have more weight than others and therefore their tuning affects and covers the effects from the other parameters. Following this notion we can affirm that concentrating only on how the liquid is transferred from source to target container is more valuable than the appearance of the liquid. Figure 3.14 illustrates simulation and real experiments for 5 different liquids.

**Figure 3.14:** Qualitative comparison of 5 different liquids in simulations and reality.

3.5 Chapter Summary

This chapter introduced the experimental setup that is used to evaluate the proposed approaches. After a general overview of the Movo's hardware upgrade the software architecture and the liquid simulator was discussed in details. At the end the simulated liquid profiles from real experiments are introduced which are implemented using NVIDIA's Flex in real-time.

Chapter 4

Nonlinear Model Predictive Control for Real-time Motion Planning



Figure 4.1: Real-time motion planning for dual-arm Kinova[®] Movo platform using the proposed NMPC-MP for low-level shared autonomy in teleoperation. The orange object is the obstacle, and the dashed-red arrow denotes the desired motion. The dashed-yellow trajectory is the generated motion in order to avoid the obstacle. [1], ©2021 IEEE.

Conventional Motion planners usually run in an offline manner. For the teleoperation of the Jaco2 manipulator in this dissertation, a motion planner needs to run at 10 Hz to generate the motion control given a goal sent by the teleoperator. Currently, there are no motion planners who can achieve this frequency. Schulman’s approach can achieve the fastest performance among planners: $100\text{-}200\text{ ms}$ [46]. An alternative is using Nakamura’s instantaneous control method [15] as a lower layer controller for teleoperation to fulfill the frequency requirement. But in contrast to offline motion planners, a great deficiency of the instantaneous control is that it lacks redundancy and dexterity. In a complicated environment, instantaneous control methods may fail to drive the robot to reach the goal while avoiding obstacles,

whereas motion planners are able to find a path heuristically. For the teleoperation application in this dissertation, the NMPC-MP is not required to be robust against local minimum, as in a shared autonomy system, operator gains more control and can guide the robot out of local minimum when the robot gets stuck in it.

Model predictive control (MPC) found its application in process industries back in 1980s. With a significant growth on computing power of microprocessors, MPC now can be applied to solve complex problems from various fields. In robotics motion planning, Nonlinear MPC (NMPC) is a good choice. It is able to fulfill most of the merits listed in Table 2.2 for the following reasons: (i) NMPC plans a sequence of control moves and thus is dexterous to handle complex problems. (ii) Using Sequential Quadratic Programming (SQP), the NMPC problem can be solved quickly, thus can be applied in real-time. (iii) NMPC has a preview capability, which makes its solution proactive to system changes. As a result, NMPC-MP generates smooth trajectories despite the changes in the goal and environment. (iv) Arbitrary objective criteria and constraints can be added to the optimization, making the NMPC problem highly customizable. The concept of NMPC is to use a dynamic model of the system to predict the future states of the system and optimize one or multiple future-state-dependent metrics to produce a sequence of optimal movements. However, only the control movement of the first time step is executed. For the next time step, system states may be different from what has been predicted before. Then, a new NMPC is formulated and solved to generate new control moves. Therefore, NMPC is proactive yet still quickly responsive to system dynamics and noise.

4.0.1 Linear MPC regulator

[83] introduces a simple MPC regulation problem to elaborate the basics of MPC and uses dynamic programming to solve the problem. The system model of the problem is:

$$\begin{aligned} \mathbf{x}^+ &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{x} \end{aligned} \quad (4.1)$$

where \mathbf{x}^+ is the next system states calculated from current inputs \mathbf{u} and current system states \mathbf{x} . The system states \mathbf{x} are directly measured and are supposed to be manipulated to the origin by the MPC regulator. Considering N time steps into the future, the sequence of the inputs that need to be optimized is:

$$\mathcal{U} = \{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{N-1}\} \quad (4.2)$$

Given \mathcal{U} and the current system state \mathbf{x}_0 , the system states for the next N time steps $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ can be forecast according to Equation 4.1. The objective function for the optimization is defined as:

$$f(\mathbf{x}_0, \mathcal{X}, \mathcal{U}) = \frac{1}{2} \sum_{k=0}^{N-1} [\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k] + \frac{1}{2} \mathbf{x}_N^T \mathbf{P}_f \mathbf{x}_N \quad (4.3)$$

subject to

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} \quad \text{for } k = 1, \dots, N$$

where \mathbf{Q} , \mathbf{R} and \mathbf{P}_f are diagonal matrices and denote cost weights for different kinds of cost. Larger values in \mathbf{Q} indicate that the regulator needs to drive the states to zero more quickly, larger values in \mathbf{R} mean that control moves should be relatively smaller and larger values in \mathbf{P}_f will lay more importance on checking whether the system states at the N -th time step are at the origin. These parameters need to be well-tuned and balanced, in order to achieve satisfying performance. Then, the following problem is formulated:

$$\min_{\mathcal{U}} f(\mathbf{x}_0, \mathcal{X}, \mathcal{U}) \quad (4.4)$$

Dynamic Programming Solution: [83] describes a dynamic programming method to solve the optimization problem in Equation 4.4. First Equation 4.3 is rewritten as:

$$f(\mathbf{x}_0, \mathcal{X}, \mathcal{U}) = \sum_{k=0}^{N-1} l(\mathbf{x}_k, \mathbf{u}_k) + l_N(\mathbf{x}_N) \quad (4.5)$$

subject to

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} \quad \text{for } k = 1, \dots, N$$

where $l(\mathbf{x}, \mathbf{u}) = (1/2)(\mathbf{x}^T \mathbf{Q}\mathbf{x} + \mathbf{u}^T \mathbf{R}\mathbf{u})$ and $l_N(\mathbf{x}) = (1/2)(\mathbf{x}^T \mathbf{P}_f \mathbf{x})$. As the first system state \mathbf{x}_0 is the current system state and is fixed, backward dynamic programming is performed and the first optimization is conducted on the last system state. Then, optimizations are proceeded in a reverse order. Based on Equation 4.5, the optimization problem in Equation 4.4 is rearranged to the following form:

$$\begin{aligned} \min_{\mathbf{u}_0, \mathbf{x}_1, \dots, \mathbf{u}_{N-2}, \mathbf{x}_{N-1}} & l(\mathbf{x}_0, \mathbf{u}_0) + \dots + l(\mathbf{x}_{N-2}, \mathbf{u}_{N-2}) + \\ & \min_{\mathbf{u}_{N-1}, \mathbf{x}_N} l(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) + l_N(\mathbf{x}_N) \end{aligned} \quad (4.6)$$

subject to

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} \quad \text{for } k = 1, \dots, N$$

Thus, the first optimization problem at the last time step is given by:

$$\begin{aligned} \min_{\mathbf{u}_{N-1}, \mathbf{x}_N} & l(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) + l_N(\mathbf{x}_N) \\ \text{subject to} & \\ & \mathbf{x}_N = \mathbf{A}\mathbf{x}_{N-1} + \mathbf{B}\mathbf{u}_{N-1} \end{aligned} \quad (4.7)$$

where \mathbf{x}_N is treated only as a constant and optimal input \mathbf{u}_{N-1}^* can be analytically derived. Then, the optimal stage cost at the last time step is obtained as: $f_{N-1}^* = f_{N-1}(\mathbf{u}_{N-1}^*)$, the system state at the last time step is also derived: $\mathbf{x}_N = \mathbf{A}\mathbf{x}_{N-1} + \mathbf{B}\mathbf{u}_{N-1}^*$. Derivation of \mathbf{u}_{N-1}^* is as follows:

$$\begin{aligned} & l(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}) + l_N(\mathbf{x}_N) \\ & = (1/2) [\mathbf{x}_{N-1}^T \mathbf{Q}\mathbf{x}_{N-1} + \mathbf{u}_{N-1}^T \mathbf{R}\mathbf{u}_{N-1} + (\mathbf{A}\mathbf{x}_{N-1} + \mathbf{B}\mathbf{u}_{N-1})^T \mathbf{P}_f (\mathbf{A}\mathbf{x}_{N-1} + \mathbf{B}\mathbf{u}_{N-1})] \end{aligned} \quad (4.8)$$

Taking the derivative of Equation 4.8 with respect to \mathbf{u}_{N-1} and setting the derivative to 0 yields:

$$\begin{aligned} \mathbf{R}\mathbf{u}_{N-1} + \mathbf{B}^T \mathbf{P}_f (\mathbf{A}\mathbf{x}_{N-1} + \mathbf{B}\mathbf{u}_{N-1}) &\doteq \mathbf{0} \\ \Rightarrow (\mathbf{R} + \mathbf{B}^T \mathbf{P}_f \mathbf{B})\mathbf{u}_{N-1} + \mathbf{B}^T \mathbf{P}_f \mathbf{A}\mathbf{x}_{N-1} &= \mathbf{0} \\ \Rightarrow \mathbf{u}_{N-1} &= -(\mathbf{R} + \mathbf{B}^T \mathbf{P}_f \mathbf{B})^{-1} \mathbf{B}^T \mathbf{P}_f \mathbf{A}\mathbf{x}_{N-1} \end{aligned} \quad (4.9)$$

Thus the optimal input and optimal stage cost are given as follows:

$$\begin{aligned} \mathbf{u}_{N-1}^* &= \mathbf{K}_{N-1} \mathbf{x}_{N-1} \\ f_{N-1}^* &= (1/2) \mathbf{x}_{N-1}^T \mathbf{\Pi}_{N-1} \mathbf{x}_{N-1} \end{aligned} \quad (4.10)$$

where \mathbf{K}_{N-1} and $\mathbf{\Pi}_{N-1}$ are defined as follows:

$$\begin{aligned} \mathbf{K}_{N-1} &\doteq -(\mathbf{R} + \mathbf{B}^T \mathbf{P}_f \mathbf{B})^{-1} \mathbf{B}^T \mathbf{P}_f \mathbf{A} \\ \mathbf{\Pi}_{N-1} &\doteq \mathbf{Q} + \mathbf{A}^T \mathbf{P}_f \mathbf{A} + \mathbf{A}^T \mathbf{P}_f \mathbf{B} \mathbf{K}_{N-1} \\ &= \mathbf{Q} + \mathbf{A}^T \mathbf{P}_f \mathbf{A} - \mathbf{A}^T \mathbf{P}_f \mathbf{B} (\mathbf{R} + \mathbf{B}^T \mathbf{P}_f \mathbf{B})^{-1} \mathbf{B}^T \mathbf{P}_f \mathbf{A} \end{aligned} \quad (4.11)$$

\mathbf{u}_{N-1}^* and f_{N-1}^* are functions of \mathbf{x}_{N-1} and the optimization problem for the last but one time step is given as:

$$\begin{aligned} \min_{\mathbf{u}_{N-2}, \mathbf{x}_{N-1}} \quad & l(\mathbf{x}_{N-2}, \mathbf{u}_{N-2}) + f_{N-1}^*(\mathbf{x}_{N-1}) \\ \text{subject to} \quad & \\ \mathbf{x}_{N-1} &= \mathbf{A}\mathbf{x}_{N-2} + \mathbf{B}\mathbf{u}_{N-2} \end{aligned} \quad (4.12)$$

This problem has the same structure as the problem formulation in Equation 4.7 and the optimal solution \mathbf{u}_{N-1}^* and f_{N-1}^* can be derived analogously:

$$\begin{aligned} \mathbf{u}_{N-2}^* &= \mathbf{K}_{N-2} \mathbf{x}_{N-2} \\ f_{N-2}^* &= (1/2) \mathbf{x}_{N-2}^T \mathbf{\Pi}_{N-2} \mathbf{x}_{N-2} \end{aligned} \quad (4.13)$$

with

$$\begin{aligned} \mathbf{K}_{N-2} &\doteq -(\mathbf{R} + \mathbf{B}^T \mathbf{\Pi}_{N-1} \mathbf{B})^{-1} \mathbf{B}^T \mathbf{\Pi}_{N-1} \mathbf{A} \\ \mathbf{\Pi}_{N-2} &\doteq \mathbf{Q} + \mathbf{A}^T \mathbf{\Pi}_{N-1} \mathbf{A} + \mathbf{A}^T \mathbf{\Pi}_{N-1} \mathbf{B} \mathbf{K}_{N-2} \\ &= \mathbf{Q} + \mathbf{A}^T \mathbf{\Pi}_{N-1} \mathbf{A} - \mathbf{A}^T \mathbf{\Pi}_{N-1} \mathbf{B} (\mathbf{R} + \mathbf{B}^T \mathbf{\Pi}_{N-1} \mathbf{B})^{-1} \mathbf{B}^T \mathbf{\Pi}_{N-1} \mathbf{A} \end{aligned} \quad (4.14)$$

To summarize for all time steps, given $\mathbf{\Pi}_k$ at k -th time step, $\mathbf{\Pi}_{k-1}$ at $(k-1)$ -th time step is obtained as:

$$\begin{aligned} \mathbf{\Pi}_{k-1} &= \mathbf{Q} + \mathbf{A}^T \mathbf{\Pi}_k \mathbf{A} - \mathbf{A}^T \mathbf{\Pi}_k \mathbf{B} (\mathbf{R} + \mathbf{B}^T \mathbf{\Pi}_k \mathbf{B})^{-1} \mathbf{B}^T \mathbf{\Pi}_k \mathbf{A} \\ \text{for } k &= N, N-1, \dots, 1 \end{aligned} \quad (4.15)$$

with a terminal condition $\mathbf{\Pi}_N = \mathbf{P}_f$. And the optimal input and cost value are calculated as:

$$\begin{aligned} \mathbf{u}_k^* &= \mathbf{K}_k \mathbf{x}_k \\ f_k^* &= (1/2) \mathbf{x}_k^T \mathbf{\Pi}_k \mathbf{x}_k \\ \mathbf{K}_k &\doteq -(\mathbf{R} + \mathbf{B}^T \mathbf{\Pi}_{k+1} \mathbf{B})^{-1} \mathbf{B}^T \mathbf{\Pi}_{k+1} \mathbf{A} \end{aligned} \quad (4.16)$$

Recursive calculations of Equation 4.15 and Equation 4.16 end until $\mathbf{u}_0^* = \mathbf{K}_0 \mathbf{x}_0$ is obtained.

Batch Solution. Aside from recursive dynamic programming, Equation 4.4 can also be directly solved by considering the cost function to be solely dependent on an input vector $\mathbf{U} = [\mathbf{u}_0^T, \mathbf{u}_1^T, \dots, \mathbf{u}_{N-1}^T]^T \in R^{N \cdot m}$. The optimization is then performed on \mathbf{U} once to calculate the optimal input \mathbf{U}^* [84]. System states at the next N time steps are collected in a vector $\mathbf{X} = [\mathbf{x}_0^T, \mathbf{x}_1^T, \dots, \mathbf{x}_N^T]^T \in R^{(N+1) \cdot n}$. \mathbf{X} is determined by the following equation:

$$\underbrace{\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} \mathbf{I}_n \\ \mathbf{A} \\ \vdots \\ \mathbf{A}^N \end{bmatrix}}_{\mathbf{S}_X} \mathbf{x}_0 + \underbrace{\begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \mathbf{B} & 0 & \cdots & 0 \\ \mathbf{AB} & \mathbf{B} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{A}^{N-1}\mathbf{B} & \cdots & \cdots & \mathbf{B} \end{bmatrix}}_{\mathbf{S}_U} \underbrace{\begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix}}_{\mathbf{U}} \quad (4.17)$$

In this way, the calculation of the sequence of the system states is written in a shorter form:

$$\mathbf{X} = \mathbf{S}_X \mathbf{x}_0 + \mathbf{S}_U \mathbf{U} \quad (4.18)$$

The original cost function in the form of summation of stage costs can also be expressed in one equation:

$$f(\mathbf{x}_0, \mathbf{U}) = \mathbf{X}^T \bar{\mathbf{Q}} \mathbf{X} + \mathbf{U}^T \bar{\mathbf{R}} \mathbf{U} \quad (4.19)$$

with $\bar{\mathbf{Q}} = \text{diag}(\mathbf{Q}, \dots, \mathbf{Q}, \mathbf{P}_f)$ and $\bar{\mathbf{R}} = \text{diag}(\mathbf{R}, \dots, \mathbf{R})$. Substituting Equation 4.18 into Equation 4.19, taking derivative of the cost function with respect to \mathbf{U} and setting the derivative to 0, we have:

$$\begin{aligned} 2\mathbf{S}_U^T \bar{\mathbf{Q}} (\mathbf{S}_X \mathbf{x}_0 + \mathbf{S}_U \mathbf{U}) + 2\bar{\mathbf{R}} \mathbf{U} &\doteq \mathbf{0} \\ \Rightarrow \mathbf{U} &= -(\mathbf{S}_U^T \bar{\mathbf{Q}} \mathbf{S}_U + \bar{\mathbf{R}})^{-1} \mathbf{S}_U^T \bar{\mathbf{Q}} \mathbf{S}_X \mathbf{x}_0 \end{aligned} \quad (4.20)$$

The sequence of the optimal inputs \mathbf{U}^* is derived and solely depends on the initial system state \mathbf{x}_0 .

Simulations are conducted by using both dynamic programming-based MPC regulator and Batch-based MPC regulator. Given a discrete LTI system model with the same form of Equation 4.1, matrices \mathbf{A} and \mathbf{B} are defined as:

$$\mathbf{A} = \begin{bmatrix} 1.5 & -2.13 & 0.78 \\ 1.2 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0.5 \\ 0 \\ 0 \end{bmatrix} \quad (4.21)$$

And cost weight matrices are chosen as:

$$\mathbf{Q} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}, \mathbf{P}_f = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, R = 20 \quad (4.22)$$

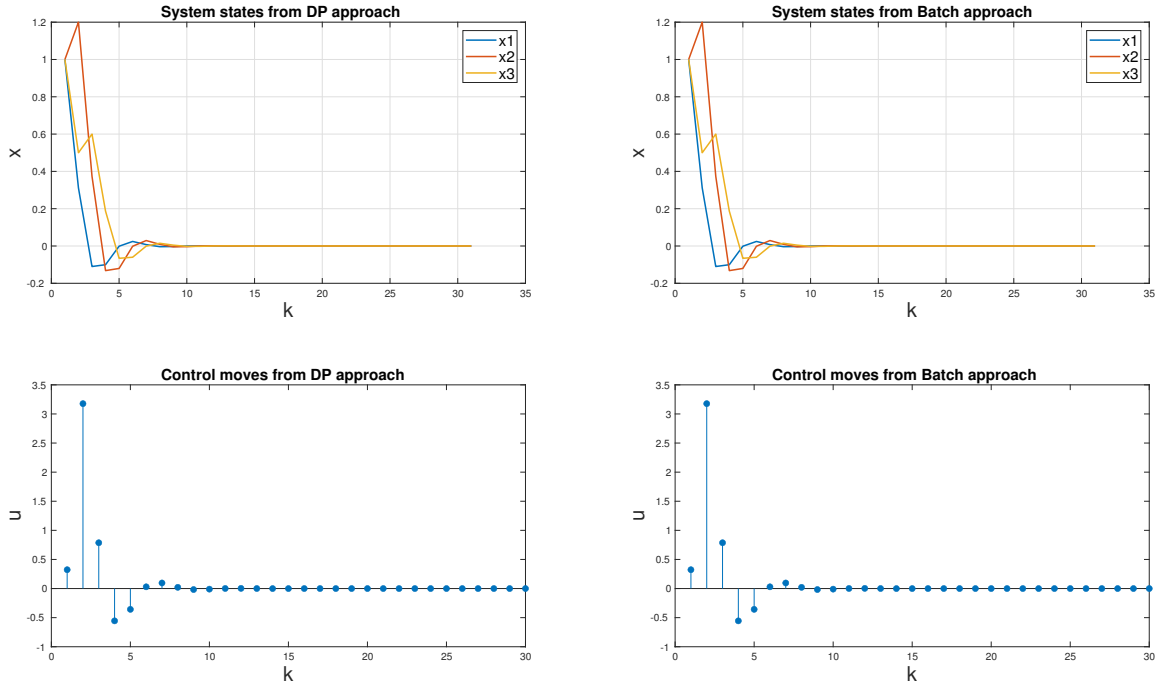


Figure 4.2: Linear MPC regulator DP vs. Batch approach

The MPC regulator of both approaches can be analytically derived and the evolution of the system states and optimal inputs is illustrated in the Figure 4.2. As shown in the figure, the two different approaches (dynamic programming-based MPC regulator and Batch-based MPC regulator) generated exactly the same control moves and system behavior.

4.1 Forward kinematics and Jacobin for robot in real world

In this dissertation, the Kinova[®] Jaco2 7Dof version manipulator is used in the experiments. For the manipulator system, the system states are defined as seven joint angles: $\mathbf{q} = [q_1, q_2, q_3, q_4, q_5, q_6, q_7]^T$ and the system input is the joint velocities sent to the joints: $\mathbf{u} = [u_1, u_2, u_3, u_4, u_5, u_6, u_7]^T$. Thus the system model is written as:

$$\begin{aligned} \mathbf{q}^+ &= \mathbf{q} + \mathbf{u} * t_s \\ \mathbf{y} &= \mathbf{q} \end{aligned} \quad (4.23)$$

However, tasks are often defined in cartesian space. Therefore, a FK that maps the points in joint space to cartesian space is necessary. Besides, a Jacobian matrix is derived for the FK.

4.1.1 Forward kinematics

The Jaco manipulator is composed of eight links connected by seven joints. We follow the convention from [85] and number the links from zero and number the joints from 1. Link 7 is

the end-effector and Link 0 is the fixed manipulator base. All joints are revolute joints, which means they can only rotate around a single axis and can not move along an axis. A joint angle q_n indicates rotating joint n by q_n . A local coordinate frame is attached to each link, i.e. $O_i x_i y_i z_i$ is attached to link i . A link will stay stationary in its own coordinate frame. Thus, the position of O_i and directions of x_i, y_i, z_i can uniquely describe the position and orientation of link i . Denavit–Hartenberg table is used to represent the deviation of a coordinate frame of link i with respect to the coordinate frame of its parent link $(i - 1)$. There are mainly two conventions in forming the DH table. One is distal convention, where coordinate frame $O_i x_i y_i z_i$ is put on the rotation axis of joint $i + 1$ and the transformation between two frames is given by the following operations order:

$$\mathbf{H}_{i-1}^i = \text{Rot}_{z_{i-1}}(\theta_i) \cdot \text{Trans}_{z_{i-1}}(d_i) \cdot \text{Trans}_{x_i}(a_i) \cdot \text{Rot}_{x_i}(\alpha_i) \quad (4.24)$$

Another is the proximal convention, where $O_i x_i y_i z_i$ is put on the rotation axis of joint i and the transformation calculation is in the following order:

$$\mathbf{H}_{i-1}^i = \text{Rot}_{x_{i-1}}(\alpha_i) \cdot \text{Trans}_{x_{i-1}}(a_i) \cdot \text{Trans}_{z_i}(d_i) \cdot \text{Rot}_{z_i}(\theta_i) \quad (4.25)$$

In this dissertation, the distal convention is used. The DH table of Jaco2 provided by [86] is in proximal convention and is modified to Table 4.1 in distal convention. Four values in the i -th row of the table indicate that parent coordinate frame $O_{i-1} x_{i-1} y_{i-1} z_{i-1}$ first rotates by θ_i about its z -axis and then, translates by d_i along the newly rotated x -axis and in the end rotating α_i about the axis. These operations yield the child coordinate frame $O_i x_i y_i z_i$. Figure 4.3 illustrates how DH parameters describe the deviation between two neighboring frames. A homogeneous transformation matrix is used to define such deviation:

$$\mathbf{H}_{i-1}^i = \begin{bmatrix} \mathbf{R}_{i-1}^i & \mathbf{T}_{i-1}^i \\ \mathbf{0} & 1 \end{bmatrix} \quad (4.26)$$

where $\mathbf{R}_i \in R^{3 \times 3}$ denotes the rotation matrix and $\mathbf{T}_i \in R^3$ the translation vector.

Table 4.1: DH table for Jaco2

i	θ_i	d_i	a_i	α_i
1	$q_1 + \pi$	-0.2755	0	$\pi/2$
2	q_2	0	0	$\pi/2$
3	q_3	-0.41	0	$\pi/2$
4	q_4	-0.0098	0	$\pi/2$
5	q_5	-0.3111	0	$\pi/2$
6	$q_6 + \pi$	0	0	$-\pi/2$
7	$q_7 + \pi/2$	-0.2638	0	π

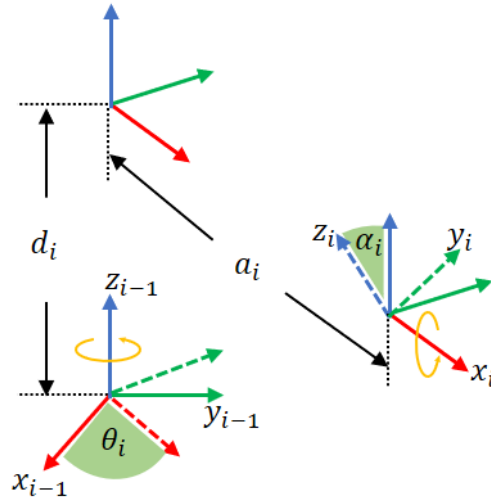


Figure 4.3: Example of DH in the distal convention

And H_{i-1}^i is calculated from DH table parameters:

$$\begin{aligned}
 H_{i-1}^i &= \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & 0 \\ \sin \theta_i & \cos \theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos \alpha_i & -\sin \alpha_i & 0 \\ 0 & \sin \alpha_i & \cos \alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} \cos \theta_i & -\sin \theta_i \cos \alpha_i & \sin \theta_i \sin \alpha_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cos \alpha_i & -\cos \theta_i \sin \alpha_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.27)
 \end{aligned}$$

Now the transformation between $O_i x_i y_i z_i$ and $O_0 x_0 y_0 z_0$ can be expressed as a function of joint values \mathbf{q} by chaining local transformation between neighboring links one by one, i.e. the transformation of link i with respect to the base frame can be calculated given \mathbf{q} :

$$H_0^i(\mathbf{q}) = H_{i-1}^i(q_i) \cdot H_{i-2}^{i-1}(q_{i-1}) \cdots H_0^1(q_1) \quad (4.28)$$

4.1.2 Orientation representation

It is impractical to use the rotation matrix in controlling the robot to the desired orientation because there are too many elements that need to be steered. Besides, these elements are not independent from each other. Actually, an orientation can be well represented by at most three quantities.

Roll-pitch-yaw, an arbitrary rotation can be decomposed into three successive rotations around fixed coordinate frame axes x_0, y_0, z_0 . Roll-pitch-yaw rotation is taken in a specific

order and in this dissertation, we perform the rotation in order x-y-z, i.e. first a roll about x_0 axis by an angle of ψ then a pitch about y_0 axis by an angle of θ and in the end a yaw about z_0 axis by an angle of ϕ . Given roll, pitch, yaw, a rotation matrix is derived:

$$\begin{aligned} \mathbf{R}_{xyz} &= \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \psi & -\sin \psi \\ 0 & \sin \psi & \cos \psi \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta \cos \psi + \sin \theta \sin \psi & \sin \theta \cos \psi + \cos \theta \sin \psi & \sin \psi \\ \sin \theta \cos \psi - \cos \theta \sin \psi & \cos \theta \cos \psi - \sin \theta \sin \psi & \cos \psi \\ -\sin \theta & \cos \theta & 0 \end{bmatrix} \end{aligned} \quad (4.29)$$

Since rotations are performed around fixed axes, rotation matrices of successive decomposed rotations are premultiplied. Reversely, given a rotation matrix, roll, pitch and yaw angles can also be determined based on [Equation 4.29](#).

In the next section, we will discuss how to minimize error between the current and desired orientations of the end-effectors for our MPC controller. The MPC controller can be viewed as a feedback controller. [87] pointed out that, despite its convenience in describing orientation intuitively, roll-pitch-yaw representation is undesirable in feedback control due to singularities and computational complexity. And the differential relationship between the rates of change of roll-pitch-yaw angles and angular velocities is highly nonlinear, leading to unpredictable behavior when steering roll-pitch-yaw angles.

Axis/Angle, any rotation can be considered as a rotation around an arbitrary axis in space, where only three quantities are used as well. This representation provides a concise way to describe a rotation in space and is crucial for the geometrical derivation of the robot's Jacobian later because angular velocities are defined around an axis in space. Given an arbitrary unit vector $\mathbf{k} = [k_x, k_y, k_z]^T$ in frame $O_0x_0y_0z_0$, the rotation matrix $\mathbf{R}_{\mathbf{k},\theta}$ for a rotation of θ around axis \mathbf{k} is given [85] by:

$$\mathbf{R}_{\mathbf{k},\theta} = \begin{bmatrix} k_x^2 v_\theta + \cos \theta & k_x k_y v_\theta - k_z \sin \theta & k_x k_z v_\theta + k_y \sin \theta \\ k_x k_y v_\theta + k_z \sin \theta & k_y^2 v_\theta + \cos \theta & k_y k_z v_\theta - k_x \sin \theta \\ k_x k_z v_\theta - k_y \sin \theta & k_y k_z v_\theta + k_x \sin \theta & k_z^2 v_\theta + \cos \theta \end{bmatrix} \quad (4.30)$$

where $v_\theta = \text{vers}\theta = 1 - \cos \theta$.

However, this representation is not intuitive and is complicated to derive. Thus it can not be used in defining certain orientation requirements for a task. Besides, neither roll-pitch-yaw nor axis/angle representations are unique for a given rotation. For example, a rotation of $-\theta$ about $-\mathbf{k}$ is the same as a rotation of θ about \mathbf{k} [85].

Quaternion, if the rotation between two coordinate frames $O_0x_0y_0z_0$ and $O_1x_1y_1z_1$ are represented in axis/angle form, i.e., a rotation of θ about a unit vector \mathbf{k} in $O_0x_0y_0z_0$. The

quaternion representation of the two coordinate frames are defined as follows [87]:

$$w \doteq \cos \theta/2, \quad \mathbf{p} \doteq (\sin \theta/2)\mathbf{k} \quad (4.31)$$

Then, $[w, \mathbf{p}]^T$ is the quaternion of $O_1x_1y_1z_1$ with respect to $O_0x_0y_0z_0$. Quaternion is normal because:

$$w^2 + \mathbf{p}^T \mathbf{p} = (\cos \theta/2)^2 + (\sin \theta/2)^2 \mathbf{k}^T \mathbf{k} = (\cos \theta/2)^2 + (\sin \theta/2)^2 = 1 \quad (4.32)$$

And quaternion is also unique as long as θ is confined to $[-180^\circ, 180^\circ]$, which is often the case.

Given a quaternion $\boldsymbol{\xi} = [w, \mathbf{q}^T]^T$, the rotation matrix can be derived:

$$\mathbf{R} = (w^2 - \mathbf{p}^T \mathbf{p})\mathbf{I}_3 + 2\mathbf{p}^T - 2w\mathbf{p}^\times \quad (4.33)$$

where \mathbf{I}_3 is a 3×3 identity matrix and \mathbf{p}^\times denotes the skew-symmetric matrix for vector \mathbf{p} and

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \mathbf{p}^\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \quad (4.34)$$

Quaternion is a good choice for orientation representation in feedback control and will be used in the optimization of our MPC controller.

4.1.3 Jacobian

Basic Jacobian describes the relationship between joint velocities and linear, angular velocities in cartesian space. Since the joints of the robot manipulator used in this dissertation are all of revolute type, only Jacobian for revolute joints is discussed.

As shown in Figure 4.4, geometrically, the rotation of a revolute joint produces exactly the same rotation of the end-effector around that joint, which means a scalar joint velocity \dot{q} causes on the end-effector an angular velocity $\boldsymbol{\omega}$ whose magnitude is $\dot{\theta}$ and direction is aligned with the rotation axis of the joint. The rotation of a joint also produces a linear velocity \mathbf{v} . \mathbf{v} 's direction should be tangent to the circle around the joint at the position of the end-effector. \mathbf{v} is perpendicular to the rotating axis and $\boldsymbol{\omega}$, as the rotating axis is perpendicular to the plane where \mathbf{v} lies on. And obviously, \mathbf{v} is also perpendicular to the circle radius \mathbf{r} pointing to the position of the end-effector. Furthermore, the magnitude of \mathbf{v} is given by:

$$\|\mathbf{v}\| = \dot{\theta} \|\mathbf{r}\| = \|\boldsymbol{\omega}\| \|\mathbf{v}\| = \|\boldsymbol{\omega}\| \|\mathbf{v}\| \sin \beta \quad (4.35)$$

β indicates the angle between \mathbf{r} and $\boldsymbol{\omega}$ and it is 90° as \mathbf{r} and $\boldsymbol{\omega}$ are also perpendicular to each other. Thus $\sin \beta = 1$. The direction and magnitude of \mathbf{v} confirm that it is the cross product of $\boldsymbol{\omega}$ and \mathbf{r} . Therefore, a joint velocity produces on the end-effector an angular velocity $\boldsymbol{\omega}$ and a linear velocity $\mathbf{v} = \boldsymbol{\omega} \times \mathbf{r}$, where \mathbf{r} denotes a vector from the position of the joint to the

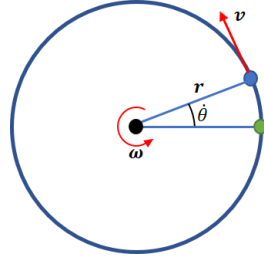


Figure 4.4: Linear and angular velocity from a single joint

end-effector. We can see that the magnitudes of v and ω are proportional to the joint velocity \dot{q} and their relationship is written as:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \underbrace{\begin{bmatrix} z \times r \\ z \end{bmatrix}}_{J_i} \dot{q} \quad (4.36)$$

where z denotes the unit rotating axis vector for the joint and $J_i \in R^6$ is one column contributed by this single joint for the whole Jacobian matrix. Since z and r both depend on joint angles q , so does J_i .

The linear and angular velocities produced by multiple joints are simply added together, yielding a total linear and angular velocities ω_{ef} and v_{ef} on the end-effector:

$$\begin{bmatrix} v_{ef} \\ \omega_{ef} \end{bmatrix} = J_1 \dot{q}_1 + J_2 \dot{q}_2 + \cdots + J_n \dot{q}_n = \underbrace{[J_1, J_2, \cdots, J_n]}_J \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} \quad (4.37)$$

where n indicates the number of joints and $J \in R^{6 \times n}$ is the Jacobian matrix for the end-effector with respect to n joints. J is also a function of joint angles q .

Joint i connects link i and link $(i - 1)$ and local coordinate frame $O_{i-1}x_{i-1}y_{i-1}z_{i-1}$ is attached to link $(i - 1)$. Distal DH convention always sets the origin of $O_{i-1}x_{i-1}y_{i-1}z_{i-1}$ to where joint i is located and sets the z -axis to be aligned with the rotating axis of joint i . Therefore, once the transformation $H_0^{i-1} = [R_0^{i-1}, T_0^{i-1}; \mathbf{0}, \mathbf{1}]$ of $O_{i-1}x_{i-1}y_{i-1}z_{i-1}$ is given, z_i is determined as the third column of R_0^{i-1} and r_i is determined as the end-effector's translation subtracting T_0^{i-1} :

$$\begin{aligned} R_0^{i-1} &= \begin{bmatrix} R_1 & R_2 & R_3 \end{bmatrix}, \quad z_i = R_3 \\ r_i &= T_0^{ef} - T_0^{i-1} \end{aligned} \quad (4.38)$$

And the Jacobian matrix is given by:

$$\mathbf{J} = \begin{bmatrix} \mathbf{z}_1 \times \mathbf{r}_1 & \mathbf{z}_2 \times \mathbf{r}_2 & \cdots & \mathbf{z}_n \times \mathbf{r}_n \\ \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_n \end{bmatrix} \quad (4.39)$$

Jacobian for quaternion: Let \mathbf{J}_ω denote the bottom three rows of \mathbf{J} . \mathbf{J}_ω describes the relationship between joint velocities $\dot{\mathbf{q}}$ and angular velocities $\boldsymbol{\omega}$ of the end-effector. In this dissertation, we use quaternion $\boldsymbol{\xi} = [\eta, \mathbf{p}^T]^T$ to represent the orientation. Thus instead of \mathbf{J}_ω for $\boldsymbol{\omega}$, a Jacobian \mathbf{J}_ξ is needed for the rate of change on quaternion. [87] gave a quaternion propagation differential equation:

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} \dot{w} \\ \dot{\mathbf{p}} \end{bmatrix} = 1/2 \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -\boldsymbol{\omega}^\times \end{bmatrix} \begin{bmatrix} w \\ \mathbf{p} \end{bmatrix} \quad (4.40)$$

Substituting $\boldsymbol{\omega} = \mathbf{J}_\omega \dot{\mathbf{q}}$ into Equation 4.40:

$$\begin{aligned} \dot{\boldsymbol{\xi}} &= 1/2 \begin{bmatrix} -\boldsymbol{\omega}^T \mathbf{p} \\ w\boldsymbol{\omega} - \boldsymbol{\omega}^\times \mathbf{p} \end{bmatrix} = 1/2 \begin{bmatrix} -\mathbf{p}^T \boldsymbol{\omega} \\ w\boldsymbol{\omega} + \mathbf{p}^\times \boldsymbol{\omega} \end{bmatrix} = 1/2 \begin{bmatrix} -\mathbf{p}^T \mathbf{J}_\omega \dot{\mathbf{q}} \\ w\mathbf{J}_\omega \dot{\mathbf{q}} + \mathbf{p}^\times \mathbf{J}_\omega \dot{\mathbf{q}} \end{bmatrix} \\ &= 1/2 \begin{bmatrix} -\mathbf{p}^T \\ w\mathbf{I}_3 + \mathbf{p}^\times \end{bmatrix} \mathbf{J}_\omega \dot{\mathbf{q}} = 1/2 \underbrace{\begin{bmatrix} -x & -y & -z \\ w & -z & y \\ z & w & -x \\ -y & x & w \end{bmatrix}}_M \mathbf{J}_\omega \dot{\mathbf{q}} = \mathbf{J}_\xi \dot{\mathbf{q}} \end{aligned} \quad (4.41)$$

Therefore, given \mathbf{J}_ω and current quaternion $\boldsymbol{\xi} = [w, p_1, p_2, p_3]^T$, $\mathbf{J}_\xi \in R^{4 \times n}$ can be calculated by $\mathbf{J}_\xi = 1/2 \cdot M \mathbf{J}_\omega$.

4.2 Model predictive control for motion planning

Here in this section will extend the MPC to design a motion planner for Jaco2 manipulator.

Consider the linear MPC regulator discussed in Section 4.1, the goal is defined as $\mathbf{0}$ in joint space. Given the system model Equation 4.1, an optimization function is formulated to find the optimal system inputs for the next N time steps into the future. This sequence of inputs is supposed to minimize an objective function. The objective function is defined as an accumulation of the quadratic error between $\mathbf{0}$ and the actual system state for the next N time steps plus a regularization on the input sequence, as Equation 4.3 states. Equation 4.19 shows that, with the current system state \mathbf{x}_0 known, the objective function is given as a quadratic function of input sequence \mathbf{U} . And the optimal solution \mathbf{U}^* can be analytically derived.

For a general MPC, a prediction horizon $ph \in \mathbb{N}^+$ defines the number of future time steps that need to be predicted and control horizon $ch \in \mathbb{N}^+ \leq ph$ defines the number

of system inputs in the predicted future. For time steps after ch and before ph , the input for the ch -th time step is repeated. Thus, the sequence of inputs for the system becomes $\{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{ch-1}, \mathbf{u}_{ch-1}, \dots, \mathbf{u}_{ch-1}\}$ and the input variable for the optimization problem is $\mathbf{U} = [\mathbf{u}_0^T, \mathbf{u}_1^T, \dots, \mathbf{u}_{ch-1}^T]^T$. The dimension of the optimization is reduced from $ph \times m$ to $ch \times m$, where m is the number of the joints of the manipulator.

Now consider the robot manipulator motion planning problems, the system model is written as:

$$\begin{aligned} \mathbf{q}^+ &= \mathbf{q} + \mathbf{u} \cdot t_s \\ \mathbf{X} &= \mathcal{F}(\mathbf{q}) \end{aligned} \quad (4.42)$$

where $\mathbf{q} \in R^7$ denotes joint positions on the 7-Dof manipulator, $\mathbf{u} \in R^7$ is the manipulated joint velocities. System output $\mathbf{X} \in R^7$ represents the three-dimensional position and four-dimensional quaternion of the end-effector in cartesian space. \mathcal{F} describes the nonlinear relationship between \mathbf{X} and \mathbf{q} . As discussed in Section 4.2, a homogeneous transformation matrix $\mathbf{H}_0^i(\mathbf{q}) = [\mathbf{R}_0^i(\mathbf{q}), \mathbf{T}_0^i(\mathbf{q}), \mathbf{0}, 1]$ for robot link i can be calculated via robot's forward kinematics, from which \mathbf{X} is determined as:

$$\mathbf{X} = \begin{bmatrix} \mathbf{T}_0^i(\mathbf{q}) \\ \mathcal{G}(\mathbf{R}_0^i(\mathbf{q})) \end{bmatrix} \quad (4.43)$$

Here \mathcal{G} is implemented as a function extracting quaternion from a rotation matrix. [88]'s method remains most popular for calculating quaternion from a rotation matrix. Equation 4.43 shows that the function \mathcal{F} is highly nonlinear.

4.2.1 Optimization problem formulation

Given a cartesian goal \mathbf{X}_d , the optimization problem for generating an optimal sequence of joint velocities is formulated as follows:

$$\begin{aligned} & \min_{\mathbf{U}, \mathbf{Q}} \left\{ \sum_{i=0}^{ph-1} [e(\mathbf{X}_d, \mathbf{X}_i)^T \mathbf{Q} e(\mathbf{X}_d, \mathbf{X}_i) + \mathbf{u}_i^T \mathbf{R} \mathbf{u}_i] + e(\mathbf{X}_d, \mathbf{X}_{ph})^T \mathbf{P}_f e(\mathbf{X}_d, \mathbf{X}_{ph}) \right\} \\ & \text{subject to} \\ & \quad \mathbf{lb} \leq \mathbf{U} \leq \mathbf{ub}, \\ & \quad \mathbf{g}(\mathbf{q}_i) \leq 0, \quad \text{for } i = 1, 2, \dots, ph \\ & \quad \mathbf{q}_i = \mathbf{q}_{i-1} + \mathbf{u}_{i-1} \cdot T_s, \quad \text{for } i = 1, 2, \dots, ch \\ & \quad \mathbf{q}_i = \mathbf{q}_{i-1} + \mathbf{u}_{ch-1} \cdot T_s, \quad \text{for } i = ch + 1, \dots, ph \end{aligned} \quad (4.44)$$

where $e(\mathbf{X}_d, \mathbf{X}_i)$ is an error metric that measures the difference between \mathbf{X}_d and \mathbf{X}_i . For position, the error can be simply defined as the subtraction between desired position \mathbf{T}_d and current position \mathbf{T}_i . For quaternion difference between $\boldsymbol{\xi}_d = [w_d, \underbrace{x_d, y_d, z_d}_{\mathbf{p}_d^T}]^T$ and

$\xi_i = [w_i, \underbrace{x_i, y_i, z_i}_{\mathbf{p}_i^T}]^T$, [87] introduced the following orientation error representation to define a relative quaternion rotation, under which ξ_i is rotated to ξ_d :

$$\Delta\xi = \begin{bmatrix} \Delta w \\ \Delta\mathbf{p} \end{bmatrix} = \begin{bmatrix} w_d w_i + \mathbf{p}_d^T \mathbf{p}_i \\ w_d \mathbf{p}_i - w_i \mathbf{p}_d - \mathbf{p}_d^\times \mathbf{p}_i \end{bmatrix} \quad (4.45)$$

where the skew-symmetric matrix \mathbf{p}_d^\times can be expressed similarly to Equation 4.34. [87] states that, $\Delta\mathbf{p}$ alone can be a good representation for orientation error between two coordinate frames, because two coordinate systems coincide if and only if $\Delta\mathbf{p} = \mathbf{0}$. Furthermore, also adding Δw into the feedback control always leads to a wrong convergence as $\Delta w = 0$ happens only when the actual and desired orientation are separated by a Euler rotation of 180° . Therefore, we choose the quaternion error as $\Delta\xi = \Delta\mathbf{p} \in R^3$ and $\Delta\xi$ can be written in an affine form with respect to ξ_i :

$$\begin{aligned} \Delta\xi = \Delta\mathbf{p} &= w_d \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} - w_i \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix} - \begin{bmatrix} 0 & -z_d & y_d \\ z_d & 0 & -x_d \\ -y_d & x_d & 0 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} -x_d & w_d & -z_d & y_d \\ -y_d & z_d & w_d & -x_d \\ -z_d & -y_d & x_d & w_d \end{bmatrix}}_{\mathbf{\Pi}_d} \xi_i \end{aligned} \quad (4.46)$$

where matrix $\mathbf{\Pi}_d$ is built from ξ_d and remains constant during the whole optimization.

Then, the error function $e(\mathbf{X}_d, \mathbf{X}_i)$ is given:

$$e(\mathbf{X}_d, \mathbf{X}_i) = \begin{bmatrix} \mathbf{T}_d - \mathbf{T}_i \\ \mathbf{\Pi}_d \xi_i \end{bmatrix} \quad (4.47)$$

$\mathbf{U} = [u_0^T, u_1^T, \dots, u_{ch-1}^T]^T$, $\mathbf{Q} = [q_0^T, q_1^T, \dots, q_{ph}^T]^T$ and $\mathbf{X}_i = \mathcal{F}(q_i)$. lb and ub denote the lower and upper limit of the joint velocities and $g(q_i)$ are defined as constraint functions regarding collision avoidance.

Problem Equation 4.44 is a constrained nonlinear optimization problem. We propose to use batch approach to solve the problem. Given current robot joint positions q_0 and an arbitrary hypothesis \mathbf{U} on the joint velocities sequence, the sequence of the joint positions in next ph time steps \mathbf{Q} is predicted according to the system model. The objective function and constraint functions in the problem Equation 4.44 are converted to be only dependent on \mathbf{U} .

4.2.2 Objective function

In order to reformulate the objective function in problem Equation 4.44, linear relationship between U and \mathcal{Q} is written in batched form:

$$\underbrace{\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \vdots \\ q_{ch} \\ q_{ch+1} \\ \vdots \\ q_{ph} \end{bmatrix}}_{\mathcal{Q} \in R^{7(ph+1)}} = t_s \cdot \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & N \end{bmatrix}}_{M \in R^{(ph+1) \times ch}} \otimes I_7 \cdot \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{ch-1} \end{bmatrix}}_{U \in R^{7ch}} + \underbrace{\begin{bmatrix} q_0 \\ q_0 \\ q_0 \\ \vdots \\ q_0 \\ q_0 \\ \vdots \\ q_0 \end{bmatrix}}_{\mathcal{Q}_0 \in R^{7(ph+1)}} \quad (4.48)$$

where $N = ph - ch + 1$. We can define $S_U = t_s \cdot M \otimes I_7$ and \otimes indicates a kronecker product. Thus, $S_U \in R^{7(ph+1) \times 7ch}$. And a simple version for Equation 4.48 is:

$$\mathcal{Q} = S_U U + \mathcal{Q}_0 \quad (4.49)$$

The FK for all ph time steps are wrapped in one function $\bar{\mathcal{F}}$, yielding a nonlinear relationship between the joint angles sequence \mathcal{Q} and the cartesian positions sequence of end-effector $\bar{\mathbf{X}} = [\mathbf{X}_0^T, \mathbf{X}_1^T, \dots, \mathbf{X}_{ph}^T]^T \in R^{7(ph+1)}$:

$$\underbrace{\begin{bmatrix} \mathbf{X}_0 \\ \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{ph} \end{bmatrix}}_{\bar{\mathbf{X}}} = \bar{\mathcal{F}}(\mathcal{Q}) = \begin{bmatrix} \mathcal{F}(q_0) \\ \mathcal{F}(q_1) \\ \vdots \\ \mathcal{F}(q_{ph}) \end{bmatrix} \quad (4.50)$$

The objective function in problem Equation 4.44 can be expressed as:

$$f(U) = \bar{e}(\mathbf{X}_d, \bar{\mathbf{X}})^T \bar{\mathcal{Q}} \bar{e}(\mathbf{X}_d, \bar{\mathbf{X}}) + U^T \bar{\mathbf{R}} U \quad (4.51)$$

where $\bar{\mathcal{Q}} = \text{diag}(\underbrace{\mathcal{Q}, \dots, \mathcal{Q}}_{ph}, \mathcal{P}_f) \in R^{6(ph+1) \times 6(ph+1)}$, $\bar{\mathbf{R}} = \text{diag}(\underbrace{\mathbf{R}, \dots, \mathbf{R}}_{ch-1}, N\mathbf{R}) \in R^{7ch \times 7ch}$

and $\bar{e}(\mathbf{X}_d, \bar{\mathbf{X}}) = [e(\mathbf{X}_d, \mathbf{X}_0)^T, e(\mathbf{X}_d, \mathbf{X}_1)^T, \dots, e(\mathbf{X}_d, \mathbf{X}_{ph})^T]^T$. Therefore, given the desired cartesian goal \mathbf{X}_d , current joint angles q_0 and a hypothesis on the joint velocities sequence U , the scalar value of the objective function can be calculated by Equation 4.51.

Gradient. The nonlinear optimization problem in Equation 4.44 can be solved by SQP. When no gradient information is provided to SQP, it will have to numerically approximate

gradients via finite differences, which is time-consuming for high-dimensional problems. Therefore, we derive the analytical form of the gradients of the objective function and constraints for Equation 4.44. Before calculating gradient of the objective function, the derivative of $\bar{\mathcal{F}}$ with respect to \mathcal{Q} is first calculated:

$$\frac{\partial \bar{\mathcal{F}}}{\partial \mathcal{Q}} = \begin{bmatrix} \frac{\partial \mathcal{F}_0}{\partial q_0} & \frac{\partial \mathcal{F}_0}{\partial q_1} & \dots & \frac{\partial \mathcal{F}_0}{\partial q_{ph}} \\ \frac{\partial \mathcal{F}_1}{\partial q_0} & \ddots & \ddots & \frac{\partial \mathcal{F}_1}{\partial q_{ph}} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial \mathcal{F}_{ph}}{\partial q_0} & \dots & \dots & \frac{\partial \mathcal{F}_{ph}}{\partial q_{ph}} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{J}_0 & & & \\ & \mathbf{J}_1 & & \\ & & \ddots & \\ & & & \mathbf{J}_{ph} \end{bmatrix}}_{\bar{\mathbf{J}} \in \mathbb{R}^{7(ph+1) \times 7(ph+1)}} \quad (4.52)$$

where $\mathcal{F}_i = \mathcal{F}(q_i) = X_i$ and $\mathbf{J}_i = \mathbf{J}(q_i)$ indicate the cartesian position and Jacobian of the end-effector at i -th time step respectively for $i = 0, 1, \dots, ph$. According to Equation 4.37, $\dot{X}_i = \mathbf{J}_i \dot{q}_i$, and we have:

$$\dot{X}_i = \frac{\partial \mathcal{F}_i}{\partial t} = \mathbf{J}_i \frac{\partial q_i}{\partial t} \Rightarrow \frac{\partial \mathcal{F}_i}{\partial q_i} = \mathbf{J}_i \quad (4.53)$$

Given a q_i at i -th time step, \mathcal{F}_i is solely determined by q_i and can be seen independent from q_s before i -th time step. Obviously, q_s after i -th time step don't affect \mathcal{F}_i , either. Therefore, we have:

$$\frac{\partial \mathcal{F}_i}{\partial q_j} = \begin{cases} \mathbf{J}_i, & \text{if } i = j \\ \mathbf{0}, & \text{otherwise} \end{cases} \quad \text{for } i, j = 0, 1, \dots, ph \quad (4.54)$$

And Equation 4.52 holds.

Second, similarly to the statement from Equation 4.54, $e(X_d, X_i)$ only depends on X_i and is independent from X before and after i -th time step. Taking the derivative of Equation 4.47 yields:

$$\frac{\partial e}{\partial X_i} = \underbrace{\begin{bmatrix} \mathbf{I}_3 \\ \mathbf{\Pi}_d \end{bmatrix}}_{\Gamma \in \mathbb{R}^{6 \times 7}} \quad (4.55)$$

The derivative of $\bar{e}(X_d, \bar{X})$ with respect to \bar{X} is given as: $\partial \bar{e} / \partial \bar{X} = \text{diag}(\underbrace{\Gamma, \dots, \Gamma}_{ph+1}) \doteq \bar{\Gamma} \in \mathbb{R}^{6(ph+1) \times 7(ph+1)}$.

In the end, we take the derivative of Equation 4.51 with respect to U :

$$\frac{\partial f}{\partial U} = 2\bar{e}(X_d, \bar{X})^T \bar{Q} \bar{\Gamma} \bar{J} S_U + 2U^T \bar{R} \quad (4.56)$$

Therefore, given U and q_0 , the gradient of the objective function at U can be calculated: $\nabla f = (\partial f / \partial U)^T$.

4.2.3 Constraints

In order to ensure that the robot will never collide with obstacles, hard constraints for obstacle avoidance are imposed. Traditional optimization-based motion planning methods integrate the proximity of the robot to obstacles into a potential function and attempt to minimize it along with the main task cost. Schulman's motion planning discarded the potential function representation for obstacles. Instead, it used the closest distance between a robot link and an obstacle to represent the proximity of a robot link to the obstacle [46]. However, this method does not consider the closest distance as a hard constraint but adds the hinge loss of the closest distance to the objective function and minimizes it. In this dissertation the closest distance between a robot link and an obstacle being larger than a safe threshold is imposed as a hard constraint to the optimization problem.

Then, the constraint function $\mathbf{g}(\mathbf{q}_i)$ for a single time step in Equation 4.44 is defined as $\mathbf{g}(\mathbf{q}_i) = [g_1^1(\mathbf{q}_i)^T, \dots, g_1^K(\mathbf{q}_i)^T, g_2^1(\mathbf{q}_i)^T, \dots, g_2^K(\mathbf{q}_i)^T, \dots, g_L^1(\mathbf{q}_i)^T, \dots, g_L^K(\mathbf{q}_i)^T]^T$, where $g_l^k(\mathbf{q}_i) = TH - sd_l^k(\mathbf{q}_i)$ states that in i -th time step, the signed distance between l -th obstacle and k -th robot collision object should be larger than a threshold TH , for $i = 1, \dots, ph$, $k = 1, \dots, K$ and $l = 1, \dots, L$, where K denotes the number of collision objects used to represent the whole robot manipulator and L denotes the number of separated obstacles. Here sd function is a collision detector that can calculate the distance between two objects given the transformations of two objects. It is positive when two objects are not in collision and negative otherwise. The transformation of the obstacle is detected and the transformation of robot's collision object is given by the FK of the robot. Hence, sd is a function of \mathbf{q}_i , and thus a function of \mathbf{U} . For the NMPC problem with ph prediction horizons, collisions in the next ph time steps are considered, yielding $ph \times K \times L$ obstacle constraints in total imposed on the optimization problem.

Aside from the obstacle constraints, the lower and upper bounds for \mathbf{U} are imposed on the joint velocities. All constraints for the optimization problem are written in the batch form:

$$\bar{\mathbf{g}}(\mathcal{Q}) = \begin{bmatrix} \mathbf{g}(\mathbf{q}_1) \\ \dots \\ \mathbf{g}(\mathbf{q}_{ph}) \\ \mathcal{Q} - \mathbf{ub} \\ \mathbf{lb} - \mathcal{Q} \end{bmatrix} \leq \mathbf{0} \quad (4.57)$$

Collision detection. We use the same collision detection method and proximity representation as those are used in Schulman's method [46], i.e. a signed distance is defined and detected to describe the proximity of the robot to an obstacle. Given two objects \mathcal{A} and \mathcal{B} and their transformations with respect to a global coordinate frame, the distance between them is defined as the length of the smallest translation, by which two shapes are moved to contact. Obviously, distance is zero if two objects are intersecting. A penetration depth is defined as the smallest translation that drives two shapes out of contact and penetration depth is zero when two shapes are not intersecting. At i -th time step, given current joint angles \mathbf{q}_i , the

transformation $\mathbf{H}_{\mathcal{A}}^i$ for a collision object \mathcal{A} of the robot is calculated by FK: $\mathbf{H}_{\mathcal{A}}^i = \mathcal{F}^{\mathcal{A}}(\mathbf{q}_i)$. It is also assumed that, the transformation $\mathbf{H}_{\mathcal{B}}^i$ of an obstacle \mathcal{B} can be obtained and remains unchanged for $i = 1, \dots, ph$. The distance between two objects can be calculated by the GJK algorithm [89]. The algorithm first detects whether two objects are in collision, if not, distance along with the closest points $\mathbf{p}_{\mathcal{A}}$ and $\mathbf{p}_{\mathcal{B}}$ on each object are calculated. If two objects are in collision, EPA algorithm [90] is used to calculate the deepest penetration depth and the contacting points. Signed distance is expressed as:

$$sd(\mathbf{q}_i) = \begin{cases} \text{GJK}(\mathcal{A}, \mathbf{H}_{\mathcal{A}}^i, \mathcal{B}, \mathbf{H}_{\mathcal{B}}^i) & \text{if } \mathcal{A} \text{ and } \mathcal{B} \text{ are not in collision} \\ \text{EPA}(\mathcal{A}, \mathbf{H}_{\mathcal{A}}^i, \mathcal{B}, \mathbf{H}_{\mathcal{B}}^i) & \text{otherwise} \end{cases} \quad (4.58)$$

Figure 4.5 shows the signed distance and the closest or contacting points for both cases.

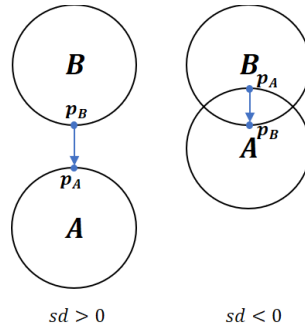


Figure 4.5: Example of signed distance

Gradient. The analytical expression for the gradient of the constraints is also necessary for the optimization. Before deriving the constraints gradient, there are two points to highlight. First, similarly to $\partial \mathcal{F}(\mathbf{q}_i)/\partial \mathbf{q}_j$, we have $\partial g(\mathbf{q}_i)/\partial \mathbf{q}_j = -\partial sd(\mathbf{q}_i)/\partial \mathbf{q}_j$ if $i = j$ and $\partial g(\mathbf{q}_i)/\partial \mathbf{q}_j = \mathbf{0}$ otherwise.

Second, collision detector outputs the positions \mathbf{p}_k^r and \mathbf{p}_l^o of either the closest or contacting points for k -th robot collision object and l -th obstacle collision object, respectively. Hence, the signed distance function can also be expressed as:

$$sd_l^k(\mathbf{q}_i) = s_l^k \sqrt{(\mathbf{p}_k^r - \mathbf{p}_l^o)^T (\mathbf{p}_k^r - \mathbf{p}_l^o)} \quad (4.59)$$

where $s_l^k = 1$ if object l and k are not in collision at i -th time step and $s_l^k = -1$ otherwise. In this way, according to [46], derivative of sd function can be approximated:

$$\frac{\partial sd_l^k(\mathbf{q}_i)}{\partial \mathbf{q}_i} \approx \frac{\mathbf{p}_k^r - \mathbf{p}_l^o}{sd_l^k(\mathbf{q}_i)} \frac{\partial \mathbf{p}_k^r}{\partial \mathbf{q}_i} = \frac{\mathbf{p}_k^r - \mathbf{p}_l^o}{sd_l^k(\mathbf{q}_i)} \mathbf{J}_l^k(\mathbf{q}_i) \quad (4.60)$$

where $\mathbf{J}_l^k \in R^{3 \times 7}$ is the Jacobian matrix for the position of k -th robot collision object.

Till now, the gradient of the constraints can be analytically written as follows:

$$\frac{\partial \bar{g}}{\partial \mathbf{U}} = \begin{bmatrix} \mathbf{0} & \frac{\partial sd_1^1(\mathbf{q}_1)}{\partial \mathbf{q}_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \frac{\partial sd_l^k(\mathbf{q}_1)}{\partial \mathbf{q}_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \frac{\partial sd_l^k(\mathbf{q}_{ph})}{\partial \mathbf{q}_{ph}} \\ & & \mathbf{I}_{7(ph+1)} & & \\ & & -\mathbf{I}_{7(ph+1)} & & \end{bmatrix} \mathbf{S}_U \quad (4.61)$$

In order to reduce the computation, the robot body is represented by three boxes and the robot manipulator is represented by three cylinders instead of by the precise meshes of robot links in collision detection. Figure 4.6 shows such simplification.

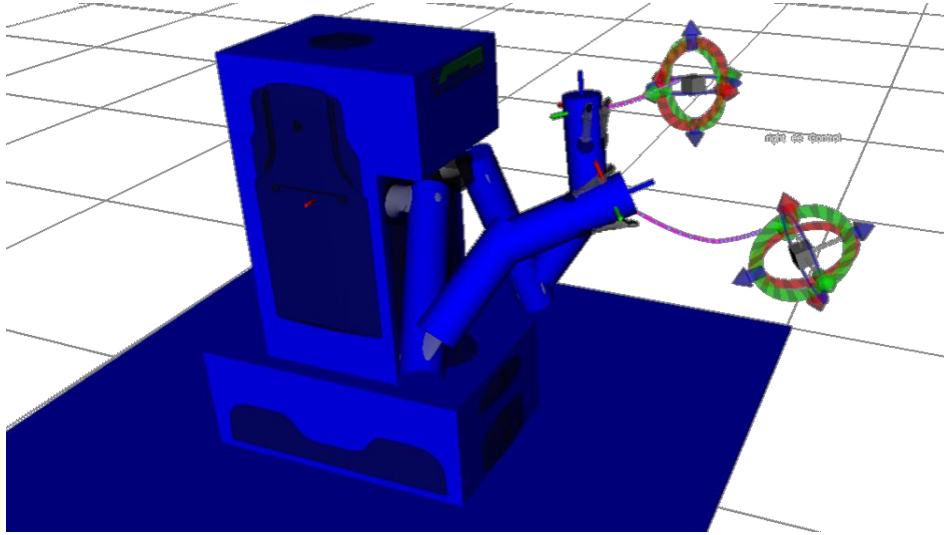


Figure 4.6: The collision objects used to represent the Movo robot. In order to plan for both arms, we run the NMPC-MP twice. As a result, the environmental constraints for each hand are displayed as primitive shapes in blue from each NMPC-MP's perspective. [1], ©2021 IEEE.

4.2.4 Sequential quadratic programming method

The NMPC Equation 4.44 for robot motion planning is converted to a nonlinear optimization problem of the following general form:

$$\begin{aligned} & \min_{\mathbf{x}} f(\mathbf{x}) \\ & \text{subject to} \\ & g_i(\mathbf{x}) \leq 0, \quad \text{for } i = 1, \dots, m \end{aligned} \quad (4.62)$$

For the NMPC problem that only considers obstacle avoidance and physical limits during motion planning, no equality constraints are imposed. As discussed in last subsection, given a hypothesis on \mathbf{x} , the values and gradients of $f(\mathbf{x})$ and $g_i(\mathbf{x})$ can be analytically obtained.

Now, we use SQP to solve problem Equation 4.62. At each iteration, an approximation is made for the Hessian of the Lagrangian function of problem Equation 4.62 using BFGS method [91]. The Hessian is used along with the given values and gradients information to form a quadratic QP subproblem. The solution of the QP is used as a search direction for the subsequent line search procedure. QP problems can be solved by different advanced QP solvers such as OSQP[92] and qpOASES[93] in microseconds. Here, we use an open-source nonlinear optimizer nlopt [94] to solve the QP and conduct line search.

Given a initial guess \mathbf{x}_0 , \mathbf{x}_{k+1} is obtained from \mathbf{x}_k by the following update rule [91]:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \quad (4.63)$$

where \mathbf{d}_k is the search direction found by solving the QP subproblem at k -th iteration and α_k is the step length found by line search.

QP subproblem. At k -th iteration, a QP is formulated by the a second-order Taylor expansion of the Lagrangian function of the original problem:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i g_i(\mathbf{x}) \quad (4.64)$$

and a first-order Taylor expansion of constraints. Then, a QP of standard form is given as:

$$\begin{aligned} \min_{\mathbf{d} \in \mathbb{R}^n} \quad & \frac{1}{2} \mathbf{d}^T \mathbf{B}_k \mathbf{d} + \nabla f(\mathbf{x}_k)^T \mathbf{d} \\ \text{subject to} \quad & \\ & \nabla g_i(\mathbf{x}_k)^T \mathbf{d} + g_i(\mathbf{x}_k) \leq 0, \text{ for } i = 1, \dots, m \end{aligned} \quad (4.65)$$

where $f(\mathbf{x}_k)$, $\nabla f(\mathbf{x}_k)$, $g_i(\mathbf{x}_k)$ and $\nabla g_i(\mathbf{x}_k)$ are calculated given \mathbf{x}_k . \mathbf{B}_k is the Hessian matrix of the Lagrangian function Equation 4.64: $\mathbf{B}_k = \nabla_{\mathbf{x}\mathbf{x}}^2 L(\mathbf{x}, \boldsymbol{\lambda})$. Exactly calculating \mathbf{B}_k at each iteration causes significantly high computation overheads. In order to increase the efficiency of SQP, the Hessian is only iteratively approximated using gradients information. This approximation of the Hessian is known as quasi-Newton methods, among which BFGS-formula is the most popular. The update rule of Hessian according to BFGS-formula is stated as follows:

$$\begin{aligned} \mathbf{B}_{k+1} &= \mathbf{B}_k + \frac{\mathbf{q}_k \mathbf{q}_k^T}{\mathbf{q}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k^T}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \\ \text{with} \quad & \\ \mathbf{s}_k &\doteq \mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k \mathbf{d}_k, \\ \mathbf{q}_k &\doteq \theta_k \boldsymbol{\delta}_k + (1 - \theta_k) \mathbf{B}_k \mathbf{s}_k \end{aligned} \quad (4.66)$$

where \mathbf{s}_k denotes the difference between current and next iterate, i.e. the calculated update, $\boldsymbol{\delta}_k$ denotes the difference between the gradients of two Lagrangian functions:

$$\boldsymbol{\delta}_k \doteq \left(\nabla f(\mathbf{x}_{k+1}) + \sum_{i=1}^m \lambda_i \nabla g_i(\mathbf{x}_{k+1}) \right) - \left(\nabla f(\mathbf{x}_k) + \sum_{i=1}^m \lambda_i \nabla g_i(\mathbf{x}_k) \right) \quad (4.67)$$

B needs to remain positive definite during the update, and this is achieved by choosing θ_k from Equation 4.66 as:

$$\theta_k \doteq \begin{cases} 1, & \text{if } \mathbf{s}_k^T \boldsymbol{\delta}_k \geq 0.2 \mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k \\ \frac{0.8 \mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k - \mathbf{s}_k^T \boldsymbol{\delta}_k}, & \text{otherwise} \end{cases} \quad (4.68)$$

This definition ensures that $\mathbf{s}_k^T \boldsymbol{\delta}_k \geq 0.2 \mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k$, which keeps B_{k+1} positive definite within the linear manifold defined by the tangent planes to active constraints at \mathbf{x}_{k+1} [91].

Line search. The solution of the QP serves as a search direction for a line search problem to find an optimal step size α^* that minimizes some merit function φ :

$$\varphi(\alpha) \doteq f(\mathbf{x}_k + \alpha \mathbf{d}_k) + \sum_{i=1}^m \mu_i |g_i(\mathbf{x}_k + \alpha \mathbf{d}_k)|^+ \quad (4.69)$$

with $|z|^+ = \max(0, z)$ and the penalty coefficient μ_i is updated according to $\mu_i \doteq \max((1/2)(\mu_i^- + |\lambda_i|), |\lambda_i|)$ for $i = 1, \dots, m$ where μ_i^- is the penalty coefficient from last iteration and λ_i denotes the Lagrangian multiplier for i -th constraint g_i .

4.2.5 Real-time scheme and parallelization

The planner has to satisfy the minimum 10Hz frequency required by the real-time teleoperation. Therefore, a maximum iteration number is set to restrain the optimizer from waiting too long for an optimum. Regardless of optimality, the updated hypothesis at the last iteration is returned. Although the solution trajectory is not optimal, it has been improved in contrast to the initial hypothesis in terms of approaching the goal. The solution is called a suboptimum.

The quality of the suboptimum can be improved and the convergence time of the optimization can be further reduced by introducing a real-time scheme. Since optimizations are performed time instant after time instant, two adjacent optimizations don't differ too much from each other, as long as the environment between two time steps doesn't change too much. For a NMPC-MP, only the joint velocities \mathbf{u}_0^* in the first time step of the planned trajectory are executed and the remaining the trajectory $[\mathbf{u}_1^*, \dots, \mathbf{u}_{ch-1}^*]$ is a close guess on the solution of next optimization. Hence, the remaining trajectory is reserved and shifted one time step forward and a new guess is appended to form a new joint velocities sequence: $[\mathbf{u}_1^*, \dots, \mathbf{u}_{ch-1}^*, \mathbf{u}_{new}]$, this sequence is a good starting point for the next optimization and cuts some iterations needed to converge to the optimum as opposed to optimizing from an arbitrary starting point.

Newly formulated starting points can be infeasible. Optimizations starting from an infeasible region have a high failure probability. Therefore, before feeding the starting point to the next optimization, another simple optimization is performed on the starting point, in order to first update it to the feasible region:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) = C \\ \text{subject to} \quad & \\ & g_i(\mathbf{x}) \leq 0, \quad \text{for } i = 1, \dots, m \end{aligned} \quad (4.70)$$

where C can be an arbitrary constant. This optimization is given a relatively larger number of iterations. After performing it, a feasible starting point is guaranteed and the subsequent optimization will be performed within the feasible region.

Warm start. A warm start is a feasible solution to your problem and the nature of optimization problems is to evaluate feasible regions (all feasible solutions) in order to find the optimal solution, which is not the original solution. Providing a warm start to the solver, you help the solver converge in fewer iterations.

Aside from the warm start, a parallelization is implemented. The bottleneck for the optimization is the calculation of the cost and constraint values, as each calculation needs to be performed ph times. Therefore, ph calculations are assigned to multiple threads. After all threads complete, values are summed up, and the final results are calculated. This concurrency further accelerates the optimization.

Till now, a NMPC-MP with SQP to find the solution is realized. The structure of the system can be illustrated in the following diagram.

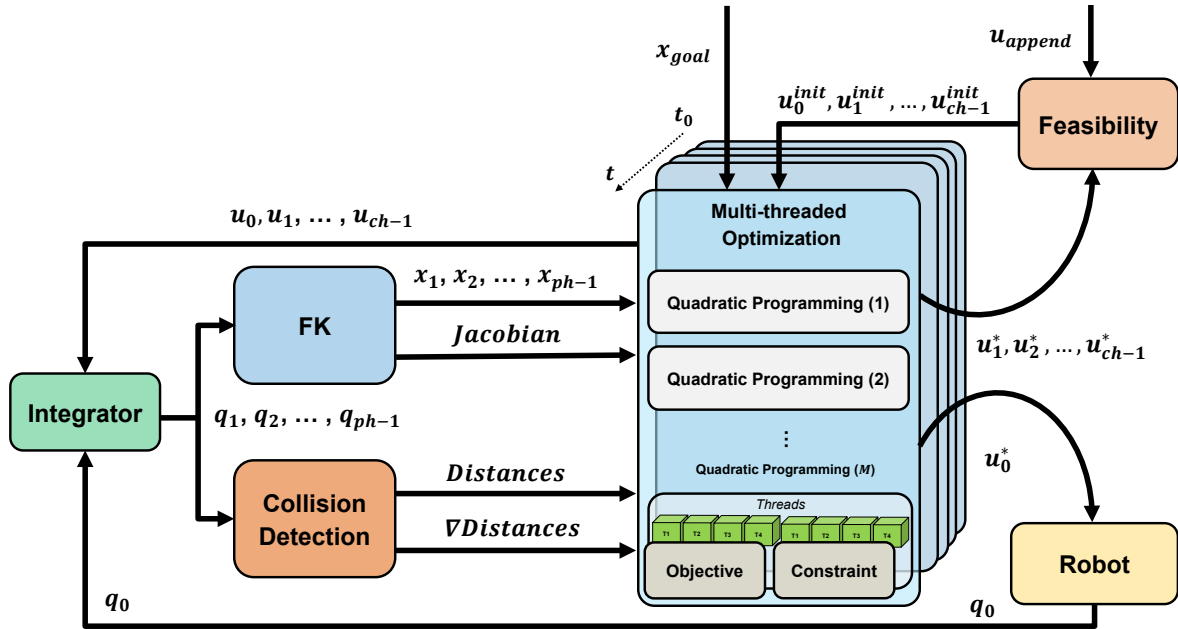


Figure 4.7: System diagram for NMPC-MP [1], ©2021 IEEE.

Algorithm 1 describes the pseudo-code for NMPC-MP. \hat{U}_{init} is the feasible warm start point built from the unexecuted solution inputs of the last optimization. Calculations of cost and constraints in loops can be split into multiple threads and performed concurrently. After Max_QP_Num iterations, the updated U is returned as the solution.

For a clear distinction in motion planner design in this dissertation, the main contributions are:

- Proposal of a motion planner based on nonlinear model predictive control for Jaco2 redundant manipulator.
- Warm start and multi-threading mechanisms have been implemented, enabling real-time motion planning. This makes it suitable for avoiding obstacles in dynamic environments.

4.3 Experiments and results

In this section, simulations and comparisons are conducted to show that our method is superior to the benchmarks in terms of planning time and planning quality. And then, the method is applied on the teleoperation of the real Movo robot to validate that our method is eligible in real-time teleoperation applications. Simulations and real experiments are running on an Intel[®] Xeon[®] Gold 6242 CPU with 16 cores at 2.80GHz. The software is running on Ubuntu[®] 18.04 LTS and ROS Melodic. Besides, for real experiments, the HMD and VR trackers are connected to ROS from Unity3D[®] via [80]. For the optimization's multi-threading mechanisms, four threads are set up for calculating the objective function and four threads for the constraints.

For the following experiments, unless specially mentioned, parameters for the NMPC-MP follow Table 4.2, where Q is built by $Q = \text{diag}(Q^{pos}, Q^{pos}, Q^{pos}, Q^{quat}, Q^{qua}, Q^{quat})$ and analogously for P_f .

4.3.1 Simulations

The proposed NMPC-MP is first tested with primitive obstacles in simulations. Figure 4.8 shows two simulations, one is with a larger static obstacle and the other with a small dynamic obstacle. Orange lines are the actual trajectories of the end-effector and blue lines with green dots are the initial predicted trajectory generated by the NMPC at the beginning based on the initial robot states and environment. As shown in Figure 4.8a, due to the limit of the prediction and control horizons, at the beginning, the NMPC can not foresee a trajectory that reaches the goal. Anyhow, it will move one step forward according to this initial "imperfect" trajectory and a new NMPC for finding the trajectory is formulated. This new NMPC should have a slightly easier optimization problem for finding the trajectory. In this way, NMPC in each time step contributes a bit to the whole problem, and the accumulation of optimizations results in a successful actual motion of the robot. In Figure 4.8b, a dynamic obstacle is applied and is moving along the red line. At the early phase, the obstacle is far away, the initial predicted trajectory (blue) and actual executed trajectory (orange) roughly coincide and approach the goal. With the obstacle approaching, as the actual trajectory shows, NMPC gives up staying at the goal position and tries to avoid the obstacle and find another way to the goal.

Algorithm 1: NMPC-MP

```

 $\mathbf{U}_{init} \leftarrow [\mathbf{u}_1^{*,T}, \dots, \mathbf{u}_{ch-1}^{*,T}, \mathbf{u}_{append}^T]^T;$ 
 $\hat{\mathbf{U}}_{init} \leftarrow \text{Feasibility}(\mathbf{U}_{init}, \mathbf{q}_0);$ 
 $\mathbf{U} \leftarrow \hat{\mathbf{U}}_{init};$ 
for  $k=1$  to  $\text{Max\_QP\_Num}$  do
     $\mathbf{q}_1, \dots, \mathbf{q}_{ph-1} \leftarrow \text{SystemModel}(\mathbf{q}_0, \mathbf{U});$ 
     $\text{cost} \leftarrow 0; \text{cost\_grad} \leftarrow \mathbf{0}_{7ch \times 1};$ 
    for  $i=1$  to  $ph-1$  do
         $\text{cost} \leftarrow \text{cost} + \text{ObjFunc}(\mathbf{x}_d, \mathbf{q}_i);$ 
         $\text{cost\_grad} \leftarrow \text{cost\_grad} + \frac{\partial \text{ObjFunc}}{\partial \mathbf{x}_i} \cdot \frac{\partial \mathbf{x}_i}{\partial \mathbf{q}_i} \cdot \frac{\partial \mathbf{q}_i}{\partial \mathbf{U}};$ 
    end
     $\text{cnt} \leftarrow \mathbf{0}_{n(ph-1) \times 1}; \text{cnt\_grad} \leftarrow \mathbf{0}_{n(ph-1)7ch \times 1};$ 
    //  $\text{cnt}$ : closest distances to obstacles;
    for  $i=1$  to  $ph-1$  do
         $\text{cnts}[in - n : in - 1], \text{cps} \leftarrow \text{FCL}(\mathbf{q}_i);$ 
        //  $n$ : number of obstacles;
        //  $\text{cps} \in R^{3n \times 2}$ :  $n$  pairs of closest points;
         $\text{dist\_directions} \leftarrow \text{Normalize}(\text{cps});$ 
         $\text{cnt\_grad}[(in - n)7ch : in7ch - 1] \leftarrow \text{dist\_directions} \cdot \frac{\partial \text{cps}[:,0]}{\partial \mathbf{q}_i} \cdot \frac{\partial \mathbf{q}_i}{\partial \mathbf{U}};$ 
    end
     $\mathbf{d} \leftarrow \text{QP}(\text{cost}, \text{cost\_grad}, \text{cnt}, \text{cnt\_grad});$ 
     $\alpha \leftarrow \text{LineSearch}(\mathbf{U}, \mathbf{d}, \mathbf{q}_0);$ 
     $\mathbf{U} \leftarrow \mathbf{U} + \alpha \mathbf{d};$ 
end
 $\mathbf{u}_0^*, \mathbf{u}_1^*, \dots, \mathbf{u}_{ch-1}^* \leftarrow \mathbf{U};$ 
Execute( $\mathbf{u}_0^*$ ); Reserve( $\mathbf{u}_1^*, \dots, \mathbf{u}_{ch-1}^*$ );

```

Table 4.2: Parameters for NMPC Motion Planner

Q^{pos}	Q^{quat}	P_f^{pos}	P_f^{quat}	R	ph	ch	Max number of iterations
5	0.1	50	10	2	20	10	20

Figure 4.9a records the evolution of stage costs and the closest distances of three collision cylinders to the obstacle. Here stage costs are defined as $e(\mathbf{X}_d, \mathbf{X}_i)^T \mathbf{Q} e(\mathbf{X}_d, \mathbf{X}_i)$, i denotes current time instant for actual cost and the i -th time step in the prediction horizon for predicted cost. In the upper figure, the orange curve predicts the change of the stage cost from the initial time to 4s into the future. The solution of this initial problem tends to converge to a local minimum. Blue curve denotes the evolution of actual stage cost, it loosely follows the predicted horizon to the local minimum at the beginning but afterward it finds the real solution from newly formulated optimizations. The lower figure shows that the closest distances to the obstacle are kept beyond a safety distance of 0.02m all the time.

Figure 4.9b simulates a scenario, where an obstacle approaches and goes through the goal position. The upper figure shows that, when the obstacle is far away, predicted and actual

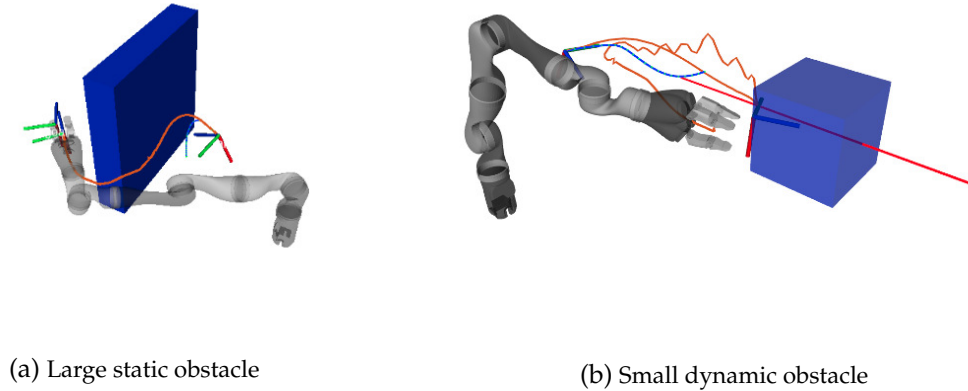


Figure 4.8: Simulations with primitive obstacles for NMPC-MP

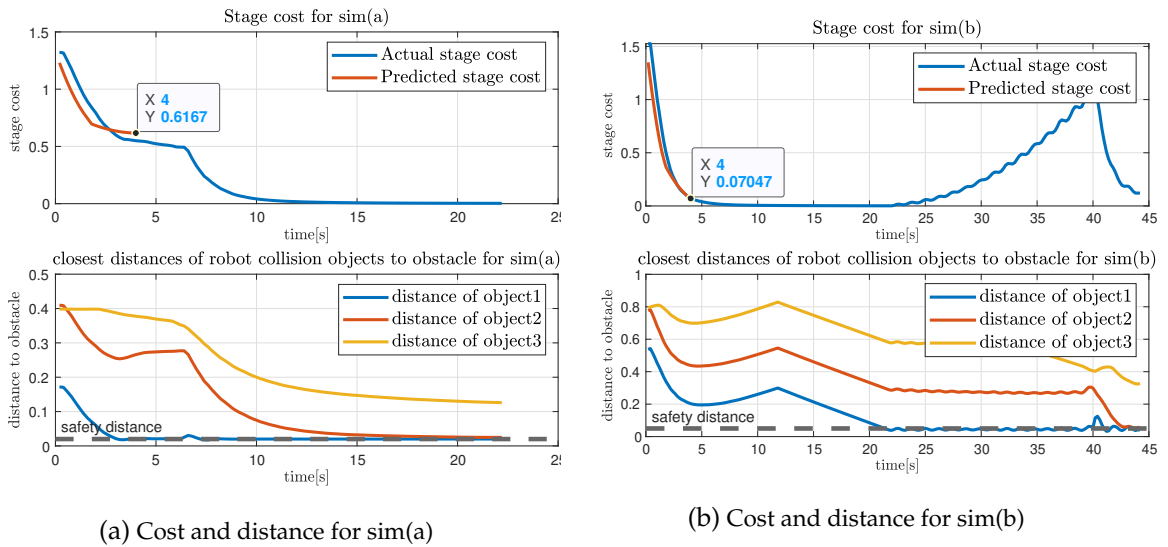


Figure 4.9: Stage costs and the closest distances to the obstacle

costs are directly reduced by a trajectory to the goal. Later, due to the hard constraints on obstacle avoidance, costs are raised once again but the NMPC-MP still attempts to find another trajectory to the goal via newly formulated optimization, in order to minimize the cost once more. In the lower figure, the distance of object1 sometimes slightly breaks the hard constraint for a safety distance of $0.05m$. This is because that, the closest distance between the robot and the obstacle is non-differentiable as a function of joint velocities in degenerate cases. And the actual closest distance deviates from the approximated one, as the approximation assumes that closest contacting points don't change during subtle motion of objects but in reality the closest points will move to other positions when objects move. Inaccurate approximation only causes a small break of obstacle constraints and this can be well handled by setting a slightly larger safety distance.

Prediction horizon and control horizon. ph and ch are the two most crucial parameters that decide the performance of the NMPC-MP. Larger values result in optimization prob-

lems of higher dimensions at each time instant while small values make the planner prone to local minima. Figure 4.10 illustrates some examples of the evolution of actual stage cost

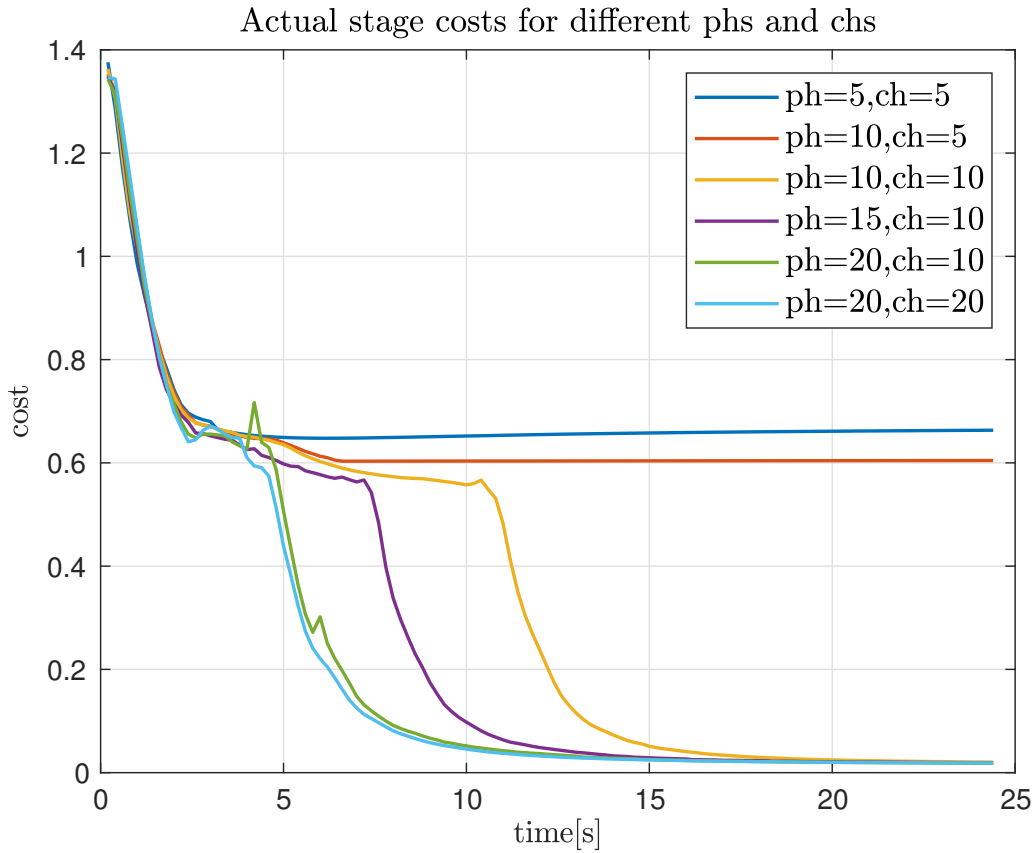


Figure 4.10: Actual stage costs for NMPC planners different phs and chs [1], ©2021 IEEE.

for NMPC-MPs with different phs and chs performing the same simulation as Figure 4.8a, where a large obstacle is intentionally set and planners can fall into local minimum easily. When $ph = 10, ch = 5$, the manipulator gets stuck in a position in front of the obstacle. For $ph = 15, ch = 10$, the manipulator stays in local minimum for some time and then, a new optimization finds a solution to reduce the cost to 0. For $ph = 20, ch = 20$, the manipulator is able to escape from the local minimum quickly.

It is challenging to select suitable ph and ch for various applications. For example, in a scenario where obstacles are few or are sparsely located, the risk for getting stuck is low, and smaller ph and ch are enough. For a scenario where obstacles are densely positioned, large ph and ch are necessary. Selecting right ph and ch is out of the scope of this dissertation, we choose $ph = 20, ch = 10$ as they produced a relatively good performance and the corresponding optimization is solved at more than $10Hz$.

Octomap obstacle. With the help of the flexible-collision-library (FCL) [95], the NMPC-MP is able to handle arbitrarily shaped obstacles of not only primitives but also mesh and octomap types. Here, a Microsoft® Kinect2 camera is used in the simulation to capture the point cloud of a bookshelf; this point cloud is converted to the octomap and considered by

the NMPC-MP as an obstacle. The end-effector trajectory in Fig. 4.11 shows how the manipulator avoids the octomap during its motion.

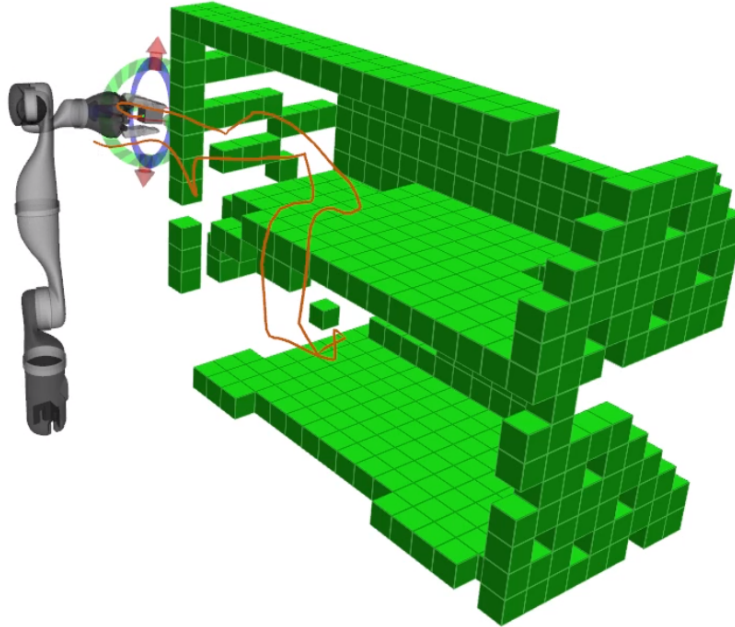


Figure 4.11: Trajectory of the end-effector during the motion of the manipulator considering octomap obstacle avoidance. [1], ©2021 IEEE.

4.3.2 Comparison scenarios

In this subsection, we compare our method NMPC-MP in simulation with popular offline motion planners and the state-of-the-art online controller RelaxedIK. Three tasks are defined: (i) jogging arm (ii) planning trajectory for far goals without obstacle; (iii) planning with a static obstacle, which is the same task as that of simulation(a) in Figure 4.12.

Task(i): Jogging Arm Without Obstacles. In the first task, the jogging of the robot arm is tested. Jogging means continuously moving the end-effector by a little to reach a goal position near the current position. Offline planners can only generate accurate motions for reachable goals. Goals that are difficult to reach lead to a low success rate and long planning time. Thus, offline planners are not applicable in jogging where planning time is limited, and planning failure is intolerable. Here our method and RelaxedIK¹ are tested. For small motions in jogging, the prediction is unnecessary, and hence, ph and ch for NMPC-MP are set to one. Table 4.3 compares the performance of the two methods. Here, the average normalized length is defined as the ratio of the actual trajectory length to the straight-line distance between starting end-effector position and the goal position. In arm jogging, this value is one as goals are close to the starting position, and the trajectories to the goals are roughly a line between them.

¹ Officially released python version of relaxedIK (Commit: 18/05/2020)

Table 4.3: Controller performance for task(i)

	Avg normalized length	Avg time (second)	Avg joint velocity (radian/second)	Avg position squared error	Avg quaternion squared error	Success rate
NMPC-MP	1.0	0.016	0.033	1.10e-3	1.90e-3	1.0
RelaxedIK	1.0	0.106	0.11	4.00e-3	2.00e-4	1.0

Table 4.4: Planner/Controller performance for task(ii)

	Avg normalized length	Avg time (second)	Avg joint velocity (radian/second)	Avg position squared error	Avg quaternion squared error	Success rate
NMPC-MP	1.365	0.0848	0.1427	2.10e-3	2.70e-3	1.0
RelaxedIK	1.3677	0.1271	0.1907	1.30e-3	4.27e-4	1.0
RRT-Connect	2.46	0.075	0.0163	5.97e-9	2.34e-7	0.833

Task(ii): Far Goals Without Obstacles. In the second task, the motion planner should generate a trajectory to be reached the far goals. We set six far goals for the end-effector to reach one after the other. Three methods are tested in this scenario: NMPC-MP, RelaxedIK, and RRT-Connect.

Fig. 4.12 draws the actual trajectories of the end-effector. The offline calculated trajectory from RRT-Connect is the smoothest. However, the success rate is low, as shown in Table 4.4. Furthermore, the trajectory from RRT-Connect sometimes can be too long. NMPC-MP and RelaxedIK are running in real-time. Their trajectories are not pre-calculated and thus are less smooth. Nevertheless, with the prediction capability, the actual trajectory from NMPC-MP gets smoothed and optimized to a shorter length. The trajectory of RelaxedIK has some sharp turns between neighboring goals, which could be dangerous and should be avoided.

Table 4.4 compares the three methods for task(ii) using different criteria. NMPC-MP exhibits strengths for real-time application but relaxes requirements on diminishing errors to guarantee success in planning.

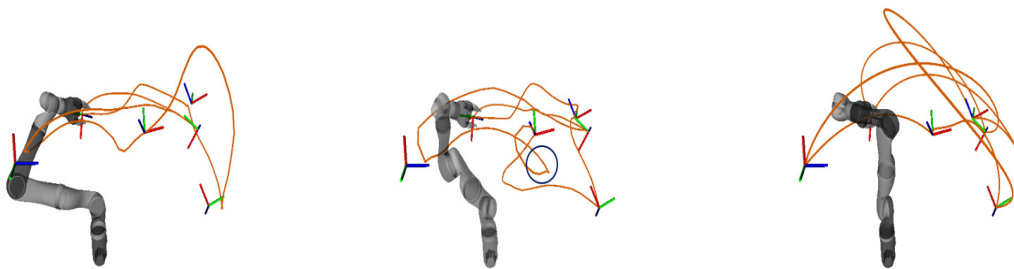


Figure 4.12: Trajectories for task(ii). *Left:* NMPC-MP. *Middle:* RelaxedIK. *Right:* RRT-Connect. [1], ©2021 IEEE.

Task(iii): Far Goal With Static Obstacle. In the third task, a box obstacle is added. RelaxedIK does not consider external obstacles and thus can not handle this task. Our method is compared against offline motion planners. Here, STOMP can only handle goal joint posi-

Table 4.5: Planner/Controller performance for task(iii).

	Avg normalized length	Avg time (second)	Avg joint velocity (radian/second)	Avg position squared error	Avg quaternion squared error	Success rate
NMPC-MP	1.76	0.088	0.063	1.66e-3	8.60e-3	1.0
STOMP	3.1	0.613	0.0138	1.81e-3	8.33e-3	1.0
RRT	5.13	0.51	0.015	7.60e-9	5.00e-1	1.0
RRT Connect	5.6	0.3	0.02	9.00e-9	4.00e-1	1.0
PRM*	3.86	5.09	0.019	2.46e-9	1.70e-2	1.0

tion; RRT, RRT-Connect and PRM* fail to find a path to fulfill the posture goal, the orientation goal is discarded, only the position goal is set.

Table 4.5 shows that NMPC-MP is the fastest and generates the shortest trajectory. Also, as can be seen in Fig. 4.13, although the motion of the robot is calculated in real-time, the actual trajectory of NMPC-MP is smooth enough.

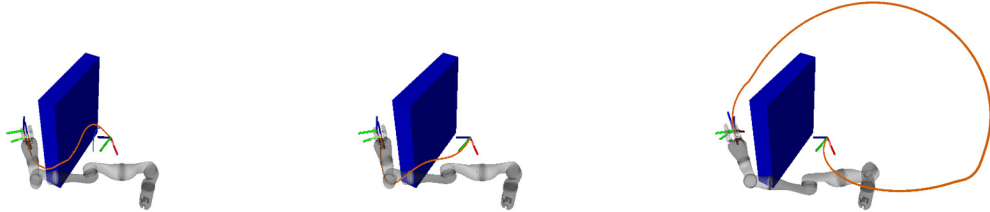


Figure 4.13: Trajectories for task(iii). *Left:* NMPC-MP. *Middle:* STOMP. *Right:* RRT-Connect. [1], ©2021 IEEE.

4.3.3 Experiments on the real Movo robot

In the end, we apply our method on the real Movo robot and teleoperate both robot end-effectors to reach goals. For the first real experiment, an HTC[®] Vive Tracker is put in front of the robot representing a spherical obstacle which a rightward motion of the left arm is supposed to avoid. ch for the task is set to 20, and the maximum QP number to 30. As shown in Figure 4.14, goal points form a trajectory going through the sphere, but the predicted trajectory from NMPC-MP in every iteration makes the robot bypass the obstacle.

In the second real experiment, two interactive markers specify goals for both arms in the Rviz visualizer and an HTC[®] Vive Tracker as the dynamic spherical obstacle comes in during their motions. In Figure 4.15, arms are avoiding the dynamic sphere.

Comparison results in simulation demonstrate that our method outperforms other motion planners and controllers on calculation time and trajectory quality. Experiments on the real Movo robot also show that our method can be applied to real-time manipulator teleoperation with safety concerns.

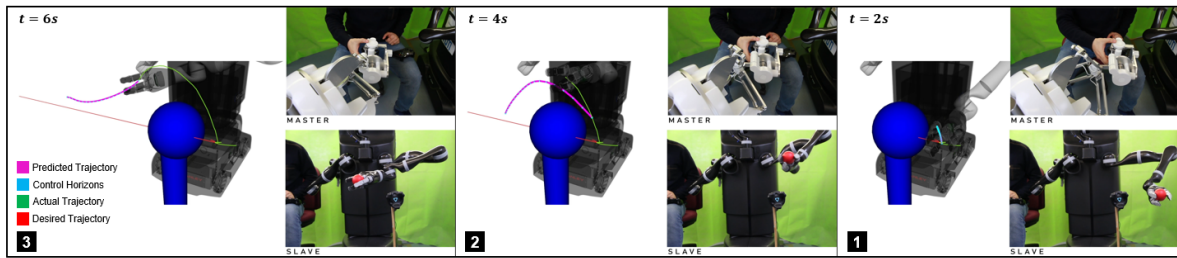


Figure 4.14: The first real experiment: NMPC-MP's static obstacle avoidance. [1], ©2021 IEEE.

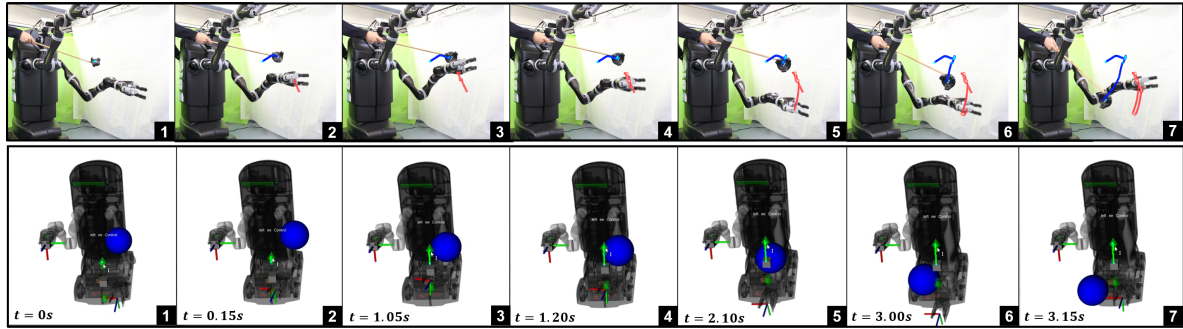


Figure 4.15: The second real experiment: NMPC-MP's dynamic obstacle avoidance in motion. [1], ©2021 IEEE.

4.4 Adapting NMPC-MP for Franka

As the Jaco2 manipulator has unlimited joints, this section will test NMPC-MP on a manipulator with limited joint movements. This will add additional constraints to the planner. For this purpose, we have selected the Franka robot from Franka. Using the Franka as the standard platform for manipulator research in this chapter, we explored the possibility of teleoperating this robot with dynamic obstacles.

4.4.1 Forward Kinematic Calculation

In this section, we recalculate the forward kinematic part to support quick and accurate calculation for getting the parameters of Franka. DH parameters are four parameters associated with a particular convention for attaching reference frames to the links of a spatial kinematic chain, or robot manipulator. For the standard DH parameter, the four transformation parameters are:

- d : offset along the previous z to the common normal
- θ : angle about the previous z , from old x to new x
- r : length of the common normal
- α : angle about common normal, from old z -axis to new z -axis

Table 4.6: Modified Denavit-Hartenberg Parameters.

Joint i	theta	d	a(i-1)	alpha(i-1)
Joint1	q_1	0.333	0	0
Joint2	q_2	0	0	$-\pi/2$
Joint3	q_3	0.316	0	$\pi/2$
Joint4	q_4	0	0.0825	$\pi/2$
Joint5	q_5	0.384	-0.0825	$-\pi/2$
Joint6	q_6	0	0	$\pi/2$
Joint7	q_7	0.107	0.088	$\pi/2$

Modified (proximal) DH parameter is a special form. The difference between them is the locations of the coordinates system attached to the links and the order of the performed transformations. Therefore, the four parameters are theta, d, and in this case, a(i-1) and alpha(i-1) that denote the length and angle from the previous joint. The table of modified DH parameters given by Franka's official website is shown in Table 4.6. According to the new DH parameters we recalculated the following functions:

- Jacobian Matrix $J \in R^{6 \times 7}$ of the end-effector in rotation format
- Quaternion Jacobian Matrix $J_{quat} \in R^{7 \times 7}$ of the end-effector with the joint position
- Reference Jacobian Matrix $J_{ref} \in R^{3 \times 7}$ for the closest point calculated by FCL on the arm cylinder
- End-effector Transform $\in R^{4 \times 4}$
- Position $\in R^{7 \times 1}$ for three translations and four quaternions

4.5 Simulated Franka

In this section, we will modify the NMPC-MP based on Franka. These steps include reshaping collision cylinders and adding the joint limits.

4.5.1 Reconfiguration of cylinders

As mentioned before the collision detection in our NMPC-MP is completed by the FCL[95]. However, the objects supported by FCL are only a few simple basic shapes, which obviously cannot directly work with the irregular shape of the manipulator. Therefore, after obtaining the manipulator model, it is necessary to put a suitable cylinder on the outside of it. In other words, we do not accurately detect the collision of the manipulator but detect the collision of a virtual, invisible cylinder shell. This is an approximate method, and there is an error that

is caused by the distance between the cylinder and the manipulator but makes the motion safer.

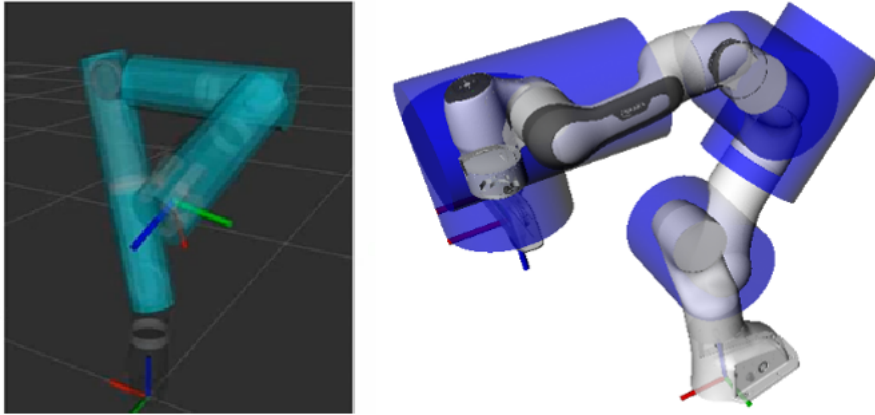


Figure 4.16: Cylinder representation of the Movio on the left and the Franka on the right.

In this part, the difference between Franka and Movio is quite big. As shown in the [Figure 4.16](#), Movio is based on three links, and three cylinders can almost cover the entire manipulator. Franka, because the link moves in different but fixed directions, it needs to be based on five links and five cylinders to cover the manipulator. The more completion the cylinder covers, the smaller the obstacles that can be dealt with by collision detection, and the lower the possibility of error. In both simulated and real experiments, collision detection is based on this cylinder model.

4.5.2 Optimization with limits

In this section, we add a constraint to solve the irregular motion caused by the joint limits of Franka's characteristics. So far, simulated Franka can apply NMPC-MP as we did for Move. The joints of the Kinova robot are continuous joints that rotate around the axis and have no upper and lower limits. Therefore, in addition to the basic parameters, the optimization process only needs to set a fixed upper and lower limit of joint velocity. In other words, the optimal joint velocity obtained by any set of joint positions can be executed by Kinova. However, the Franka robot has revolute joints that rotate along the axis and have a limited range specified by the upper and lower limits as [Table 4.7](#). Whether it is a simulated Franka or a real Franka, once the joint position exceeds the position limit, the joints of the robot will be locked by the joint "collision error" state, and cannot move anymore.

The upper and lower joint velocity limits in the Kinova program are the specified fixed velocity range for safe motion, so when the program starts and initializes nlopt optimizer, the settings can be completed at one time. Considering that the upper and lower limits of Franka need to be adjusted according to the difference between the current joint position and the limit at each time step, we put the setting of velocity limit of nlopt optimizer in the function that receives the joint state from the ROS topic with the receiving time interval as 0.2 seconds. After receiving the current joint state, we read the joint position, calculate the position

error from limits to get the velocity limit range of the current time step, compare it with the customized defined safe velocity if needed, and assign it to the nlopt optimizer. Note that, on the simulation side, the limits can be set according to the data in the Table 4.7.

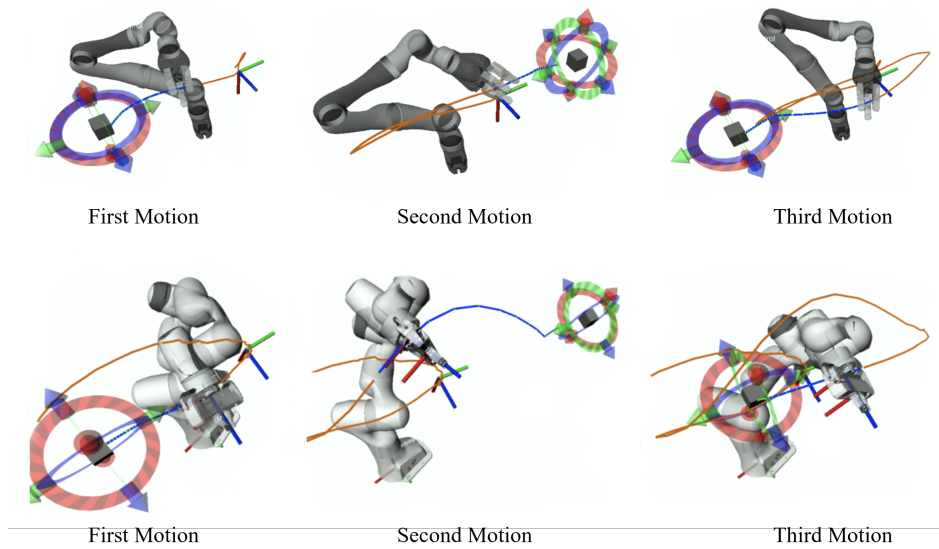


Figure 4.17: The trajectory of Mov0 and Franka

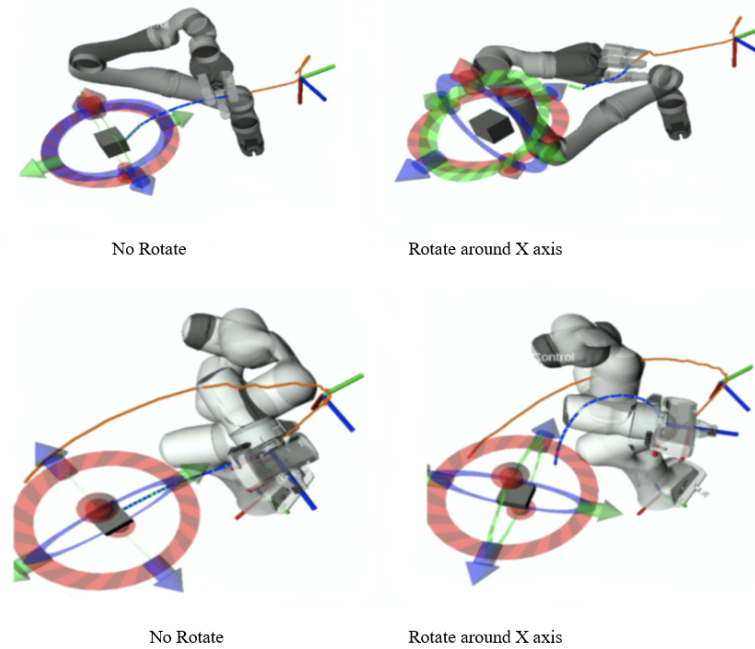


Figure 4.18: Rotated goal to Mov0 and Franka

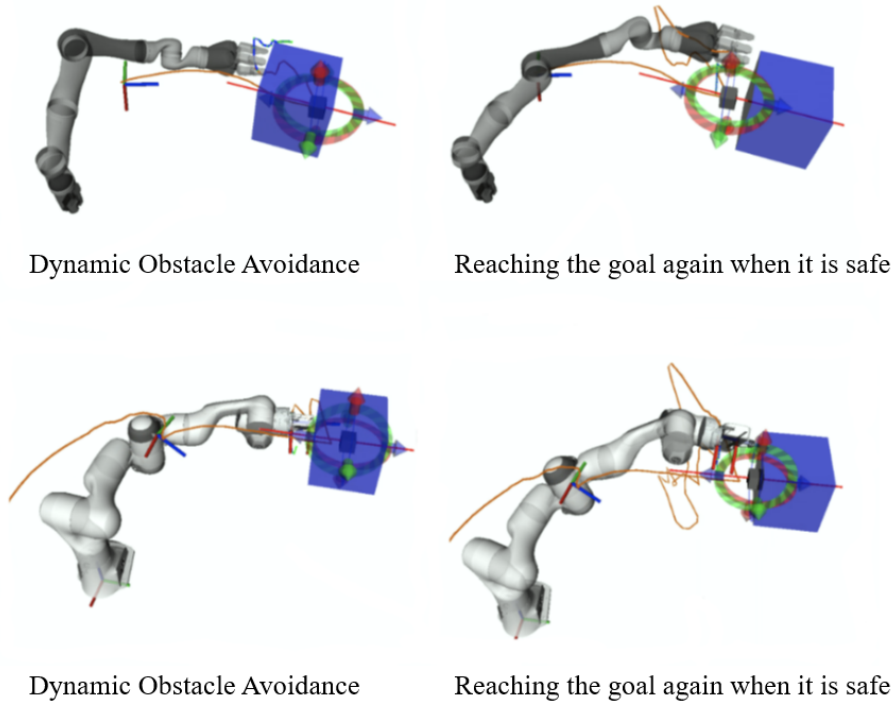


Figure 4.19: Virtual obstacle avoidance on Movo and Franka.

Table 4.7: Joint Space Limits of Franka.

Name	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Joint 7	Unit
q_{\max}	2.8973	1.7628	2.8973	-0.0698	2.8973	3.7525	2.8973	rad
q_{\min}	-2.8973	-1.7628	-2.8973	-3.0718	-2.8973	-0.0175	-2.8973	rad
\dot{q}_{\max}	2.1750	2.1750	2.1750	2.1750	2.6100	2.6100	2.6100	$\frac{\text{rad}}{\text{s}}$
\ddot{q}_{\max}	15	7.5	10	12.5	15	20	20	$\frac{\text{rad}}{\text{s}^2}$

4.6 Real Franka

When NMPC-MP runs successfully in the simulated Franka environment, it means that our overall program has been almost completely modified. In this chapter, we verify whether the real Franka can also achieve this motion planner.

In this experiment, we show the predicted path (the blue line represents the predicted path, the green dot represents the 20 predicted results in sequence), and the motion trajectory of the manipulator (red line). We set the initial end-effector position of the robots to the same position through pre-processing. During the experiment, the goal is moved several times in roughly the same direction, which was used to verify the response of the NMPC-MP to the translation in the position and whether the path optimization function met expectations.

In this experiment, while moving the goal, we apply a rotation (counterclockwise on the

read cycle) to change the quaternion of the goal, verifying that NMPC-MP can adjust for a rotated goal to improve stability by ensuring that the contact point between them remains unchanged, and the operating direction remains the same.

Next, we verify the obstacle avoidance. We set the goal at the dynamic obstacle motion path, verify that when the obstacle approaches, the manipulator can avoid the obstacle with a relatively smooth trajectory, and when the obstacle is far away, instantly update the predicted path in the current state, and try to re-reach the goal.

As shown in Figure 4.17, it can be seen that Franka, like Movo, successfully generated the predicted path, and because we only use the first predicted value for each step, the subsequent predicted values are only for reference and used for warm start, so the executed trajectory is different from the predicted path and does not completely overlap.

In Figure 4.18, when the goal rotates, the predicted path of NMPC-MP changes obviously. It can be seen that NMPC-MP reacts very sensitively to quaternion change, and can update the corresponding prediction path instantly, adjust the end-effector angle of the manipulator, to finally maintain a stable relative position with the goal.

And in Figure 4.19, when the obstacle is approaching, the manipulator avoids it by a smaller margin, and when the obstacle is far away, the manipulator generates a new prediction path, which is as close to the goal as possible under the constraints of the mechanical hardware. When the obstacle completely leaves the detection area, the goal will be re-reached as soon as possible.

After completing the functional verification in the simulated environment, we replace the manipulator with real Franka, execute the real Franka through virtual scene operations, and check the fluency of NMPC-MP process. Finally, we use HTC[®] Vive Trackers as a more flexible, real obstacle to verify the dynamic obstacle avoidance capability.

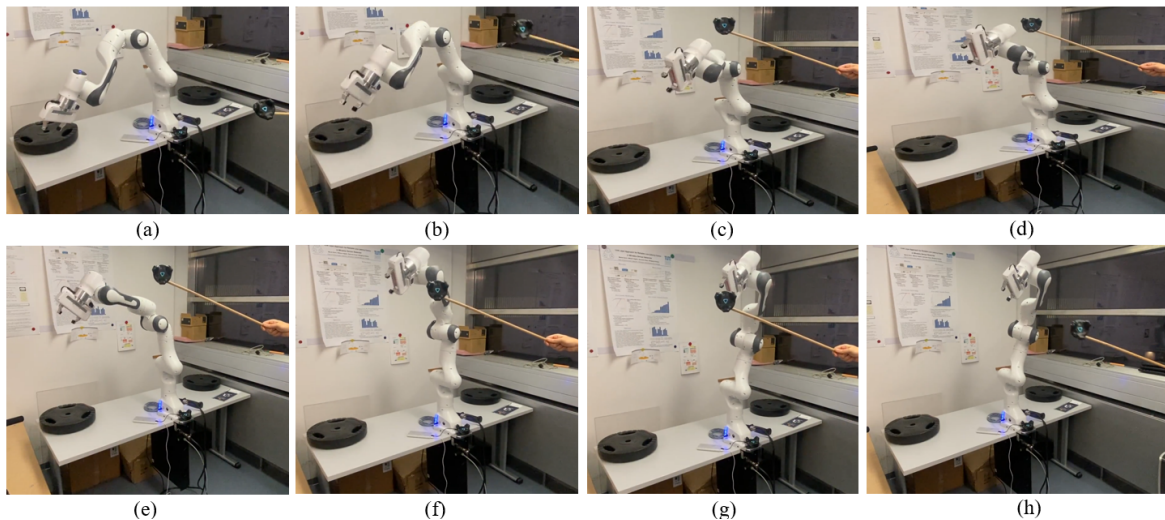


Figure 4.20: Demonstration of Franka’s real-time capability to avoid dynamic obstacles in remote teleoperation. The dynamic obstacle here in this picture is the HTC[®] Vive tracker.

4.7 Chapter Summary

Control and planning of the manipulator motion are essential components of manipulator teleoperation. Active obstacle avoidance can be challenging via real-time motion control, and is often accompanied by fluctuations and unsafe motion. By contrast, offline motion planning generates precise and secure trajectories for complex manipulation but they are not suitable for teleoperation. In this chapter, a real-time nonlinear model predictive control based motion planner (NMPC-MP) is designed for teleoperated manipulation. In contrast to traditional NMPC-based approaches, our model considers a complex environment with dynamic obstacles. Our multi-threaded NMPC-MP allows for real-time planning, including dynamic objects. We evaluate our approach both in a simulated environment and with real-world experiments using the Kinova[®] Movo platform and Franka Emika[®] Panda manipulator. The comparison to state-of-the-art approaches (e.g., RRT-Connect, CHOMP, and STOMP) shows a significant improvement in real-time motion planning using NMPC-MP. In real-world tests, the proposed planner was applied on a dual manipulator setup using the Kinova Movo platform as shown in [Figure 4.6](#) as well as the Franka manipulator [Figure 4.20](#). Our results show that the NMPC-MP runs in real-time and generates smooth and reliable trajectories. According to the experiments, the planner is able to precisely track goals from the teleoperator while avoiding self-collision and dynamic obstacles.

Chapter 5

Real-time Skill Refinement for Shared Autonomy in Manipulator Teleoperation

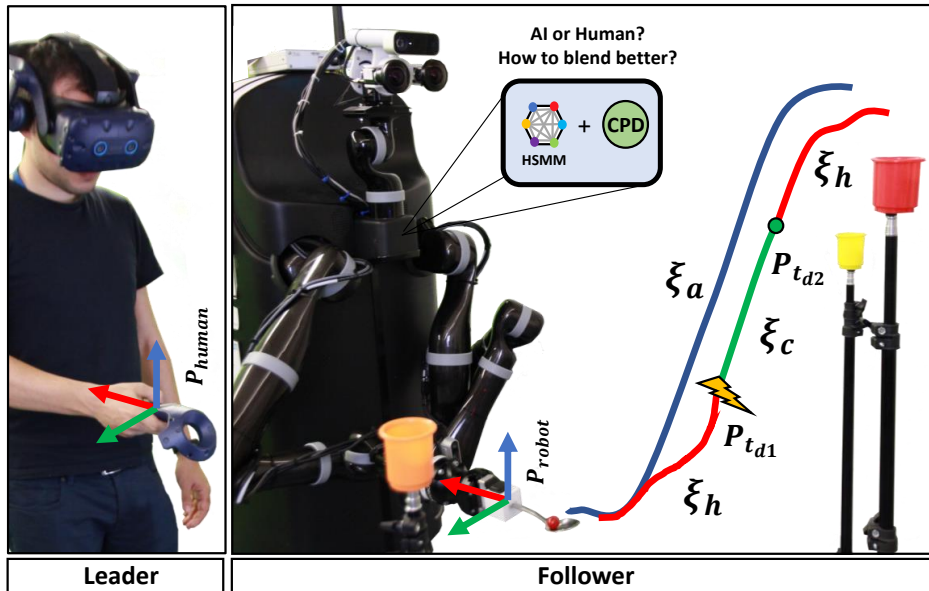


Figure 5.1: The robot-assisted remote feeding scenario is simulated as marble pouring with the Kinova[®] Movo platform using the proposed Skill-CPD for high-level skill refinement during the shared-autonomy control switching. P_{human} is the coordinate frame of the leader (here HTC Vive[®] controller), and P_{robot} is the coordinate frame of the follower. The main objective is to deliver the red marble inside the spoon to one of the three main pouring destinations represented by orange, red, and yellow containers. The robot continuously predicts the operator's intention, and in case of network disconnection, executes the task from the previously learned skill with minimum inconsistency. Red ξ_h denotes the human trajectory, blue ξ_a denotes the learned skill trajectory, green ξ_c denotes the refined trajectory, and P_{td1} indicates the network disconnection point. The network reconnection point is indicated by P_{td2} [9], ©2022 IEEE.

Teleoperation is increasingly applied to assist humans in performing specialized real-life tasks such as remote patient and elderly care or even inspection and exploration in deep space, or underwater missions. However, performing teleoperated manipulation tasks are challenging due to unavoidable network impairments such as delay, jitter, or packet drops. A promising approach to mitigate the aforementioned artifacts is to use shared-autonomy

instead of direct teleoperation [51]. In this approach, the desired trajectory is synthesized according to the recognized intention. In the presence of communication impairments, the system will switch to shared autonomy or, for longer network disconnection to autonomous control mode [51], [65]. In [51], the authors introduced a shared autonomy solution for assistive teleoperation, which has two phases; First, individual skills are trained using Learning from Demonstrations (LfD) and are encoded using Hidden Semi-Markov Models (HSMMs) [52]. Second, the trained model from the first step is used to infer the operator's intention during teleoperation, and the Linear Quadratic Tracker (LQT) is used to synthesize the recognized skill. Similarly, Tian et al. [4] adopted the same approach to predict the operator's intention in an English letter drawing task. However, the stability performance is affected by varying environmental conditions. Shared autonomy suffers from intensive fluctuation when the directly demonstrated teleoperation varies from the learned skill, yielding undefined situations for the robot. This is exaggerated when the system wants to hand over the control to the autonomous mode. This is the major problem, which motivated us to extend the previous solutions by developing a new skill refinement strategy to improve the performance of task reproduction. In order to fit the reproduced trajectory and the demonstrated trajectory better, the skill refinement in our work is formulated as a non-rigid point set registration problem. We propose a real-time skill refinement framework to support the shared autonomy of the robot manipulator. Our trajectory refinement is based on the Coherent Point Drift (CPD) [5] algorithm. Our experimental results reveal that our system improves the performance of the remote manipulation task by mitigating the effect of imprecise movements, especially for the intersection part of the partial demonstration and the reproduction.

5.1 Problem Statement

In shared autonomy, autonomous control and direct teleoperated trajectories are blended with an arbitration strategy. If the network is disconnected, we cannot rely on the user's input, so the system needs to turn on autonomous control. In telerobotics, we cannot run the entire task in autonomous control mode from the beginning because robots cannot yet perform tasks entirely independently due to limited perception or safety regulations. Due to the spatial differences (e.g., scaling, rotation and translation) between the human operator and the remote robot side, if the system switches to the autonomous control from direct control, it cannot guarantee a smooth and reliable transition. This chapter addresses this issue by formulating it as a point set registration problem, which is the process of finding a spatial transformation that aligns two point sets. The mapping may consist of a rigid or non-rigid transformation. For rigid registration, the iterative closest point (ICP) [96] is a promising solution. Since, our skill refinement strategy cannot assume a rigid transformation, we employ a non-rigid registration algorithm known as Coherent Point Drift (CPD) that takes a probabilistic approach to align point sets.

5.2 System Architecture

Our system architecture is shown in Fig. 5.2. The n task-related skills with m points ${}^n\xi_s = \{{}^n s_0, {}^n s_1, \dots, {}^n s_m\}$ are learned with the help of the HSMM beforehand. Point s_i is a 3D vector with $[x, y, z]$ coordinates. The human teleoperator performs the manipulation task of a trajectory $\xi_h = \{h_0, h_1, \dots, h_t, \dots\}$ via a communication network, while the trajectory-based intention is being recognized. Our system performs direct teleoperation using direct trajectory commands ξ_h to control the remote robot when the network is available and stable. A network discontinuity detector is defined to monitor the quality and reliability of the communication network. A network disconnection is detected by the deadline policy, which is triggered in case of no packet allocation within every $\lambda_{deadline}$ ms. This policy is regardless of the connection type. Considering that $\lambda_{deadline}$ denotes the threshold for detecting the network disconnection and $\lambda_{minconf}$ indicates the threshold for HSMM recognition accuracy, Skill-CPD switches to the autonomous control mode once the $\lambda_{deadline}$ and $\lambda_{minconf}$ are satisfied at disconnection time t_d , meaning that network disconnection is detected and the relative task trajectory is known. The system first generates the skill trajectory (autonomous mode trajectory) $\xi_a = \{a_0, a_1, \dots, a_m\}$ using LQT according to the recognized intention. Second, it is able to improve the performance of the reproduced trajectory ξ_a by registering it with the direct teleoperated trajectory ξ_h using the CPD algorithm and yield the aligned trajectory $\xi_c = \{c_0, c_1, \dots, c_t, \dots\}$. Ultimately, the non-linear model predictive control-based motion planner (NMPC-MP) receives the final refined trajectory $\xi_{ee} = \{h_0, h_1, \dots, h_{t_d}, c_{t_d+1}, \dots\}$ and generates the velocities $\xi_v = \{v_0, v_1, \dots, v_t, \dots\}$ in real-time for the robot joints. NMPC-MP is the trajectory-level motion planner for the Jaco-2 manipulator introduced in Chapter 4. The arbitration switch illustrated in Fig. 5.2 is implemented as a combination of the direct and the autonomous control signals u_h and u_a as follows [63]:

$$\mathcal{H}(u_h, u_a) = \alpha u_h + (1 - \alpha) u_a, \quad (5.1)$$

where u_h and u_a indicate in our case the trajectory commands ξ_h and ξ_a respectively. $\alpha \in \{0, 1\}$ denotes the weight of the direct control from the human and can be described as:

$$\alpha = \begin{cases} 0, & ID(\lambda_{deadline}) \cap HSMM(\lambda_{minconf}), \\ 1, & otherwise. \end{cases} \quad (5.2)$$

5.3 Technical Details on Skill-CPD

5.3.1 Hidden Semi-Markov Model (HSMM)

To recognize the intention of the human teleoperator during task execution, we adopt HSMM to encode and decode both temporal and spatial information [4], [51], [27]. First of all, the task trajectory is demonstrated n times and the Expectation-Maximization (EM) algorithm is applied for training the HSMM parameterized by $\theta = \{\Pi_i, \{a_{i,j}\}_{j=1}^K, \mu_i, \Sigma_i, \mu_i^S, \Sigma_i^S\}_{i=1}^K$. We define $h_t \in \{1, 2, \dots, K\}$ as the hidden state at time t which is represented by a multivariate Gaussian and thus a Gaussian Mixture Model (GMM) with parameters $\{\mu_i, \Sigma_i\}_{i=1}^K$ is formulated by K hidden states.

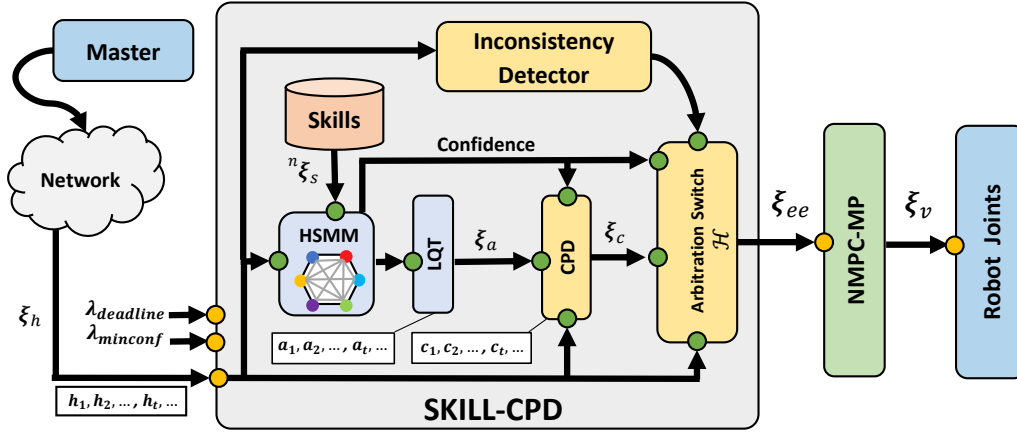


Figure 5.2: Skill-CPD System Diagram. ©2022 IEEE.

Fig. 5.3 illustrates how the GMM clusters segment the whole demonstrated trajectory for a 2D and a 3D scenario. During the task execution, the directly teleoperated trajectory $\xi_h = \{h_1, \dots, h_t, \dots\}$ is transmitted to Skill-CPD and the recognition is carried out by calculating and comparing the probability of the trajectory point $h_t \in \xi_h$ to be in the hidden state at time t . More details can be found in [4], [51] and [27].

5.3.2 Linear Quadratic Tracker (LQT)

We synthesize the trajectory of the recognized task using a LQT, which is a commonly used tool from control theory to infer an optimal control policy for manipulation tasks [27], [22]. By minimizing a scalar cost function we obtain the optimal control policy $u_{a,t}$ at each time step and thus the reproduced trajectory $\xi_a = \{a_1, \dots, a_t, \dots\}$. More details are introduced in [27] and [22].

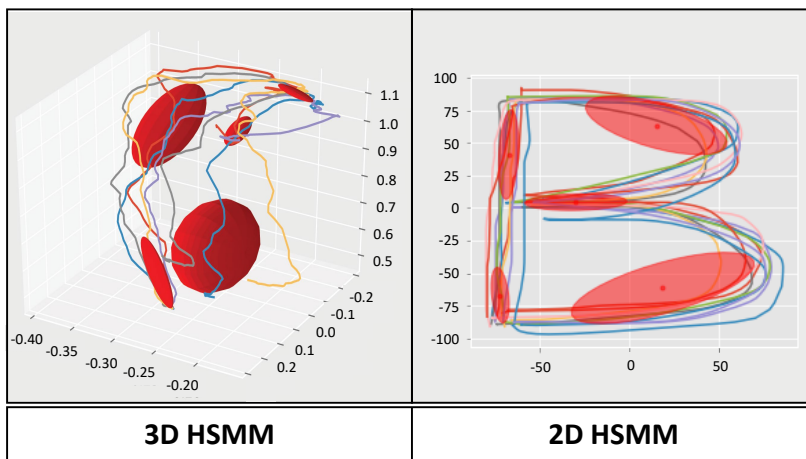


Figure 5.3: The picture on the right illustrates the ten demonstrated trajectories of the example letter B for the 2D scenario (in pixel). The picture on the left shows the five demonstrations of the marble pouring task to the red container for the 3D scenario (in meter). The HSM clusters are shown in red. ©2022 IEEE.

5.3.3 Coherent Point Drift (CPD) Algorithm

After receiving the reproduced trajectory from LQT, the CPD algorithm is applied to refine it by registering the trajectory ξ_a to the ξ_h . The CPD algorithm is known as a robust probabilistic multi-dimensional point sets registration algorithm for both rigid and non-rigid transformations [5]. Gaussian Mixture Models (GMM) and Expectation Maximisation (EM) are applied to detect point cloud changes. In contrast to the ICP, which minimizes distances between points, it maximizes the probability of each source point given a set of Gaussian's centered at the target points.

We introduce briefly the general methodology according to [5]. Considering two point set $\mathbf{X}_{N \times D}$ and $\mathbf{Y}_{M \times D}$, the objective is to find an optimal transformation $\mathcal{T} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ with transformation parameters θ_T that aligns \mathbf{Y} with \mathbf{X} , which can be converted to a probability density estimation problem. The points in \mathbf{Y} are viewed as the Gaussian Mixture Model (GMM) centroids and the points in \mathbf{X} represent the target points generated by the GMM. The GMM probability density function $p(\mathbf{x})$ is expressed as:

$$p(\mathbf{x}) = \sum_{m=1}^{M+1} P(m)p(\mathbf{x}|m), \quad (5.3)$$

with the the Gaussian distribution centered on point $\mathbf{y}_m \in \mathbf{Y}$:

$$p(\mathbf{x}|m) = \frac{1}{(2\pi\sigma^2)^{D/2}} \exp -\frac{\|\mathbf{x} - \mathbf{y}_m\|_2}{2\sigma^2}. \quad (5.4)$$

In order to account for noise and outliers that could appear in the point sets, an additional uniform distribution $p(\mathbf{x}|M+1) = \frac{1}{N}$ with the weight parameter $0 \leq w < 1$ is also considered in the mixture model, which represents that the probability of a point $\mathbf{x}_n \in \mathbf{X}$ is generated by one of the M Gaussian components is uniform. The membership probabilities $P(m) = \frac{1}{M}$ and the isotropic covariances σ^2 are equal for all Gaussian components $m = 1, \dots, M$. Therefore, the mixture model can be expressed as:

$$p(\mathbf{x}) = w \frac{1}{N} + (1-w) \sum_{m=1}^M \frac{1}{M} p(\mathbf{x}|m). \quad (5.5)$$

The correspondences of the two point sets can be optimized by maximizing the GMM posterior probability for the given point in \mathbf{X} and converted to the minimization problem of the negative log-likelihood function:

$$\mathcal{L}(\theta_T, \sigma^2) = - \sum_{n=1}^N \log \sum_{m=1}^{M+1} P(m)p(\mathbf{x}|m). \quad (5.6)$$

The EM algorithm is applied for optimizing θ_T and σ^2 : In the E-step, the posterior probability distribution with the "old" parameter θ_T^{old} is calculated according to Eq. 5.4 and 5.5. In the M-step, the "new" parameter θ_T^{new} is then found by solving the minimization problem of the cost function \mathcal{Q} which is the expectation of the negative log-likelihood function \mathcal{L} in Eq. 5.6.

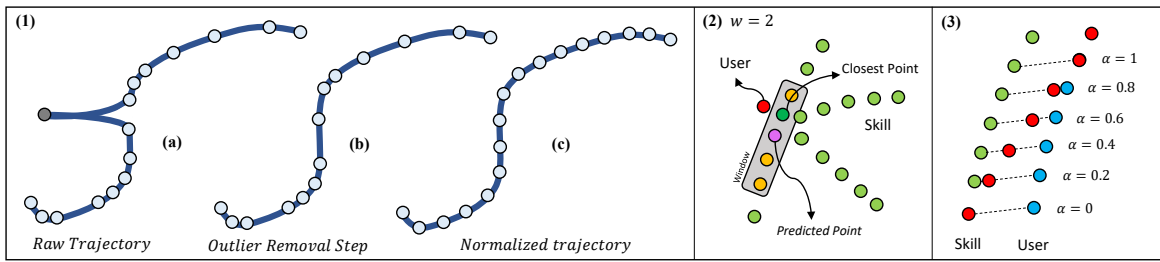


Figure 5.4: (1.a) is the raw trajectory from the human operator which could contain outliers and noise. (1.b) indicates the trajectory after outlier removal and (1.c) denotes the final normalized trajectory. (2) illustrates the closest point detection strategy. (3) shows the reverse takeover step as a linear interpolation approach.

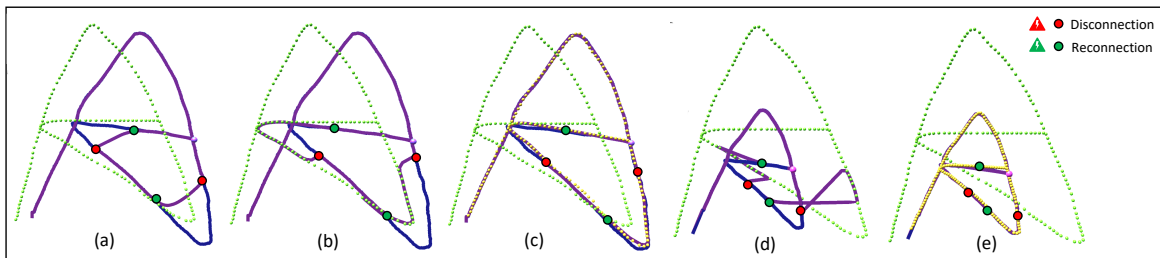


Figure 5.5: The simulation experiment for the task(i): 2D English letter drawing. The green dotted trajectory denotes the learned skill. Blue continuous trajectory is the desired input from the human operator (ground truth). The purple continuous trajectory is the system final output. (a) is the normal system without shared autonomy (direct teleoperation). (b) and (d) are the standard shared autonomy systems with rigid and non-rigid deformations. (c) and (e) represent our Skill-CPD system with same deformations. Yellow dotted trajectory is the registered skill.

After reaching the minimum of the cost function Q , we acquire the optimal transformation parameters θ_T and thus the aligned trajectory point set:

$$\mathcal{T} = \arg \min_{\mathcal{T}} Q(\theta_T, \sigma^2). \tag{5.7}$$

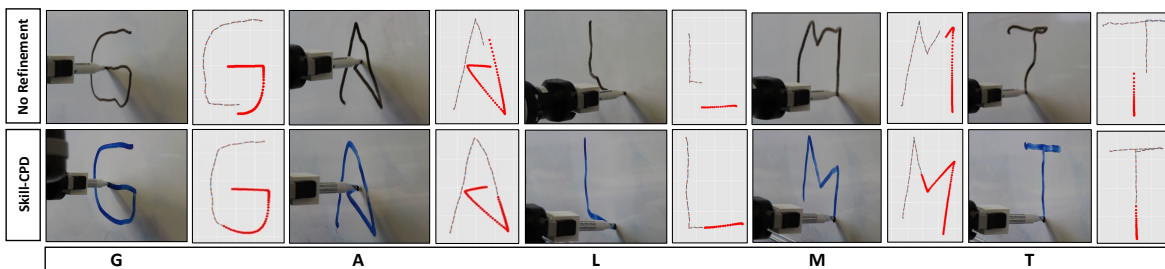


Figure 5.6: The real experiment for the task(i): 2D English letter drawing. The gray dotted trajectory denotes the direct teleoperation and the red dotted trajectory denotes the autonomous controller output. In the second row, we applied Skill-CPD whereas in the first we didn't.

Algorithm 2: Skill-CPD

```

 $\xi_h \leftarrow \{h_0, h_1, \dots, h_t\}; \xi_a \leftarrow \{a_0, a_1, \dots, a_t\};$ 
Current_step  $\leftarrow 0$ ; state  $\leftarrow$  "Direct";
if time  $\geq$  next_time then
  next_time = time +  $\Delta t$ ;
  d  $\leftarrow$  Dist( $h_t$ , old_desired_robot_position);
  if d >  $\Delta d$  then
    desired_robot_position =  $h_{t_d}$ ;
    if ID( $\lambda_d$ ) & HSMM( $\lambda_m$ )  $\neq$  True then
      if state = "Direct" then
         $\alpha \leftarrow 1$ ;
        robot_position  $\leftarrow$  desired_robot_position;
         $\xi_h^* \leftarrow$  robot_position;
      end
      if state = "Skill" then
        robot_position  $\leftarrow$  (1 -  $\alpha$ ) *  $\xi_c$ [current_step] +  $\alpha$  * desired_robot_position;
         $\alpha \leftarrow \alpha + 0.1$ ;
        if  $\alpha = 1$  then
          | state  $\leftarrow$  "Direct"
        end
      end
      current_step  $\leftarrow$  current_step + 1;
    end
  else
    if state = "Direct" then
      | state  $\leftarrow$  "ToSkill";
      | DoRegistration();
    end
    else if state = "Skill" then
      |  $\alpha \leftarrow 0$ ;
      | robot_position  $\leftarrow$   $\xi_c$ [current_step];
      |  $\xi_h^* \leftarrow$  robot_position;
      | current_step  $\leftarrow$  current_step + 1;
    end
  end
end
end

```

Algorithm 3: DoRegistration

```

 $\xi_c \leftarrow$  DoRigidCPD( $\xi_h^*$ ,  $\xi_a$ );
error  $\leftarrow$  Dist( $\xi_h^*$ ,  $\xi_c$ ); w  $\leftarrow 10$ ; i  $\leftarrow 0$ ;
if error <  $\Delta e$  then
  | i  $\leftarrow$  FindClosestStep(robot_position,  $\xi_c$ , w);
end
else
  |  $\xi_c \leftarrow$  DoNonRigidCPD( $\xi_h^*$ ,  $\xi_a$ );
  | i  $\leftarrow$  FindClosestStep(robot_position,  $\xi_c$ , w);
end
current_step  $\leftarrow$  i;
state  $\leftarrow$  "Skill";

```

5.3.4 Skill Registration

Considering the directly teleoperated trajectory $\xi_h = \{h_1, \dots, h_N\}$ and reproduced LQT trajectory $\xi_a = \{a_1, \dots, a_M\}$ in D dimensions, the objective of Skill-CPD is to find an optimal transformation \mathcal{T} with transformation parameters θ_T that aligns ξ_a with ξ_h . The transformation $\mathcal{T}(\xi_a, \theta_T)$ can be specified for different cases. We only consider the situation of $D = 3$ for 2D and 3D scenarios in our experiments and keep the z -coordinate constant for writing letters on a whiteboard in the 2D case. As shown in Eq. 5.8, the rigid registration is used in case only rotation and translation are considered, whereas the non-rigid registration is used when a non-linear transformation is involved. Since the non-rigid registration is computationally demanding, we apply the Fast Gauss Transform (FGT) [97] in the CPD algorithm to reduce the computational complexity from $o(MN)$ to $o(M + N)$ for accelerating the registration process. Thus, the registration can be conducted in real-time. (See Fig. 5.9).

$$\mathcal{T}(\xi_a; \theta_T) = \begin{cases} \mathbf{R}a_m + \mathbf{t}, & \text{Rigid} \\ \xi_a + \mathbf{G}\mathbf{W}, & \text{NonRigid} \end{cases} \quad (5.8)$$

5.3.4.1 Trajectory Sampling and Step Indicator

The update rate of our system is $\Delta t = 33ms$. To normalize the task trajectories, point P_{robot} will be valid as long as the Euclidean distance from the previous point exceeds the threshold $\Delta d = 0.6cm$. We call this the normalization step which can determine the total number of points in the trajectory (See Fig. 5.4 and Fig. 5.9). We carry out the sampling step on the learned skill ξ_a and the input trajectory ξ_h to eliminate the redundant points and convert the continuous trajectory point set to a discrete finite point set. As illustrated in Fig. 5.4, we define the skill as a specific temporal and spatial trajectory that the robot end-effector performs. Meanwhile, the velocity pattern that the human operator executes during teleoperation is assumed to be constant, as it was with the primary skill learned from HSMMs, and can only be scaled slightly if the entire trajectory is scaled ($\pm 1.3x$). Otherwise, the accuracy of intention recognition could be significantly lowered. HSMM-based approaches are limited in this respect. Although the CPD algorithm can be applied for extremely non-rigid deformations ($\gg \pm 1.3x$), it's still not supported in our system.

5.3.4.2 Adaptive Registration and Switching Strategy

Assuming that the network is disconnected and the predicted skill from HSMMs is correct, our system will activate the control switching strategy and the correspondence disconnection point index in the skill can be determined roughly. First, we consider that the trajectory from the human operator is only shifted or rotated compared to the learned skill. We split the skill from the beginning to the detected disconnection index and register these two point sets using the rigid registration. After finding the appropriate transformation \mathcal{T} , we apply it to the whole skill trajectory. Then we calculate the Euclidean distance between each point of the robot's trajectory with the registered trajectory and determine the error E . If E is less than the threshold $P_{rigid_error} = 0.1cm$, the rigid registration is considered to be valid and

switch to the skill. Because we cannot guarantee that the detected disconnection index on the skill is the closest point, the closest point search strategy is applied. To find that, we use a window with a size of $w = \pm 10$ steps (see Fig. 5.4) and select the closest point according to the Euclidean distance between all points $\mathbf{a}_t \in \xi_a$ and the point \mathbf{h}_{t_d} :

$$\mathbf{a}_{t_d} := \arg \min_{\mathbf{a}_t \in \xi_a} \|\mathbf{a}_t - \mathbf{h}_{t_d}\|_2. \quad (5.9)$$

In case the registration error P_{rigid_error} in the previous step is not satisfied, we apply the non-rigid registration strategy. Since the non-rigid registration is computationally heavier, we always perform the rigid registration first and suppose the human operator's trajectory is only shifted or rotated. In non-rigid registration, the only difference is that we apply the complete skill registration without splitting it. Non-rigid registration will automatically shift all the skill points coherently to keep the skill's consistency. Similar to the rigid version, we will apply the closest point search strategy and then switch to the skill. Finally, our system switches to the registered skill points and continues the task trajectory without obvious displacement.

5.3.4.3 Reverse Takeover Strategy

When the network is connected again, the human operator takes over the control of our system and could perform the task with a slightly different relative velocity during the disconnection period. Therefore, we can not hand over the control immediately. To tackle this, our system applies the trajectory interpolation strategy to slowly switch from autonomous control mode to the direct teleoperation. The parameter α in arbitration Eq. 5.1 is set to 0 and will be ascended in 10 steps to hand over the control to the human operator smoothly (see Fig. 5.4). For each step, we increase α by:

$$\alpha = \alpha + 0.1. \quad (5.10)$$

5.4 Experimental Evaluation

5.4.1 Experimental Setup

The real experiments are running on a Intel[®] Core-i7[®] CPU with 8 cores at 2.80 GHz. The software is running on Ubuntu[®] 18.04 LTS and ROS Melodic. Besides, the HTC Vive[®] HMD and VR controllers for 3D experiments and the Phantom-Omni[®] haptic master device for 2D experiments both simulation and real are used and connected to ROS from Unity3D[®] via [80]. All scenarios have been tested using the Kinova[®] Movo platform. In addition, for 3D robot-assisted feeding the marbles and containers are adapted from the YCB real-world object dataset [82]. For the following experiments, unless specially mentioned, parameters for Skill-CPD follow Table 5.2. For the CPD algorithm the internal parameters were $\beta = 3$ and $\lambda = 3$. We tested our solution over various connection types: LAN, WiFi (considered as the main interface in this dissertation) and even 5G over realistic satellite link emulated connection `calvo2020optical`.

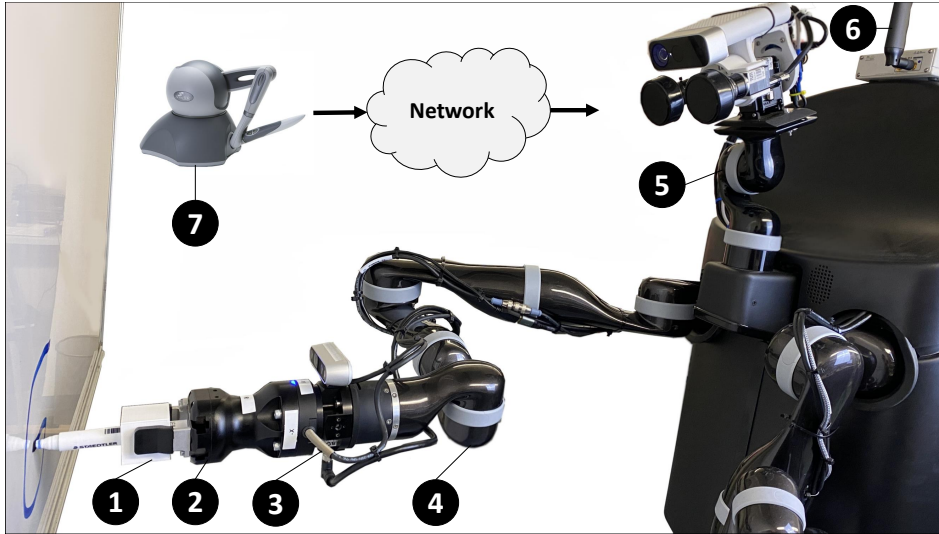


Figure 5.7: Closeup view of the 2D English letter drawing. We apply force feedback control to keep the marker on the whiteboard. 1) the 3d-printed marker holder, 2) Robotiq[®] Hand-e gripper 3) Botasys[®] SensONE force/torque sensor 4) Kinova[®] Jaco-2 manipulator 5) Pan/Tilt/Roll unit 6) Network interface (5G UE) 7) Phantom-Omni[®] haptic master.

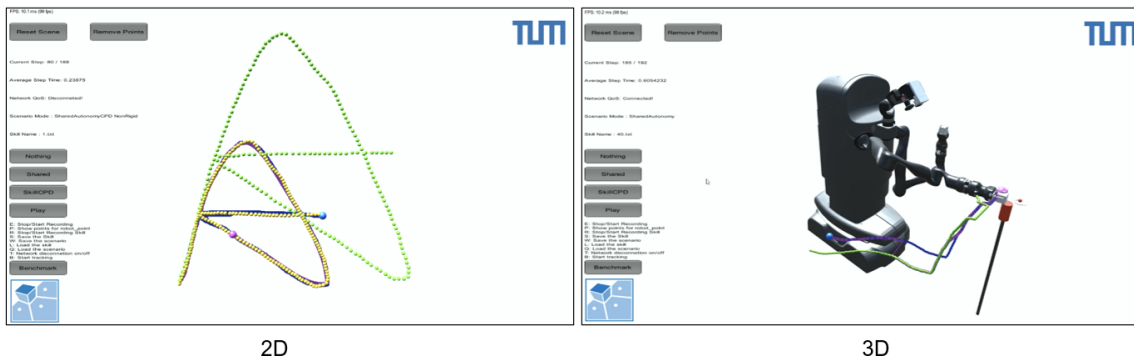


Figure 5.8: Motion generation and dataset creation environment.

5.4.2 Task(i): 2D English Letter Experiments in Simulation

We used the dataset from [4] and performed several automated experiments to benchmark the performance of the proposed approach and make it reproducible. For each letter we applied ± 10 cm shifting, ± 30 degree rotation and $\pm 1.3x$ scaling. Overall, eight variations are considered. We have simulated the task executions using the Unity3D engine. After 40 percent of task execution, we disconnected the network randomly to make sure the letter detection confidence meets the minimum threshold of 0.9. The communication network between the human operator and the remote robot is disconnected randomly and reconnected with random intervals. The maximum disconnection period was 5 seconds. Fig. 5.4 illustrates the letter A in this experiment with different modes. We can observe the dramatic trajectory consistency improvement for rigid and non-rigid scenarios.

Table 5.1: Performance Analysis in Simulation and Real Experiments

Task	Experiment	Average Error (cm)	No Shared Autonomy	Standard Shared Autonomy [51], [4]	Skill-CPD
2D	(1.1)	Max Displacement (Identical Skill)	9.98	0.90	0.82
2D	(1.2)	Trajectory RMSE (Identical Skill)	3.74	0.0020	0.0010
2D	(1.3)	Max Displacement (± 10 cm translated)	9.95	14.42	0.85
2D	(1.4)	Trajectory RMSE (± 10 cm translated)	3.76	2.96	0.0014
2D	(1.5)	Max Displacement (± 30 deg. rotated)	9.95	14.05	0.85
2D	(1.6)	Trajectory RMSE (± 30 deg. rotated)	3.73	1.86	0.0012
2D	(1.7)	Max Displacement (± 1.3 x scaled)	4.23 to 11.94	7.15 to 15.63	0.83
2D	(1.8)	Trajectory RMSE (± 1.3 x scaled)	1.20 to 5.75	2.13 to 5.12	0.0013
2D	(2)	Max Displacement (± 10 cm translated)	7.96	12.67	0.97
2D	(2)	Trajectory RMSE (± 10 cm translated)	5.98	4.73	0.023
3D	(3)	Max Displacement (± 15 cm translated)	20.93	14.58	1.12
3D	(3)	Trajectory RMSE (± 15 cm translated)	12.34	9.76	0.56

Table 5.2: Parameters for Skill-CPD for task(i), task(ii), and task(iii)

Task	Experiments	Demonstration Number n	Normalization Samples (Average)	Task Length (s)	$\lambda_{deadline}$	$\lambda_{minconf}$	Δt	Δd	Hidden States Number K
2D Sim	624 (3x26x8)	10	160	20	50 ms	0.9	33 ms	0.6 cm	5
2D Real	78 (3x26)	10	200	40	50 ms	0.8	33 ms	0.6 cm	5
3D Real	9 (3x3)	5	200	30	50 ms	0.8	33 ms	1 cm	5

5.4.3 Task(ii): 2D English Letter Experiments in Reality

We define the manipulation task in the 2D scenario as drawing an English letter $l \in \{A, B, \dots, Z\}$ on the whiteboard. The objective of this task is to draw the desired letter with a marker in a smooth and reliable manner even if the communication network becomes unstable or interrupted during direct teleoperation.

To guarantee that the marker held by the robot manipulator is in contact with the whiteboard, force feedback-based control is applied (see Fig. 5.6). In this 2D case, we consider the consistency and smoothness of the whole trajectory more important than reaching the same remaining target points. Skill-CPD maps the most likely part of the LQT reproduction to the direct teleoperated trajectory part by finding the closest point of the LQT trajectory to the current trajectory. (as illustrated in Fig. 5.4).

5.4.4 Task(iii): 3D robot-assisted Feeding in Reality

As a 3D scenario, we select feeding with a spoon to elderly who are not able or hard to move. The goal of this task is to pour the food, which we replace with marbles, into the desired container. Only slight displacements are allowed to make sure that the marble could not fall down. We use the orientation parameters from our previous demonstrations to accomplish the pouring action. Fig. 5.11 illustrates the trajectories for each scenarios during the task execution.

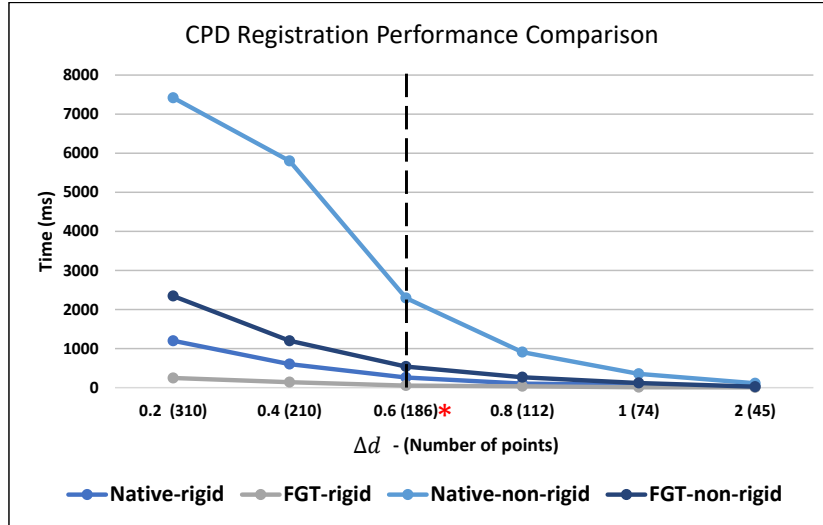


Figure 5.9: The effect of Δd selection on CPD real-time performance

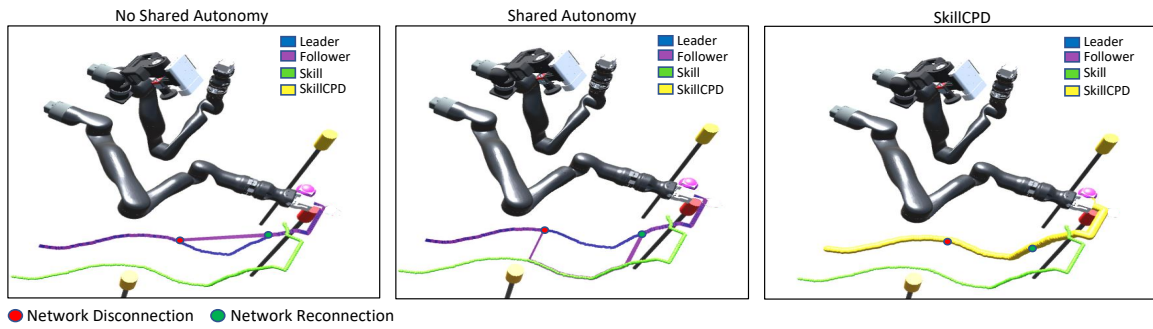


Figure 5.10: The real experiment for the task(iii): 3D robot-assisted feeding. It illustrates the different scenarios from robot perspective.

5.4.5 Discussion

We summarize the quantitative results of our experiments in Table 5.1. In both 2D and 3D experiment we measure the maximum displacement error, which is the Euclidean distance between points in the trajectory and it can indicate the disconnection point $h_{t_d} \in \xi_h$, greater than Δd . Additionally, the trajectory Root Mean Square Error (RMSE) is calculated by summation of the Euclidean distance between individual points of the leader (ξ_h) and the follower (ξ_{ee}). We can generally see that Skill-CPD improves the performance by mitigating the displacement error in both 2D and 3D experiments. The real-time performance of the CPD algorithm is affected by the number of points which as trade-off ($\Delta d = 0.6cm$) was the best parameter in our experiments. (See Fig. 5.9).

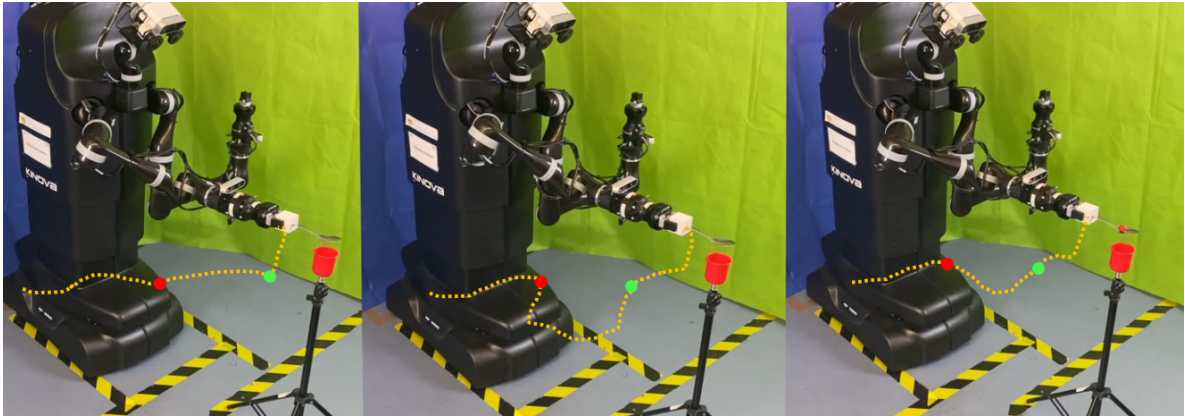


Figure 5.11: The real experiment for the task(iii): 3D robot-assisted feeding. It illustrates the different scenarios from robot perspective. The red marble (as a liquid handling task) for Skill-CPD remains in the container after the network impairment but due to the acceleration caused by the displacement error, it falls in other modes.

5.5 Chapter Summary

Advanced wireless communication networks provide lower latency and a higher transmission rate. Although this is an enabler for many new teleoperation applications, the risk of network instability or packet drop is still unavoidable. Real-time manipulator teleoperation requires data transmission with no discontinuity. Shared autonomy (SA) is a standard method to mitigate this issue. In this way, if the data from the remote side is unavailable, the controller can continue based on the previously observed models. However, due to the spatial gap between human and robot trajectories, indisputable fluctuations occur, which cause issues in teleoperation applications. This motivates us to propose a new skill refinement strategy to modify the previously trained skill and mitigate the sudden unwanted motions within the control takeover phase. To this end, our approach comprises applying the Hidden Semi-Markov Model (HSMM) and Linear Quadratic Tracker (LQT) in combination to learn and predict the user's intentions and then exploiting Coherent Point Drift (CPD) to refine the executable trajectory. We test our method both in simulation and in the real world for 2D English letter drawing and 3D robot-assisted feeding scenarios. Our experimental results using the Kinova[®] Movo platform show that the proposed refinement approach generates a stable trajectory and mitigates the control switching inconsistency.

Chapter 6

Liquid Pouring Through Curriculum and Curiosity-based Reinforcement Learning

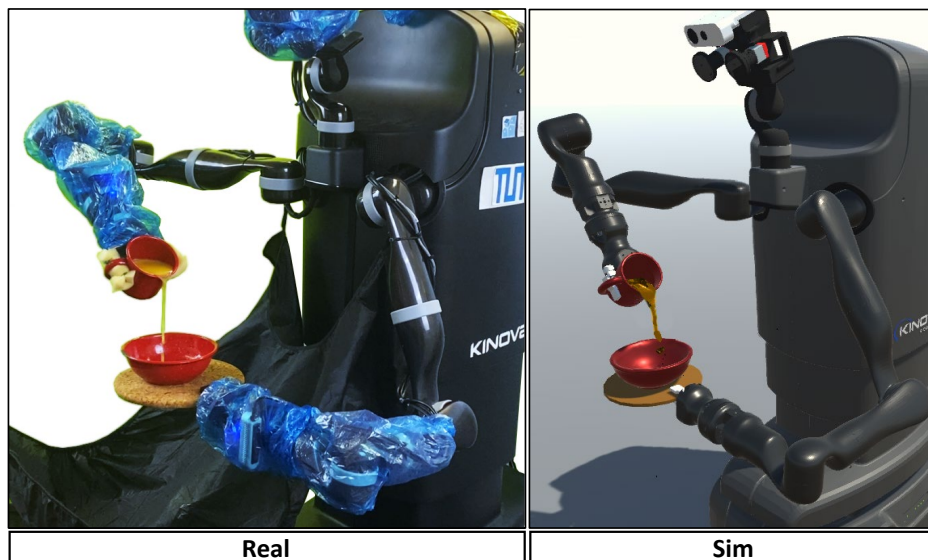


Figure 6.1: On the right, the simulation environment is illustrated during the pouring of a water-like liquid. On the left, the robot is pouring orange juice in a real environment using our trained policy network, PourNet. PourNet learns to perform natural human-like pouring actions through several simulated experiences [8]. ©2022 IEEE.

Our ability to accurately pour liquids into containers might seem trivial to us. On the contrary, such an ordinary interaction is one of the most challenging tasks in Robotics. With automation entering more and more aspects of our professional and private life, pouring liquids is becoming an indispensable skill for robots. The potential of using precision pouring robots is immense in the catering/hospitality industry. Likewise, in service robotics, precision pouring as a skill will make robots valuable kitchen assistant robots. Furthermore, robots can aid as drinking and eating assistants for motor-impaired patients or the elderly.

Given the complexity of the fluid dynamics and the limited degrees of freedom in robots, formulating a direct control mechanism for pouring is difficult. For example, solving the Navier-Stokes [98] formulation to predict the behaviour of liquids in 3D is still an open mil-

lennium prize problem. Therefore, most of the recent works simplify the state space or the action space of the pouring problem. This results in failure under real world scenarios and to the best of our knowledge, there hasn't been any robust controller that can handle the robot pouring scenario using different liquids. Recent progress in deep reinforcement learning (RL) [99] has brought promising results for controlling agents in complex environments with large state or action spaces. This motivated us to solve the robotic pouring using deep reinforcement learning. One of the challenges in applying RL to robotics problems is that acquiring a large number of experiences from the real world can be time and cost expensive. Therefore, often it is more beneficial to carry out the training in a simulation environment and later transfer the skill to the real world. To this end, we implemented a simulation environment for the interaction between robots and the liquids. Through extensive real-world and simulated experiments, we design the optimized state-space, action-space, and the reward functions. As a result not only the robot can easily learn the tasks within the simulation environment, but the transition from simulation to real goes smoothly.

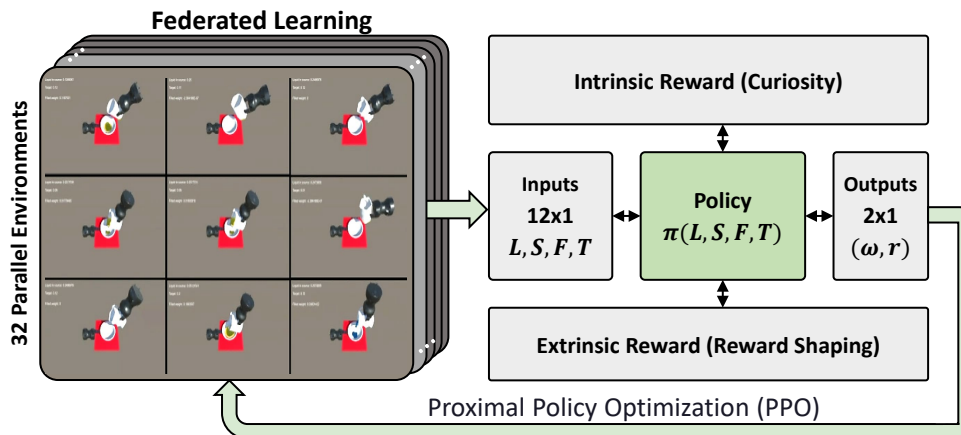


Figure 6.2: Training Architecture. ©2022 IEEE.

6.1 Liquid Property Awareness

The motion of the fluid is largely described by the Navier-Stokes equations [98]. These equations make up the bulwark of the Fluid Mechanics Theory with a great importance in the science and engineering. However, the theoretical understanding of their solutions is incomplete. Particularly, for the three-dimensional system of equations, and given some initial conditions, existence of smooth solution is yet to be proven. This makes the “Navier-Stokes existence and smoothness problem” one of the worthy “Millennium Prize problems” in the field of Mathematics [100]. With a complete solution of the Navier-Stokes equation, a revolution in motion and trajectory prediction is expected. However, till then, it's expected that an intelligent liquid pouring agent has an abstract understanding of the liquid properties relevant for precision pouring. Some of these properties are described below (as shown in Figure 6.3):

- *Adhesion*: This property of fluids makes them stick to the surface of containers. This behaviour emanates because of the attractive electromagnetic and van der Waals forces between liquid and container solid particles [101]. This is visible to a greater extent in

fluids like Ketchup, Marmalade, etc. The adhesive forces are always perpendicular to the surface of solids directed into the solid [102]. This is visualized in the Figure 6.3.

- *Viscosity*: This property of fluids characterizes the degree of the internal friction in the liquids [102]. This internal friction arises because of the resistance between two layers of the liquid while moving relative to each other. Hence, viscosity results in loss of the Kinetic energy of the fluid in form of internal energy. The viscosity in whole is one of the important factors deciding the velocity field of the fluid in motion. As a result, the liquids with high viscosity tend to flow slowly and have less volumetric flow rate per unit time. This can be seen in Figure 6.3.
- *Cohesion*: Cohesion is the intermolecular attractive force between the molecules of the same kind or phase as seen in the Figure 6.3. The liquid is held together by the cohesive forces between it's molecules. The imbalance of cohesive forces with the molecules on a liquid's surface try to minimize the surface area of the liquid which manifests in macro-world property of *Surface Tension*.
- *Surface Tension*: This is the fluid property observed on the fluid's surface due to imbalance of cohesive forces between the fluid's molecules on the surface when compared to the molecules remaining in the bulk of the liquid [102]. Surface tension causes the water to form small droplets at the lip of the container while pouring as seen in the Figure 6.3.
- *Density*: Density of the fluid is defined as the mass a fluid has per unit volume ($\rho = \frac{m}{V}$). This has a direct implication in terms of expected mass a liquid will occupy for an agent to apply suitable force-torque in order to resist gravity and also to control the dynamics of the pouring motion.

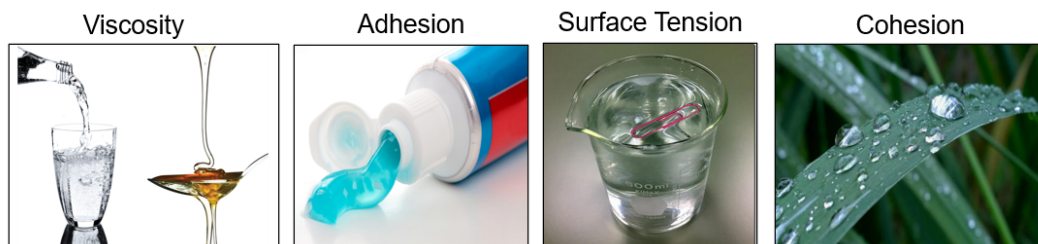


Figure 6.3: Comparison of flow rates of low viscous fluids with the high viscous fluids. [103] [104] [105] [106].

6.2 Operating Environment Awareness

For a precision pouring task, the underlying mechanism must rely on some methods to perceive it's environment. This can be a machine vision-based, audition-based, force-torque sensor-based or a closed-loop control mechanism-based methodology. Each approach can be thought of as a digital adaptation of sensory organs that animals use to make sense of

their environment. These methodologies can either be used independently or in a quorum to enable a pouring agent to make sense of its working environment. Each approach, however, comes with limitations. For instance, it is hard to purely use vision as classification of liquid pixels is tricky and getting a dataset to model this behaviour might be expensive to acquire. Likewise, using purely audition-based approach would require a large dataset of pouring sounds with added noises and again expensive to acquire. However, each approach also comes with a unique set of features that are relevant for precision pouring task. As a result, to have an awareness of the operating environment using sensor-based systems is indispensable for an agent to pour liquids. These can then be used to ensure a generic pouring policy which works on any liquid and any container type. This dissertation relies on force-torque sensor feedback as a fundamental approach in a Reinforcement Learning setup. So, in summary, to automate precision pouring tasks, the agent should know the fluid, the vessel geometry, and the environment in which it is working. The design of PourNet is based on this consideration. The agent trains in a simulation environment where it knows exactly the properties of the fluid, the geometry of the containers, and the environment.



Figure 6.4: Different types of containers with variation in geometrical features. [107]

6.3 Pouring Container Geometry Awareness

The containers used for keeping liquids can come in a vast variety of shapes and sizes as shown in the Figure 6.4. Hence, to be sensitive towards the geometry of the containers is a property that can improve the precision of pouring task. The pouring profile of the liquid is vastly affected by the angle of tilt of the pouring container and its inherent geometry. Dong et al. [67] describe one such scenario where the liquid volume is estimated by integrating the cross-sectional areas of slices a container is divided into from its base to the liquid surface's height by factoring in the angle of the tilt of the container.

6.4 Problem Statement

PourNet is a continuous control RL model for liquid pouring. Given a set of observations in an operating environment, PourNet generates actions from a policy that is trained to maximize its cumulative reward (Figure 6.2). This can be described as:

Set of observations:

\mathcal{L} : Set of liquid features,

S : Set of state information,

F : Feedback measurements from the environment,

T : Desired liquid to be poured.

Set of actions:

ω : Pouring Container’s rotational velocity,

r : Pouring Container’s translational velocity.

Such that,

$$(\omega, r) = \pi(\mathcal{L}, S, F, T) \quad (6.1)$$

Where, π is the policy and *Success Criteria* will be satisfied if the poured quantity is within some tolerable deviation of T .

6.5 Technical Details on PourNet

The success criteria described in Section 6.4 makes precision pouring a “Sparse Reward Reinforcement Learning” problem. This is attributed to the fact that the agent can only receive reinforcing rewards from the environment, if the poured quantity in the receiving container is within some tolerable deviation. From the technical point of view, this chapter proposes *PourNet* as an intelligent agent which is capable of pouring a liquid with liquid property awareness, container geometry awareness and environment awareness using force/torque based perception.

6.5.1 Average Pouring Error

We introduce “**Avg. Pouring Error**” as a performance metric. This is defined as the average pouring deviation over n successfully completed pouring episodes. We use Avg. Pouring Error to evaluate the agent’s progress during training. Therefore, we design the extrinsic rewards of an agent as a function of Avg. Pouring Error such that the agents with a tolerable average pouring errors are rewarded additionally to the consequence of the actions taken during a single episode. This allows us to also use this metric for hyperparameter tuning such that the average pouring error does not increase many-fold over time.

6.5.2 Proximal Policy Optimization

The RL algorithm considered in this chapter is based on Proximal Policy Optimization (PPO) [34]. The choice of PPO was based on a training experiment in the simulation setup using a single liquid profile. For this experiment, it was observed that the PPO observed a

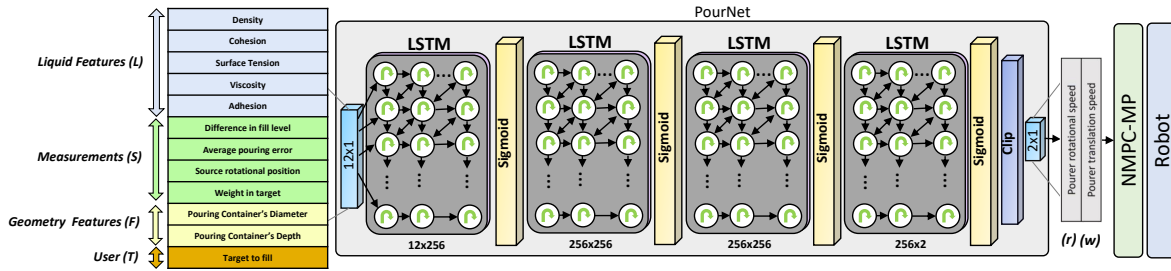


Figure 6.5: PourNet Model Architecture.

better average pouring error when compared to the Soft-Actor-Critic (SAC) [35] based RL algorithm.

6.5.3 Action Handler

PourNet’s policy described in Section 6.4 outputs two floating number values, i.e. translation and rotation speed; therefore, the action handler of the agent is designed to reflect these values in terms of 6D poses and 3D motions. In Algorithm 4, the chapter discusses the action handler, which converts the PourNet’s actions to the subsequent 3D motions. The training of PourNet is always **without** the real robot in the loop. However, for transferring the policy to a robot, the pouring container’s pose essentially becomes the manipulator’s end-effector. This is because the pouring container is held within the grippers, and hence, becomes the end-effector for the motion-planner to execute the actions. Ultimately, the Non-linear Model Predictive Control-based Motion Planner (NMPC-MP) introduced in Chapter 4 receives the final trajectory.

6.5.4 Reward Shaping

As established in the introduction of Section 6.5, the PourNet’s RL problem essentially is a Sparse Reward problem. This has a great implication on the agent to train for development of a generalized pouring policy, as it’s not guaranteed that the agent will see positive reinforcing rewards in each episode of pouring. The extrinsic reward shaping in this context is described in Algorithm 5.

6.5.5 Intrinsic Curiosity Module

The reward function as discussed in Algorithm 5, is essentially a Sparse Reward Problem. In the absence of recurrent reinforcing rewards, this means that an agent requires a large number of training steps to explore the environment for training a robust policy. One way to essentially improve this situation is to use an intrinsic reward-based system. In addition to the extrinsic rewards, intrinsic rewards can aid an agent to explore it’s environment along with exploitation of the policy. This chapter uses one such intrinsic reward-based system called the “Intrinsic Curiosity Module” based on the work of Pathak et al. [108]. The idea of curiosity-driven learning is to build a reward function intrinsic to an agent, such that it acts as a self-learner, and improves it’s own policy based on the intrinsic reward. This involves using the feature representation of the states of two successive time steps to predict the cur-

rent time-step's actions and using current time step's actions and feature representation of the states to predict next time step's feature representation of the state. The discrepancy in prediction of next feature representation of the states, finally forms the basis of the intrinsic reward signal. A large prediction error is indicative of exploration of uncharted environment. This in turn helps the agent to explore, thus, finding better extrinsic results in the future. In this work, a prominent observation is made in the PourNet's performance improvement when using the ICM. Chapter 6.7, showcases this improvement for LSTM-based PourNet. As compared to the standalone PPO-based model, using ICM sees a steady improvement of the performance in terms of reducing average pouring errors.

Algorithm 4: Action handler algorithm uses output from PourNet to plan 3D motion at Robot's end effector.

Given:

$\mathbf{S} \leftarrow$ Pouring container's Position.

$\mathbf{T} \leftarrow$ Receiving container's Position.

$L_w \leftarrow$ Weight of liquid in the receiving container.

$W_r \leftarrow$ Target to be filled in the receiving container.

$\Delta L_w \leftarrow$ Tolerance in deviation.

$T_{geometry} \leftarrow$ Geometry of receiving container defined by center \mathbf{T} and extents.

(ω, r) Output from PourNet $(\Delta x, \Delta \theta)$ 3D Motion at the robot's end effector **if**

$S_x \notin [T_{geometry_{x_{min}}}, T_{geometry_{x_{max}}}]$ & $S_z \notin [T_{geometry_{z_{min}}}, T_{geometry_{z_{max}}}]$ **then**

if $S_y - T_y \geq 2 \times T_{geometry_{y_{extents}}} + 0.05$ **then**

$\mathbf{d} = \mathbf{T} - \mathbf{S}$

else

$\mathbf{d} = (T_x, S_y, T_z) - (S_x, S_y, S_z)$

 // $\mathbf{d} \leftarrow$ Direction of Motion.

$\Delta x = \mathbf{S} + (r \times \Delta t) \times \mathbf{d}$

 output Δx

else

if $L_w \leq (W_r \pm \Delta L_w)$ **then**

$\Delta \theta = \omega \times \Delta t \quad \forall \omega \geq 0$

else

$\Delta \theta = \omega \times \Delta t \quad \forall \omega < 0$

 // $\Delta \theta \leftarrow$ Rotation about pouring container's forward axis.

 output $\Delta \theta$

6.5.6 Curriculum Learning

Curriculum Learning [109] is inspired from human behavior of breaking down a complex difficult task into smaller structured tasks with increasing difficulty.

As an inherent part of humans learning from their environment, it can be observed that learning is certainly better and more efficient if it's organized into some meaningful order. The order is illustrative of gradually increasing concepts, and complexity. Hence, an intelligent agent using curriculum learning approach, first starts out with only easy examples of a task and then gradually increases the task difficulty.

This is often considered as a robust Environment Parameter Randomization strategy divided into a hierarchical structure of lessons. An agent tries to solve a task using easier lessons at first. As the policy becomes good in determining appropriate actions for the observations, gradually, the difficulty of the next lessons is increased.

PourNet uses the Curriculum Learning as a primary “**Liquid Properties Randomization Strategy**”. Some liquids are easier to handle for pouring. However, as viscosity and adhesion in liquids tend to increase, the pouring task becomes more challenging. As a result, PourNet uses a hierarchically designed curriculum for liquid parameter randomization.

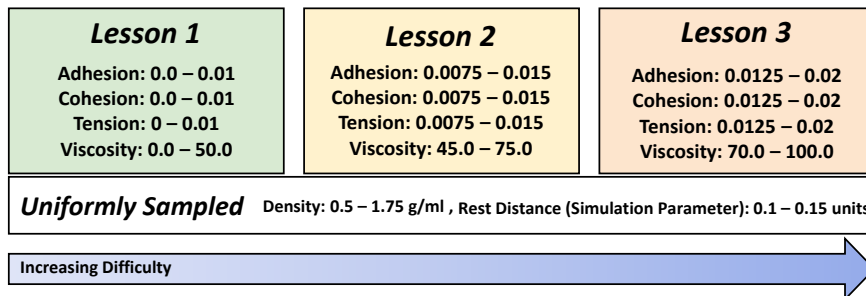


Figure 6.6: Curriculum Learning lessons and parameters. The idea is to domain randomize the parameters in increasing order of their difficulty in pouring dynamics.

6.5.6.1 Designing Pouring Curriculum

This chapter considers the following liquid properties for designing a curriculum-based pouring scenario: Density, Rest Particle Distance, Adhesion, Cohesion, Surface Tension, Viscosity.

As such, out of these enumerated liquid properties, the effect of adhesion, cohesion, surface tension, and viscosity affects the pouring behavior to a greater extent. These are either sources of spillage, or factor in the Navier-Stokes [98] formulation describing fluids in motion. For instance, more viscous and adhesive liquids like honey and ketchup are difficult to pour as compared to the less viscous ones like water. As a result, the curriculum can be designed with these four properties in order of increasing values, representative of the increasing complexity of the task. Density and Rest Particle Distance, on the other hand, can be domain randomized using a simple uniform sampling strategy.

The PourNet’s curriculum as proposed in this chapter in terms of randomized Nvidia Flex liquid parameters has been depicted in the Figure 6.6. The PourNet’s performance improvement can be seen in the Figure 6.8 when using Curriculum Learning along with curiosity. This is compared to the scenario where PPO is used standalone and where PPO is used along with the ICM. This conforms with the expectation as the policy is trained with a three-lesson curriculum of increasing difficulty. Hence, policy performs better from the first lesson itself as it sees enough positive rewards from the environment and adjusts it’s knowledge as the difficulty of the lesson begins to increase.

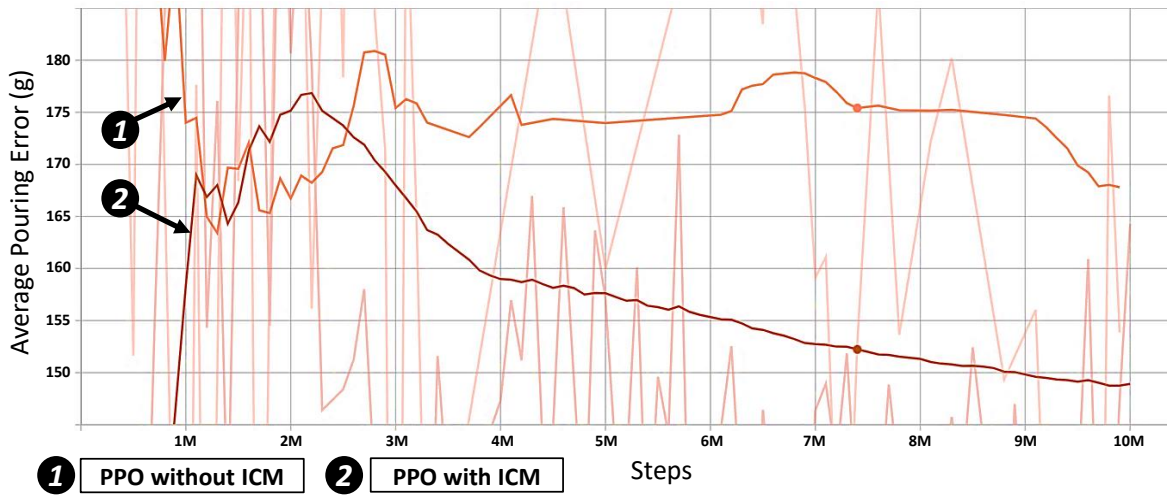


Figure 6.7: Performance Improvement by using ICM.

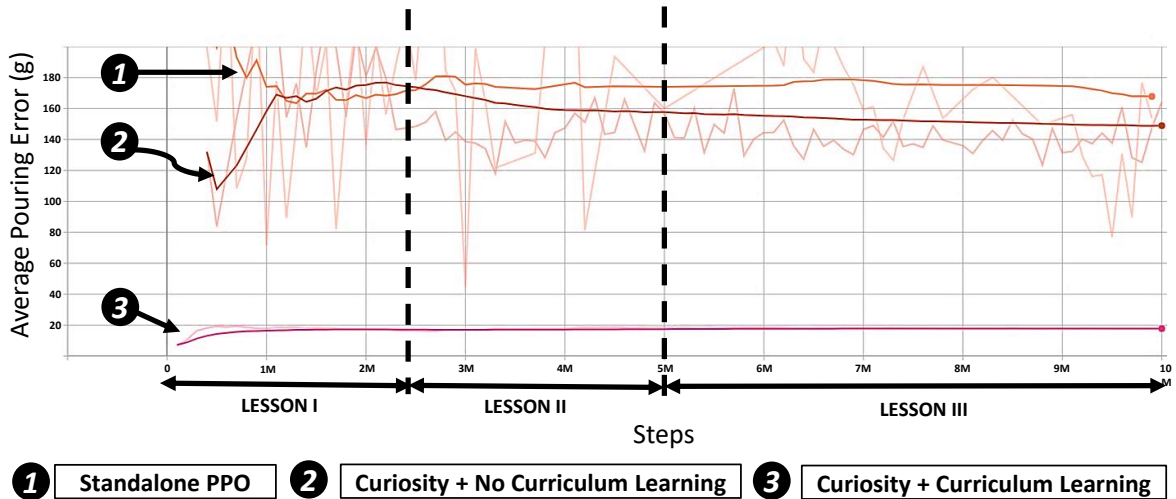


Figure 6.8: Performance Improvement by using Curriculum Learning.

6.5.7 End-to-end Technical Specifications

The details of the overall technical specifications, for PourNet as seen in Chapter 6.2 are:

1. **PourNet:** It is the intelligent agent in the environment which observes the input and takes action by using it's knowledge from the pouring policy.
2. **Observations:** The observation is a 12x1 vector with the following inputs. *Liquid Features:* The liquid features are a 5x1 vector of observation including information of Density, Cohesion, Surface Tension, Viscosity, and Adhesion. *Pouring Container Geometry:* It is a 2x1 vector of the pouring container's diameter and depth. *Force/Torque Feedback:* This includes a 1x1 vector of weight information of the liquid in the target container. *User Input:* This is a 1x1 input of the target level of the liquid to fill. *Measurements:* This is a 3x1 vector which includes measured observation from the environment and

these include the difference in the fill level in the target container, average pouring error for all pouring trials until the current time step and the rotational tilt of the pouring container.

3. **ICM:** This is the Intrinsic Curiosity Module, which form the basis of rewards intrinsic to the PourNet.
4. **Actions:** The PourNet takes actions based on the observation by exploiting it's underlying pouring policy which updates the pouring container's rotational and translational positions in the scene.
5. **Environment:** It is the simulation environment where the PourNet gains the experiences for a robust policy.

Algorithm 5: PourNet: Extrinsic Reward Shaping Algorithm

Given:

Coll: State of collision between pouring and receiving container.

ϵ_t : Current episode's pouring error.

$\Delta\epsilon$: Average Pouring error over past n successful episodes.

$L_w \leftarrow$ Weight of liquid in the receiving container.

$W_r \leftarrow$ Target to be filled in the receiving container.

$\Delta L_w \leftarrow$ Tolerance in deviation.

$(\epsilon_t, \Delta\epsilon) r_t^e$ Extrinsic reward for the current episode. $r_t^e = 0$

while $L_w \leq (W_r \pm \Delta L_w)$ **do**

$r_t^e = r_t^e + \frac{-1}{Max.Number.of.Steps}$

if *Coll is True* **then**

$r_t^e = r_t^e - 1$

return r_t^e

if $\epsilon \leq 10$ grams **then**

$r_t^e = r_t^e + 1$

if $\Delta\epsilon \leq 20$ grams **then**

$r_t^e = r_t^e + 1$

output r_t^e

6.5.8 PourNet's Architecture

Overall, two contesting deep neural network architectures were considered for the PourNet. One based on a recurrent neural network using LSTM [110] as foundational blocks and the other using a feed-forward neural network.

The primary investigation of this chapter is to establish the applicability of the RL to precision pouring problem. Hence, the aforementioned architectures forms the basis of underlying model which uses RL for policy modeling.

6.5.8.1 LSTM-based PourNet

The LSTM-based PourNet architecture is described in the [Figure 6.5](#). The architecture has the following features: Number of Layers: 3, Number of hidden units: 256 units per hidden layer, Memory Size: 256, Sequence Length: 64.

6.5.8.2 Feed-forward neural network-based PourNet

Another architecture for the PourNet is based on the Feed-forward neural network (FNN). The PourNet's FNN-based architecture is similar to the LSTM-based one, where instead of a LSTM block, a feedforward neural network is used. The performance comparison is as described in [Table 6.3](#).

6.6 Experimental Evaluation

6.6.1 Simulation

6.6.1.1 Simulation Setup

The primary precision pouring setup discussed in this chapter is based on a novel simulation environment. The basis of our simulation environment are the Unity3D Game Engine [111] and the NVIDIA® Flex a *Position Based Fluid* [112] engine, and NVIDIA® PhysX 4.0 for rigid body simulations. The Machine Learning (ML) in the Unity3D® is catered to by ML-Agents [111] library. The overall end-to-end pipeline is as described in [Figure 6.2](#). We tested on the simulated robot with a motion planner in the loop by bridging it to ROS via the approach proposed in [80].



Figure 6.9: Liquid Pouring with robot in the loop scene inside RRS-PL.

6.6.1.2 Training

The PourNet-based policy was trained for maximum *10 million time-steps* in a **Federated Learning** approach [113]. There are 32 parallel environments that independently provide a set of observations and actions to collaboratively train a single brain i.e. PourNet-based pouring policy. Essentially, each environment keeps all the local training data private to itself. By using this approach, not only is the training sped up, but the experiences are randomized as well. Randomness is a key to robust policy learning. The curriculum learning was applied with the lesson progression as 25% of time-steps for the first lesson, further 25% for the second lesson, and finally, remaining 50% of the time-steps for the third and the final lesson. Training and simulations was done using two NVIDIA® Quadro RTX6000 GPUs.

6.6.1.3 Discussion

The Table 6.3 enumerates the results of the average pouring error for the simulated liquid profiles as shown in Figure 6.10 with the NVIDIA Flex parameters as enlisted in Table 6.2. Each simulated liquid profile was poured 10 times using a trained PourNet in the pure simulation environment using a randomized pouring container as shown in Figure 6.10. It can be observed that when comparing the LSTM-based PourNet with FNN-based PourNet, LSTM-based PourNet performs slightly better than the FNN-based PourNet for all simulated liquid profiles. Because of this better performance, all further experiments and results in the subsequent sections are using LSTM-based PourNet. There is a 26.7% improvement for water, 16.6% for ink, 11.94% for oil, 15.6% for the glycerine, 4.5% for honey, and finally 8.04% for the ketchup in terms of average pouring deviation. It should be noted that, since the simulated ketchup profile is very adhesive and also viscous, in absence of shaking action from the PourNet, the higher targets could not be filled.

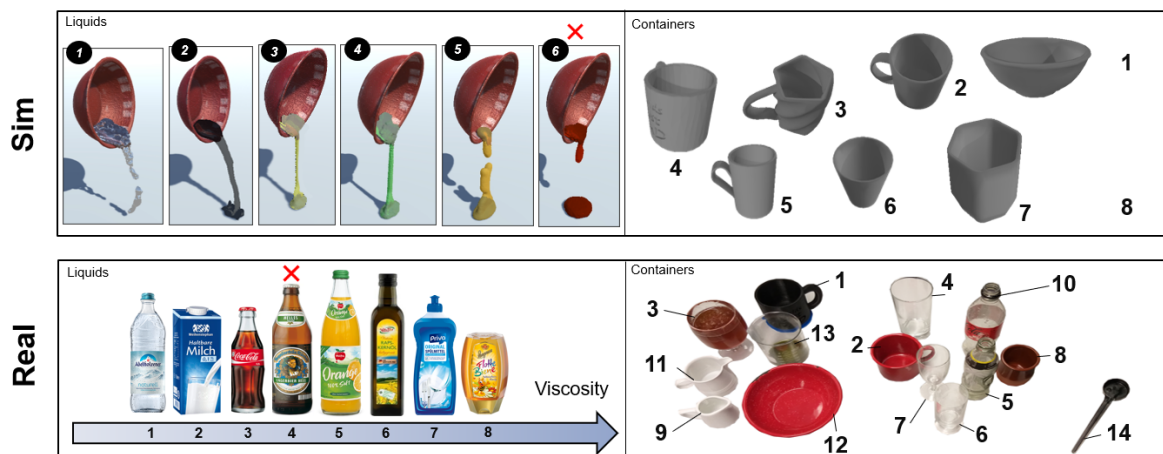


Figure 6.10: The simulated (top) vs real (bottom) liquids and containers. On the right the test environment with robot in the loop is illustrated.

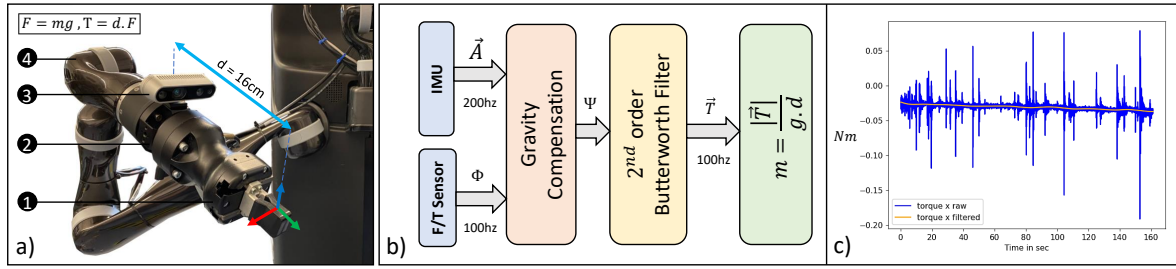


Figure 6.11: a) Experimental Setup with Kinova Movo. b) Force measurement steps using F/T sensor. c) Second-order ButterWorth Low-pass Filter, operating at 100 Hz, with a cut-off frequency of 1 Hz.

Table 6.1: Average Pouring deviation for water compared to the state-of-the-art approaches

Liquid	Closed-loop Control [67]	MPNet [71]	MultiFrame CNN (thermal vision) [74]	DDPG [114]	PourNet
Water	0.95 ± 1.16 ml to 11.88 ± 4.63 ml	6.3 ± 6.1 ml to 23.7 ± 18.7 ml	38 ml	19.96 ml	3.8 ± 0.69 ml to 8.66 ± 1.52 ml

6.6.2 Real Experiments

6.6.2.1 Platform Setup

The experimental setup to transfer PourNet to a real robot is based on the Kinova[®] Movo. As such, the experimental setup includes the following sub-systems: Robotiq HandE Gripper, BotaSys[®] SENSONE force/torque sensor operating at 100 Hz, and Intel[®] RealSense D435i only as Inertial Measurement Unit (IMU) at 200 Hz. Figure 6.11 highlights the overall experimental setup with the appropriate modules labeled. In all of our experiments, the stationary left arm is holding a tray with a radius of 10cm, and the pouring is carried out by the right arm.

Table 6.2: Simulation Liquid Profiles (Flex Parameters)

Liquid	Adhesion	Cohesion	Surface Tension	Viscosity
Water	0.0001	0.001	0.005	0.01
Ink	0.0001	0.0025	0.0075	0.05
Oil	0.001	0.001	0.0001	6.5
Glycerine	0.005	0.001	0.0001	50
Honey	0.025	0.001	0.0001	65
Ketchup	0.1	0.001	0.00001	80

6.6.2.2 Weight Measurement

For accurate object in hand weight measurements, we should eliminate the effect of the gripper's weight. After applying the gravity compensation using the IMU sensor [115] by factoring in the sensor bias, an accurate reading can be provided. Additionally, as a result, the

Table 6.3: LSTM-based PourNet: Experiment with liquid simulation profiles

Type	Target(g)	Water	Ink	Oil	Glycerine	Honey	Ketchup
LSTM	50	7.72	4.44	4.66	4.4	9.1	11.66
LSTM	100	2.3	6.34	8.64	7.66	5.26	20.36
LSTM	150	4.96	8.38	6.44	8.8	12	-
LSTM	200	4.76	5.9	7.38	6.9	10.54	-
FNN	50	8.49	7.65	5.35	8.3	9.4	15.42
FNN	100	5.13	7.94	8.91	8.52	8.26	19.40
FNN	150	7.65	8.32	8.3	8.6	10.2	-
FNN	200	5.65	6.17	8.25	7.5	10.79	-

sensor possesses a noise-free resolution of as low as 0.005 Nm torque for stationary left hand and 0.01 Nm torque for moving right hand at the 100 Hz operating frequency. In our experiments we are using the torque measurements on the left arm to measure the weight of the target container. However, in order to ensure a smooth signal from the sensor, filtering must be performed in order to safeguard PourNet’s observations from short-duration spikes. This might result in unpredictable action response, which might be hazardous. As a result, for filtering the raw force/torque reading from the sensors, a second-order *Butterworth Low-Pass Filter* [116] was designed. On the left arm, we achieved an accuracy of 2 grams for the target weight measurements.

6.6.2.3 Experimental Containers and Liquids

Figure 6.10 describes the containers and liquids used for pouring. As such, ten novel pouring containers were used for the experimental evaluation along with one 3D printed training container. These cover a wide range of container types demonstrating variation in geometrical features. Likewise, seven liquids were considered for the pouring experiments. These liquids cover a wide range of experimental liquid features demonstrating generalized behavior of the PourNet.

6.6.2.4 Pouring Experiments

Figure 6.12 describes the action sequence that the robot follows for precision pouring task. The pouring experiments were divided into two categories i.e. *I*: demonstrating the generalization to different liquid types, and *II*: demonstrating generalization to different container geometries. In the experiment *I*, the pouring container was kept the same. The results for the experiment *I* has been enumerated in the Table 6.4. For each liquid, the container 2 as seen in the Figure 6.10, was used to pour three target profiles i.e. 50g, 100g, and 150g. Against each target, a liquid was poured thrice. Likewise, for the experiment *II*, the liquid type was fixed by choosing water for each pouring container. For each container, 50g of water was poured thrice. The Figure 6.13 depicts the pouring experiment for the experiment *II*. Beer cannot be

simulated due to its effervescence and subsequent large foam formation, therefore in the real experiments it failed (see Figure 6.12).

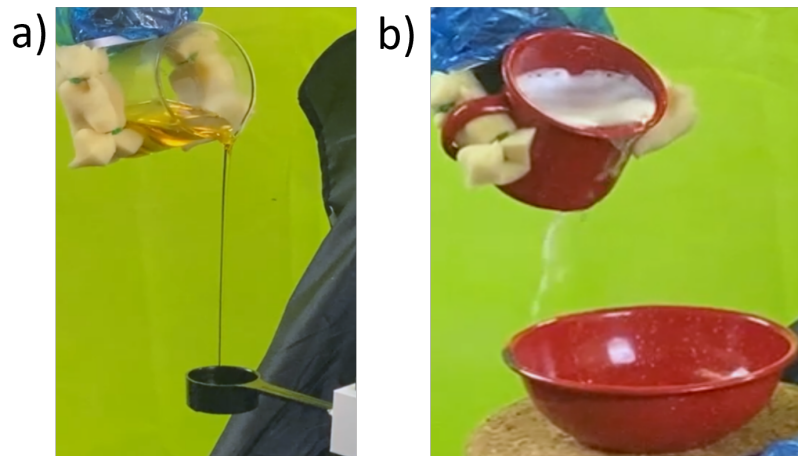


Figure 6.12: a) Denotes the novel oil to spoon pouring experiment, container number 13 to number 14 b) Denotes the beer pouring failure.

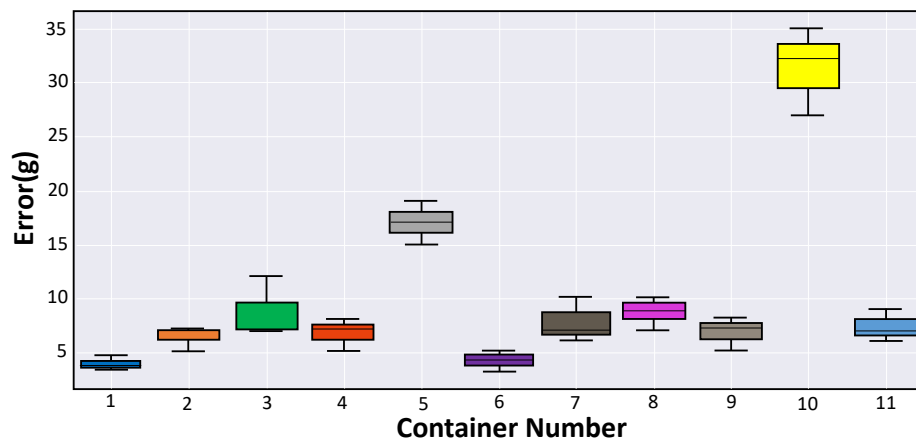


Figure 6.13: Experiments using water and different containers.

Table 6.4: Deviation from target using different liquids.

Target (g)	Water	Milk	Cola	Orange Juice	Rapeseed Oil	Dish Soap	Honey
50	6.33	3.33	6.00	2.67	8.55	3.33	9.33
100	3.55	3.66	7.55	2.33	5.67	1.67	8.66
150	7.33	5.53	6.00	4.67	8.55	2.12	5.23

6.6.2.5 Discussion

From Figure 6.13, it can be observed that the errors are within 10 grams for most of the pouring containers. It can be noted however, that container number 5 and container number 10 are outliers. This can be attributed to the fact that these are bottles, and, were not a part of the training setup. As a result, PourNet had no experience in dealing with such objects while training. Hence, ignoring the outliers, on comparing the performance for pouring water with some of the state-of-the-art methodologies, the results are as enlisted in Table 6.1. It can be observed that the PourNet outperforms these with small errors.

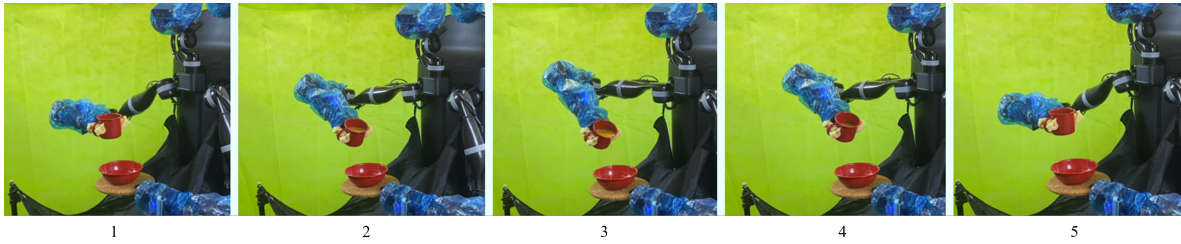


Figure 6.14: Demonstration of pouring orange juice in the bowl.



Figure 6.15: Demonstration of pouring oil in the spoon.

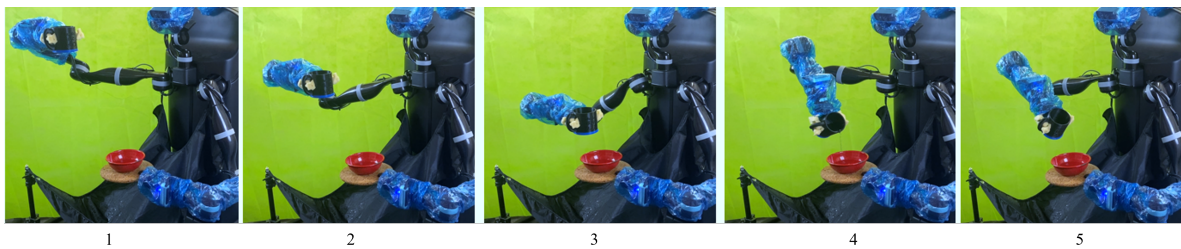


Figure 6.16: Demonstration of generalization of the starting point.

6.7 Chapter Summary

Pouring liquids accurately into containers is one of the most challenging tasks for robots as they are unaware of the complex fluid dynamics and the behavior of liquids when pouring. Therefore, it is not possible to formulate a generic pouring policy for real-time applications. In this chapter, we propose PourNet, as a generalized solution to pouring different liquids into containers. PourNet is a hybrid planner that uses deep reinforcement learning for end-effector planning, and Nonlinear Model Predictive Control, for joint planning. In this work,

we introduce a novel simulation environment using Unity3D and NVIDIA-Flex to train our agents. By effective choice of the state space, action space and the reward functions, we allow for a direct sim-to-real transfer of the learned skills without additional training. In the simulation, PourNet outperforms the state-of-the-art by an average of 4.9g deviation for water-like, and 9.2g deviation for honey-like liquids. In the real-world scenario using Kinova Movo Platform, PourNet achieves an average pouring deviation of 2.3g for dish soap when using a novel pouring container. The average pouring deviation measured for water was 5.5g.

Chapter 7

Conclusion and Future work

The integration of teleoperation with medical robotics in elderly care is an active research topic. The associated goal is that elderly people will be able to receive care with fewer caregivers. After a brief introduction of background and related works as well as the experimental setup this dissertation started with designing a real-time motion planner for redundant manipulators. Our next step was to extend the work to non-stable network connections, and to use shared autonomy to handle network disconnections. We understood that there is always a gap when we are switching to the predefined skill. We tackled this displacement by applying non-rigid registration. At the end, we designed a hybrid planner using reinforcement learning to train the robot to learn how to pour liquids. On the Kinova Movo platform, we demonstrated that the pouring agent is capable of performing the task with various liquid profiles and geometries in real-time.

7.1 Summary

We summarized the three main topics of the dissertation here:

This first topic of the dissertation was the NMPC-MP, a motion planner that can be used for robot manipulator teleoperation. The planner has three features: (i) it works in real-time; (ii) it can avoid dynamic and arbitrarily shaped obstacles of primitive, mesh, or octomap type; (iii) it has a preview to the future, thus real-time motions are smoothed. Comparisons in simulation show that NMPC-MP is the fastest among the tested planners and can be used in real-time. Furthermore, it generates trajectories of high quality. We implemented our system for bimanual teleoperation on the real Movo robot. Self-collision avoidance for robot arms and body is integrated into the system, and external dynamic obstacles can be tackled as well.

For the second topic, we presented Skill-CPD, a high-level skill adaptation strategy for shared autonomy (SA) in manipulator teleoperation. Our approach refines the reproduced LQT trajectory after the HSMM-based intention recognition. We formulate the skill refinement as a point set registration problem using the CPD algorithm. Finally, we evaluate Skill-CPD in a 2D English letter drawing and a 3D robot-assisted feeding scenario. The experimental results illustrate that Skill-CPD performs the manipulation task with minimum displacement, thus enhances the overall trajectory consistency.

Furthermore, we for the third topic we explored a RL method for precision pouring problem. Precision pouring was primarily driven by curiosity and curriculum-based reinforcement learning. Based on the trained PourNet, various experiments were carried out both in simulation and by transferring the trained policy on a real robot. Based on the experiments, we observed generalization of the PourNet to different liquid types. The performance in terms of pouring deviations ranged from 2.33 g for dish soap to 7.55 g for honey. Likewise, the dissertation demonstrated PourNet’s ability to generalize to novel pouring containers. The performance varied from 3.8 ± 0.69 ml for the best performing container to the 8.6 ± 1.52 ml for the worst performing container. This thus demonstrates PourNet as a precise pouring agent with an ability to generalize for different liquid types and different pouring container geometries.

7.2 Limitations

One disadvantage of NMPC-MP is that it is prone to a local minimum when complicated obstacles are integrated. For Skill-CPD the super long trajectories with many points will slow down the real-time behavior of the task performance and will lower the intention recognition accuracy. PourNet could not pour beer as it’s foam resulted in excessive spillage about the pouring container’s outer surface. Hence, it might be fruitful to investigate all liquids that exhibit an effervescence behavior.

7.3 Future Work

One improvement for the motion planner could be adaptive modifications on parameters ph and ch for different scenarios. Another direction is to study novel QP solvers and to have faster optimizations. Rigorous proof of the stability for the NMPC-MP is left for future work as well. For shared autonomy, further improvement of intention recognition would be beneficial. For example, the recurrent neural network (RNN) or deep neural network (DNN) have been widely employed to perform sequential recognition. Also since CPD in general is resource hungry and slow, one promising direction for future work is to utilize neural network-based methods instead of CPD. For the liquid pouring agent we can explore other intrinsic reward signals using techniques such as Generative Adversarial Imitation Learning (GAIL) [117]. As a result, the training speed will improve and the exploration will be easier and more accurate for the agent.

Bibliography

Publications by the author

- [1] E. Babaians, S. Hu, M. Karim, and E. Steinbach, "Nmpc-mp: Real-time nonlinear model predictive control for safe motion planning in manipulator teleoperation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 8309–8316. DOI: [10.1109/IROS51168.2021.9636802](https://doi.org/10.1109/IROS51168.2021.9636802).
- [6] S. Ayvaşık, E. Babaians, A. Papa, *et al.*, "Remote robot control with haptic feedback over the munich 5g research hub testbed," in *2023 IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, IEEE, 2023, pp. 349–351.
- [7] R. M. Calvo, T. de Cola, J. Poliak, *et al.*, "Optical feeder links for future very high-throughput satellite systems in b5g networks," in *2020 European Conference on Optical Communications (ECOC)*, 2020, pp. 1–4. DOI: [10.1109/ECOC48923.2020.9333405](https://doi.org/10.1109/ECOC48923.2020.9333405).
- [8] E. Babaians, T. Sharma, M. Karimi, S. Sharifzadeh, and E. Steinbach, "Pournet: Robust robotic pouring through curriculum and curiosity-based reinforcement learning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 9332–9339.
- [9] E. Babaians, D. Yang, M. Karimi, X. Xu, S. Ayvasik, and E. Steinbach, "Skill-cpd: Real-time skill refinement for shared autonomy in manipulator teleoperation," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 6189–6196.
- [10] M. Karimi, M. Oelsch, O. Stengel, E. Babaians, and E. Steinbach, "Lola-slam: Low-latency lidar slam using continuous scan slicing," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2248–2255, 2021.
- [11] M. Karimi, E. Babaians, M. Oelsch, and E. Steinbach, "Deep fusion of a skewed redundant magnetic and inertial sensor for heading state estimation in a saturated indoor environment," *International Journal of Semantic Computing*, vol. 15, no. 03, pp. 313–335, 2021.

- [12] M. Karimi, E. Babaian, M. Oelsch, T. Aykut, and E. Steinbach, "Skewed-redundant hall-effect magnetic sensor fusion for perturbation-free indoor heading estimation," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, IEEE, 2020, pp. 367–374.

General publications

- [2] dreamstime. "Hand-elderly-lady." (2022), [Online]. Available: <https://www.dreamstime.com/royalty-free-stock-photos-hand-elderly-lady-holding-apple-image10920328> (visited on 05/15/2022).
- [3] freepik. "Pouring-antipathetic-syrup." (2022), [Online]. Available: https://www.freepik.com/premium-photo/woman-hand-pouring-antipathetic-syrup-from-bottle-spoon-healthcare-medical-concept_4848961.htm (visited on 05/15/2022).
- [4] N. Tian, A. K. Tanwani, K. Goldberg, and S. Sojoudi, "Mitigating network latency in cloud-based teleoperation using motion segmentation and synthesis," in *International Symposium on Robotics Research*, 2019.
- [5] A. Myronenko and X. Song, "Point set registration: Coherent point drift," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2262–2275, 2010. DOI: [10.1109/TPAMI.2010.46](https://doi.org/10.1109/TPAMI.2010.46).
- [13] Y. Nakamura and H. Hanafusa, "Optimal redundancy control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 1, pp. 32–42, 1987. DOI: [10.1177/027836498700600103](https://doi.org/10.1177/027836498700600103). eprint: <https://doi.org/10.1177/027836498700600103>. [Online]. Available: <https://doi.org/10.1177/027836498700600103>.
- [14] O. KHATIB, "Dynamic control of manipulators operating in a complex environment," *Proceedings of the 3rd CISM-IFTOMM Symposium on Theory and Practice of Robots and Manipulators*, pp. 267–282, 1978. [Online]. Available: <https://ci.nii.ac.jp/naid/10007973016/en/>.
- [15] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, 1987. DOI: [10.1177/027836498700600201](https://doi.org/10.1177/027836498700600201). eprint: <https://doi.org/10.1177/027836498700600201>. [Online]. Available: <https://doi.org/10.1177/027836498700600201>.
- [16] S. LaValle, "Rapidly-exploring random trees : A new tool for path planning," *The annual research report*, 1998.
- [17] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. DOI: [10.1109/70.508439](https://doi.org/10.1109/70.508439).
- [18] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 489–494. DOI: [10.1109/ROBOT.2009.5152817](https://doi.org/10.1109/ROBOT.2009.5152817).

-
- [19] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, May 2011, pp. 4569–4574. DOI: [10.1109/ICRA.2011.5980280](https://doi.org/10.1109/ICRA.2011.5980280).
- [20] Wikipedia. "Prediction horizon in mpc." (2022), [Online]. Available: https://en.wikipedia.org/wiki/File:MPC_scheme_basic.svg (visited on 05/15/2022).
- [21] P. A. Gagniuc, *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.
- [22] A. K. Tanwani, "Generative models for learning robot manipulation skills from humans," 2018. DOI: [10.5075/epfl-thesis-8320](https://doi.org/10.5075/epfl-thesis-8320). [Online]. Available: <http://infoscience.epfl.ch/record/234536>.
- [23] S.-Z. Yu, "Hidden semi-markov models," *Artif. Intell.*, vol. 174, pp. 215–243, Feb. 2010. DOI: [10.1016/j.artint.2009.11.011](https://doi.org/10.1016/j.artint.2009.11.011).
- [24] A. K. Tanwani and S. Calinon, "A generative model for intention recognition and manipulation assistance in teleoperation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 43–50. DOI: [10.1109/IROS.2017.8202136](https://doi.org/10.1109/IROS.2017.8202136).
- [25] D. Reynolds, "Gaussian mixture models," *Encyclopedia of Biometrics*, Jan. 2008. DOI: [10.1007/978-0-387-73003-5_196](https://doi.org/10.1007/978-0-387-73003-5_196).
- [26] A. K. Tanwani and S. Calinon, "Learning robot manipulation tasks with task-parameterized semitied hidden semi-markov model," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 235–242, 2016. DOI: [10.1109/LRA.2016.2517825](https://doi.org/10.1109/LRA.2016.2517825).
- [27] A. K. Tanwani, J. Lee, B. Thananjeyan, et al., *Generalizing robot imitation learning with invariant hidden semi-markov models*, 2018. arXiv: [1811.07489](https://arxiv.org/abs/1811.07489) [cs.RO].
- [28] A. K. Tanwani and S. Calinon, "Learning robot manipulation tasks with task-parameterized semitied hidden semi-markov model," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 235–242, 2016. DOI: [10.1109/LRA.2016.2517825](https://doi.org/10.1109/LRA.2016.2517825).
- [29] *Point set registration*, https://en.wikipedia.org/wiki/Point_set_registration.
- [30] M Waltz and K Fu, "A heuristic approach to reinforcement learning control systems," *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 390–398, 1965.
- [31] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [32] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [33] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.
- [34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

- [35] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.
- [36] M. Liu and G. Liu, "Smoothed particle hydrodynamics (sph): An overview and recent developments," *Archives of computational methods in engineering*, vol. 17, no. 1, pp. 25–76, 2010.
- [37] M. M. Miles Macklin, "Position based fluids," *ACM TOG* 32(4), 2013.
- [38] K. Bodin, C. Lacoursiere, and M. Servin, "Constraint fluids," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 3, pp. 516–526, 2011.
- [39] M. Müller, D. Charypar, and M. H. Gross, "Particle-based fluid simulation for interactive applications.," in *Symposium on Computer animation*, vol. 2, 2003.
- [40] J. J. Monaghan, "Smoothed particle hydrodynamics," *Annual review of astronomy and astrophysics*, vol. 30, pp. 543–574, 1992.
- [41] B. Solenthaler and R. Pajarola, "Predictive-corrective incompressible sph," in *ACM SIGGRAPH 2009 papers*, 2009, pp. 1–6.
- [42] S. An and D. Lee, "Prioritized inverse kinematics with multiple task definitions," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1423–1430. DOI: [10.1109/ICRA.2015.7139376](https://doi.org/10.1109/ICRA.2015.7139376).
- [43] F. Flacco, A. De Luca, and O. Khatib, "Control of redundant robots under hard joint constraints: Saturation in the null space," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 637–654, 2015. DOI: [10.1109/TRO.2015.2418582](https://doi.org/10.1109/TRO.2015.2418582).
- [44] D. Rakita, B. Mutlu, and M. Gleicher, "Relaxedik: Real-time synthesis of accurate and feasible robot arm motion," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, 2018. DOI: [10.15607/RSS.2018.XIV.043](https://doi.org/10.15607/RSS.2018.XIV.043).
- [45] S. Karaman and E. Frazzoli, *Sampling-based algorithms for optimal motion planning*, 2011. arXiv: [1105.1186](https://arxiv.org/abs/1105.1186) [cs.RO].
- [46] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, 2013. DOI: [10.15607/RSS.2013.IX.031](https://doi.org/10.15607/RSS.2013.IX.031).
- [47] H. Girgin and S. Calinon, *Nullspace structure in model predictive control*, 2019. arXiv: [1905.09679](https://arxiv.org/abs/1905.09679) [cs.RO].
- [48] M. Rubagotti, T. Taunyazov, B. Omarali, and A. Shintemirov, "Semi-autonomous robot teleoperation with obstacle avoidance via model predictive control," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2746–2753, 2019.
- [49] N. Imanberdiyev and E. Kayacan, "Redundancy resolution based trajectory generation for dual-arm aerial manipulators via online model predictive control," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2020, pp. 674–681.

-
- [50] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe, "Safe and fast tracking on a robot manipulator: Robust mpc and neural network control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3050–3057, 2020.
- [51] A. K. Tanwani and S. Calinon, "A generative model for intention recognition and manipulation assistance in teleoperation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 43–50.
- [52] S.-Z. Yu, "Hidden semi-markov models," *Artificial intelligence*, vol. 174, no. 2, pp. 215–243, 2010.
- [53] Wentao Yu, R. Alqasemi, R. Dubey, and N. Pernalet, "Telemanipulation assistance based on motion intention recognition," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 1121–1126. DOI: [10.1109 / ROBOT. 2005.1570266](https://doi.org/10.1109/ROBOT.2005.1570266).
- [54] R. M. Aronson, N. Almutlak, and H. Admoni, "Inferring goals with gaze during teleoperated manipulation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.
- [55] A. K. Tanwani, P. Sermanet, A. Yan, R. Anand, M. Phielipp, and K. Goldberg, "Motion2vec: Semi-supervised representation learning from surgical videos," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2174–2181.
- [56] A. Dragan and S. Srinivasa, "A policy blending formalism for shared control," *International Journal of Robotics Research*, 2013.
- [57] D. Aarno and D. Kragic, "Motion intention recognition in robot assisted applications," *Robot. Auton. Syst.*, vol. 56, no. 8, pp. 692–705, Aug. 2008, ISSN: 0921-8890. DOI: [10.1016/j.robot.2007.11.005](https://doi.org/10.1016/j.robot.2007.11.005). [Online]. Available: <https://doi.org/10.1016/j.robot.2007.11.005>.
- [58] M. Li and A. M. Okamura, "Recognition of operator motions for real-time assistance using virtual fixtures," in *Proceedings of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'03)*, ser. HAPTICS '03, USA: IEEE Computer Society, 2003, p. 125, ISBN: 0769518907.
- [59] J. R. Medina, M. Lawitzky, A. Mörtl, D. Lee, and S. Hirche, "An experience-driven robotic assistant acquiring human knowledge to improve haptic cooperation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2416–2422. DOI: [10.1109/IROS.2011.6095026](https://doi.org/10.1109/IROS.2011.6095026).
- [60] Z. Wang, K. Mülling, M. P. Deisenroth, *et al.*, "Probabilistic movement modeling for intention inference in human-robot interaction," *Int. J. Rob. Res.*, vol. 32, no. 7, pp. 841–858, Jun. 2013, ISSN: 0278-3649. DOI: [10.1177 / 0278364913478447](https://doi.org/10.1177/0278364913478447). [Online]. Available: <https://doi.org/10.1177/0278364913478447>.

- [61] F. Meier, E. Theodorou, and S. Schaal, "Movement segmentation and recognition for imitation learning," in *Seventeenth International Conference on Artificial Intelligence and Statistics, La Palma, Canary Islands*, Apr. 2012. [Online]. Available: <http://www-clmc.usc.edu/publications/M/meier-AISTATS2012>.
- [62] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, "Learning and reproduction of gestures by imitation," *IEEE Robotics Automation Magazine*, vol. 17, no. 2, pp. 44–54, 2010. DOI: [10.1109/MRA.2010.936947](https://doi.org/10.1109/MRA.2010.936947).
- [63] M. Selvaggio, M. Cognetti, S. Nikolaidis, S. Ivaldi, and B. Siciliano, "Autonomy in physical human-robot interaction: A brief survey," *IEEE Robotics and Automation Letters*, 2021.
- [64] A. D. Dragan and S. S. Srinivasa, "A policy-blending formalism for shared control," *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 790–805, 2013.
- [65] A. S. Bauer, P. Birkenkamp, A. Albu-Schäffer, and D. Leidner, "Bridging the gap between supervised autonomy and teleoperation," in *Proc. FAIM/ISCA Workshop on Artificial Intelligence for Multimodal Human Robot Interaction*, 2018, pp. 44–47.
- [66] T. Weber Martins, A. Pereira, T. Hulin, *et al.*, "Space factory 4.0-new processes for the robotic assembly of modular satellites on an in-orbit platform based on „industrie 4.0“ approach," in *Proceedings of the International Astronautical Congress, IAC*, 2018.
- [67] C. Dong, M. Takizawa, S. Kudoh, and T. Suehiro, "Precision pouring into unknown containers by service robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 5875–5882.
- [68] Y.-Y. Tsai, B. Xiao, E. Johns, and G.-Z. Yang, "Constrained-space optimization and reinforcement learning for complex tasks," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 683–690, 2020.
- [69] M. Akbulut, E. Oztop, M. Y. Seker, X. Hh, A. Tekden, and E. Ugur, "Acnmp: Skill transfer and task extrapolation through learning from demonstration and reinforcement learning via representation sharing," in *Conference on Robot Learning*, PMLR, 2021, pp. 1896–1907.
- [70] L. Rozo, P. Jiménez, and C. Torras, "Force-based robot learning of pouring skills using parametric hidden markov models," in *9th International Workshop on Robot Motion and Control*, IEEE, 2013, pp. 227–232.
- [71] H. Liang, C. Zhou, S. Li, *et al.*, "Robust robotic pouring using audition and haptics," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 10 880–10 887.
- [72] M. Kennedy, K. Schmeckpeper, D. Thakur, C. Jiang, V. Kumar, and K. Daniilidis, "Autonomous precision pouring from unknown containers," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2317–2324, 2019.
- [73] M. Kennedy, K. Queen, D. Thakur, K. Daniilidis, and V. Kumar, "Precise dispensing of liquids using visual feedback," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2017, pp. 1260–1266.

-
- [74] C. Schenck and D. Fox, "Visual closed-loop control for pouring liquids," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 2629–2636.
- [75] R. Mottaghi, C. Schenck, D. Fox, and A. Farhadi, "See the glass half full: Reasoning about liquid containers, their volume and content," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1871–1880.
- [76] C. Do, C. Gordillo, and W. Burgard, "Learning to pour using deep deterministic policy gradients," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3074–3079. DOI: [10.1109/IROS.2018.8593654](https://doi.org/10.1109/IROS.2018.8593654).
- [77] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: <http://arxiv.org/abs/1509.02971>.
- [78] *Fifty2 preonlab - avl.com*, <https://www.avl.com/fifty2-preonlab>, (Accessed on 03/01/2022).
- [79] D. Kubus, T. Kroger, and F. M. Wahl, "On-line rigid object recognition and pose estimation based on inertial parameters," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 1402–1408. DOI: [10.1109/IROS.2007.4399184](https://doi.org/10.1109/IROS.2007.4399184).
- [80] E. Babaian, M. Tamiz, Y. Sarfi, A. Mogoei, and E. Mehrabi, "Ros2unity3d; high-performance plugin to interface ros with unity3d engine," in *2018 9th Conference on Artificial Intelligence and Robotics and 2nd Asia-Pacific International Symposium*, IEEE, 2018, pp. 59–64.
- [81] E. Babaian. "Rrs technical details." (2022), [Online]. Available: <https://github.com/cxdcxd/RRS/blob/master/RRS1.0.pdf> (visited on 05/15/2022).
- [82] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *2015 international conference on advanced robotics (ICAR)*, IEEE, 2015, pp. 510–517.
- [83] J. Rawlings, D. Mayne, and M. Diehl, "Model predictive control: Theory, computation, and design," in Nob Hill Publishing, 2017, pp. 11,18–20, ISBN: 9780975937730. [Online]. Available: <https://books.google.de/books?id=MrJctAEACAAJ>.
- [84] F. Borrelli, A. Bemporad, and M. Morari, "Predictive control for linear and hybrid systems," in Cambridge University Press, 2016, pp. 164–165. [Online]. Available: <http://www.mpc.berkeley.edu/mpc-course-material>.
- [85] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*, <http://repository.fue.edu.xmlui/handle/123456789/3946>, Accessed on 05.09.2020, 2020.
- [86] K. Robotics, *KINOVA Gen2 Ultra lightweight robot User Guide*, <https://www.kinovarobotics.com/en/resources/gen2-technical-resources>, Accessed on 05.10.2020, 2020.
- [87] J. S. Yuan, "Closed-loop manipulator control using quaternion feedback," *IEEE Journal on Robotics and Automation*, vol. 4, no. 4, pp. 434–440, 1988. DOI: [10.1109/56.809](https://doi.org/10.1109/56.809).

- [88] S. W. Shepperd, "Quaternion from rotation matrix," *Journal of Guidance and Control*, vol. 1, no. 3, pp. 223–224, 1978.
- [89] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988. DOI: [10.1109/56.2083](https://doi.org/10.1109/56.2083).
- [90] G. van den Bergen, "Proximity queries and penetration depth computation on 3d game objects," in *Game Developers Conference*, 2001.
- [91] D. Kraft, *A software package for sequential quadratic programming*, ser. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988. [Online]. Available: <https://books.google.de/books?id=4rKaGwAACAAJ>.
- [92] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. DOI: [10.1007/s12532-020-00179-2](https://doi.org/10.1007/s12532-020-00179-2). [Online]. Available: <https://doi.org/10.1007/s12532-020-00179-2>.
- [93] J. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "Qpoases: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, Dec. 2014. DOI: [10.1007/s12532-014-0071-1](https://doi.org/10.1007/s12532-014-0071-1).
- [94] S. G. Johnson, *The NLOpt nonlinear-optimization package*, <http://github.com/stevengj/nlopt>, Accessed on 05.08.2020.
- [95] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 3859–3866.
- [96] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *International journal of computer vision*, vol. 13, no. 2, pp. 119–152, 1994.
- [97] L. Greengard and J. Strain, "The fast gauss transform," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 79–94, 1991.
- [98] P. Constantin and C. Foias, *Navier-stokes equations*. University of Chicago Press, 1988.
- [99] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [100] O. A. Ladyzhenskaya, *The mathematical theory of viscous incompressible flow*. Gordon and Breach New York, 1969, vol. 2.
- [101] D. Surblys, F. Leroy, Y. Yamaguchi, and F. Müller-Plathe, "Molecular dynamics analysis of the influence of coulomb and van der waals interactions on the work of adhesion at the solid-liquid interface," *The Journal of chemical physics*, vol. 148, no. 13, p. 134707, 2018.
- [102] H. C. Verma, *Concepts of physics*. Bharati Bhawan, 1993, ch. 14, pp. 277–300.
- [103] amsoil. "Motor oil." (2022), [Online]. Available: <https://blog.amsoil.com/a-beginners-guide-to-motor-oil/> (visited on 05/15/2022).

-
- [104] disprolim. "Toothpast." (2022), [Online]. Available: https://www.disprolim.com.br/index.php?route=product/product&product_id=346 (visited on 05/15/2022).
- [105] usgs. "Tension." (2022), [Online]. Available: <https://www.usgs.gov/special-topics/water-science-school/science/surface-tension-and-water> (visited on 05/15/2022).
- [106] nigerianscholars. "Cohesion." (2022), [Online]. Available: <https://nigerianscholars.com/tutorials/chemical-foundation-of-life/waters-cohesive-adhesive-properties/> (visited on 05/15/2022).
- [107] husky. "Containers." (2022), [Online]. Available: <https://www.husky.co/en/industries/consumer-goods/beverage/> (visited on 05/15/2022).
- [108] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International conference on machine learning*, PMLR, 2017, pp. 2778–2787.
- [109] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.
- [110] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [111] A. Juliani, V.-P. Berges, E. Teng, *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.
- [112] M. Macklin and M. Müller, "Position based fluids," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pp. 1–12, 2013.
- [113] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.
- [114] Y. Huang, J. Wilches, and Y. Sun, "Robot gaining accurate pouring skills through self-supervised learning and generalization," *Robotics and Autonomous Systems*, vol. 136, p. 103692, 2021.
- [115] D. Kubus, T. Kroger, and F. M. Wahl, "On-line rigid object recognition and pose estimation based on inertial parameters," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2007, pp. 1402–1408.
- [116] S. Butterworth *et al.*, "On the theory of filter amplifiers," *Wireless Engineer*, vol. 7, no. 6, pp. 536–541, 1930.
- [117] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, pp. 4565–4573, 2016.

List of Figures

1.1	Dual hand liquid pouring. One of the basic requirements of a dual arm motion planning system is self-awareness. In a sense, each manipulator is a dynamic obstacle to the other. [1]	1
1.2	Image (a) illustrates an example application for rigid object manipulation [2]. An example of liquid handling and feeding is shown in image (b). Image (c) illustrates precise liquid pouring. Here is an example of medicine in the form of syrup [3].	2
1.3	Assistive teleportation in dynamic environments.	3
1.4	Dissertation big picture.	5
1.5	Overview of research questions and subsequent improvements.	5
2.1	This figures illustrates the kinematic redundancy of a simple interconnected kinematic chain: thanks to one additional joint, the kinematics solver can find a large set of solutions.	7
2.2	Simulated planar robot manipulator	8
2.3	Critical points on the robot	12
2.4	Concept of MPC.	17
2.5	Prediction horizon in MPC [20].	17
2.6	General HSMM [23] ©2010 Elsevier.	18
2.7	Graphical representation of a GMM, HMM, and HSMM with 7 components [22] ©2018 EPFL.	20
2.8	Trajectory reproduction of two example letters	22
2.9	Non-rigid point set registration problem [5] ©2010 IEEE.	24
2.10	Affine registration using CPD [29]	25
2.11	(a) Real-time rendered fluid ellipsoid splatting. (b) Underlying simulation particles. (c) Particle clumping due to neighbor deficiencies. (d) with artificial pressure term. [37] ©2013 ACM.	31
3.1	5G Research Hub Munich. The image illustrates the room setup with different pieces of equipment.	39
3.2	Movo platfrom, LMT version. 1) Robotiq® Hand-e gripper 2) Intel® Real-sense camera 3) Botasys® F/T sensor 4) Kinova® Jaco2 manipulator 5) The roll actuator to extend the head motion with the new Robotis® Dynamixel MX-106 actuator 6) XIMEA® Stereo cameras 7) TYRO® Wireless emergency stop 8) Customized power regulators 9) Hp® embedded computers.	40
3.3	Butterworth filter result. The raw sensory data is shown in blue, and the result of a low pass filter is shown in orange.	42
3.4	The new Hand-e gripper and the 6 axis force torque sensor and the Intel Real-sense camera.	43
3.5	Illustrations of a peg-in-the-hole application and the results of the experiment.	44
3.6	LMT's shared autonomy interface, the graphical user interface (GUI) software.	45

3.7	Software architecture	45
3.8	PBF Algorithm[37]. ©2013 ACM.	46
3.9	Photo realistic high definition rendering pipeline in Unity3D on the left and sensor outputs on the right. All the robot sensors such as IMU, LiDAR, cameras, depth cameras, transformations, and joint force and position feedback are provided to the ROS side through RRS.	47
3.10	Operating Movo directly from the app using a tablet	48
3.11	YCB Object Set [82]. ©2015 IEEE.	49
3.12	Real liquids used for the experiments.	50
3.13	Marble position in 6 liquids after 0.5 seconds.	51
3.14	Qualitative comparison of 5 different liquids in simulations and reality.	52
4.1	Real-time motion planning for dual-arm Kinova [®] Movo platform using the proposed NMPC-MP for low-level shared autonomy in teleoperation. The orange object is the obstacle, and the dashed-red arrow denotes the desired motion. The dashed-yellow trajectory is the generated motion in order to avoid the obstacle. [1], ©2021 IEEE.	55
4.2	Linear MPC regulator DP vs. Batch approach	60
4.3	Example of DH in the distal convention	62
4.4	Linear and angular velocity from a single joint	65
4.5	Example of signed distance	72
4.6	The collision objects used to represent the Movo robot. In order to plan for both arms, we run the NMPC-MP twice. As a result, the environmental constraints for each hand are displayed as primitive shapes in blue from each NMPC-MP's perspective. [1], ©2021 IEEE.	73
4.7	System diagram for NMPC-MP [1], ©2021 IEEE.	76
4.8	Simulations with primitive obstacles for NMPC-MP	79
4.9	Stage costs and the closest distances to the obstacle	79
4.10	Actual stage costs for NMPC planners different phs and chs [1], ©2021 IEEE.	80
4.11	Trajectory of the end-effector during the motion of the manipulator considering octomap obstacle avoidance. [1], ©2021 IEEE.	81
4.12	Trajectories for task(ii). <i>Left</i> : NMPC-MP. <i>Middle</i> : RelaxedIK. <i>Right</i> : RRT-Connect. [1], ©2021 IEEE.	82
4.13	Trajectories for task(iii). <i>Left</i> : NMPC-MP. <i>Middle</i> : STOMP. <i>Right</i> : RRT-Connect. [1], ©2021 IEEE.	83
4.14	The first real experiment: NMPC-MP's static obstacle avoidance. [1], ©2021 IEEE.	84
4.15	The second real experiment: NMPC-MP's dynamic obstacle avoidance in motion. [1], ©2021 IEEE.	84
4.16	Cylinder representation of the Movo on the left and the Franka on the right.	86
4.17	The trajectory of Movo and Franka	87
4.18	Rotated goal to Movo and Franka	87
4.19	Virtual obstacle avoidance on Movo and Franka.	88
4.20	Demonstration of Franka's real-time capability to avoid dynamic obstacles in remote teleoperation. The dynamic obstacle here in this picture is the HTC [®] Vive tracker.	89

5.1	The robot-assisted remote feeding scenario is simulated as marble pouring with the Kinova [®] Movo platform using the proposed Skill-CPD for high-level skill refinement during the shared-autonomy control switching. P_{human} is the coordinate frame of the leader (here HTC Vive [®] controller), and P_{robot} is the coordinate frame of the follower. The main objective is to deliver the red marble inside the spoon to one of the three main pouring destinations represented by orange, red, and yellow containers. The robot continuously predicts the operator's intention, and in case of network disconnection, executes the task from the previously learned skill with minimum inconsistency. Red ξ_h denotes the human trajectory, blue ξ_a denotes the learned skill trajectory, green ξ_c denotes the refined trajectory, and $P_{t_{d1}}$ indicates the network disconnection point. The network reconnection point is indicated by $P_{t_{d2}}$ [9], ©2022 IEEE.	91
5.2	Skill-CPD System Diagram. ©2022 IEEE.	94
5.3	The picture on the right illustrates the ten demonstrated trajectories of the example letter B for the 2D scenario (in pixel). The picture on the left shows the five demonstrations of the marble pouring task to the red container for the 3D scenario (in meter). The HSMM clusters are shown in red. ©2022 IEEE.	94
5.4	(1.a) is the raw trajectory from the human operator which could contain outliers and noise. (1.b) indicates the trajectory after outlier removal and (1.c) denotes the final normalized trajectory. (2) illustrates the closest point detection strategy. (3) shows the reverse takeover step as a linear interpolation approach.	96
5.5	The simulation experiment for the task(i): 2D English letter drawing. The green dotted trajectory denotes the learned skill. Blue continuous trajectory is the desired input from the human operator (ground truth). The purple continuous trajectory is the system final output. (a) is the normal system without shared autonomy (direct teleoperation). (b) and (d) are the standard shared autonomy systems with rigid and non-rigid deformations. (c) and (e) represent our Skill-CPD system with same deformations. Yellow dotted trajectory is the registered skill.	96
5.6	The real experiment for the task(i): 2D English letter drawing. The gray dotted trajectory denotes the direct teleoperation and the red dotted trajectory denotes the autonomous controller output. In the second row, we applied Skill-CPD whereas in the first we didn't.	96
5.7	Closeup view of the 2D English letter drawing. We apply force feedback control to keep the marker on the whiteboard. 1) the 3d-printed marker holder, 2) Robotiq [®] Hand-gripper 3) Botasys [®] SensONE force/torque sensor 4) Kinova [®] Jaco-2 manipulator 5) Pan/Tilt/Roll unit 6) Network interface (5G UE) 7) Phantom-Omni [®] haptic master.	100
5.8	Motion generation and dataset creation environment.	100
5.9	The effect of Δd selection on CPD real-time performance	102
5.10	The real experiment for the task(iii): 3D robot-assisted feeding. It illustrates the different scenarios from robot prospective.	102
5.11	The real experiment for the task(iii): 3D robot-assisted feeding. It illustrates the different scenarios from robot prospective. The red marble (as a liquid handling task) for Skill-CPD remains in the container after the network impairment but due to the acceleration caused by the displacement error, it falls in other modes.	103
6.1	On the right, the simulation environment is illustrated during the pouring of a water-like liquid. On the left, the robot is pouring orange juice in a real environment using our trained policy network, PourNet. PourNet learns to perform natural human-like pouring actions through several simulated experiences [8]. ©2022 IEEE.	105
6.2	Training Architecture.©2022 IEEE.	106

6.3	Comparison of flow rates of low viscous fluids with the high viscous fluids. [103] [104] [105] [106].	107
6.4	Different types of containers with variation in geometrical features. [107]	108
6.5	PourNet Model Architecture.	110
6.6	Curriculum Learning lessons and parameters. The idea is to domain randomize the parameters in increasing order of their difficulty in pouring dynamics.	112
6.7	Performance Improvement by using ICM.	113
6.8	Performance Improvement by using Curriculum Learning.	113
6.9	Liquid Pouring with robot in the loop scene inside RRS-PL.	115
6.10	The simulated (top) vs real (bottom) liquids and containers. On the right the test environment with robot in the loop is illustrated.	116
6.11	a) Experimental Setup with Kinova Movo. b) Force measurement steps using F/T sensor. c) Second-order ButterWorth Low-pass Filter, operating at 100 Hz, with a cut-off frequency of 1 Hz.	117
6.12	a) Denotes the novel oil to spoon pouring experiment, container number 13 to number 14 b) Denotes the beer pouring failure.	119
6.13	Experiments using water and different containers.	119
6.14	Demonstration of pouring orange juice in the bowl.	120
6.15	Demonstration of pouring oil in the spoon.	120
6.16	Demonstration of generalization of the starting point.	120

List of Tables

1.1	Summarizing levels of autonomy in this dissertation.	2
2.1	Comparison of reinforcement learning algorithms.	29
2.2	The state-of-the-art motion controllers and planners comparison.	34
2.3	Comparison of the state-of-the-art shared autonomy-based teleoperation.	35
2.4	PourNet compared to the other state-of-the-art approaches.	37
3.1	Serial SensONE Force Torque Sensor from BOTA system.	43
3.2	Parameters of the new gripper.	44
3.3	Density calculation from liquids.	49
3.4	Measurements of different liquids for obtaining their viscosity	51
3.5	Liquids' parameters in RRS Simulator	52
4.1	DH table for Jaco2	61
4.2	Parameters for NMPC Motion Planner	78
4.3	Controller performance for task(i)	82
4.4	Planner/Controller performance for task(ii)	82
4.5	Planner/Controller performance for task(iii).	83
4.6	Modified Denavit-Hartenberg Parameters.	85
4.7	Joint Space Limits of Franka.	88
5.1	Performance Analysis in Simulation and Real Experiments	101
5.2	Parameters for Skill-CPD for task(i), task(ii), and task(iii)	101
6.1	Average Pouring deviation for water compared to the state-of-the-art approaches	117
6.2	Simulation Liquid Profiles (Flex Parameters)	117
6.3	LSTM-based PourNet: Experiment with liquid simulation profiles	118
6.4	Deviation from target using different liquids.	119

