UNIVERSITY OF CALIFORNIA SAN DIEGO

Differentiable Neural Motion Planning under Task Constraints

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Intelligent Systems, Robotics and Control)

by

Ahmed H. Qureshi

Committee in charge:

Professor Michael C. Yip, Chair
Professor Henrik Christensen
Professor Vikash Gilja
Professor Sonia Martinez
Professor Nuno Vasconcelos

2021

University of California San Diego

2021

DEDICATION

I dedicate this thesis to my family: My late father for his support towards my education and for his countless survival tips and life-learned lessons. My mother for her prayers and for always ensuring that I am doing well. My dear wife, for her untiring support and care. In all trials and tribulations, she stood by me, discussed my ideas, gave me hope and encouragement, and helped me rule out several wrong turns during the process. My older brother for being a true friend and for always having my back. My sisters for virtually engaging me in all family gatherings and making me feel like I'm at home even though I was several thousand miles away. My younger brother for always being a message away.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am very grateful to my supervisor, Prof. Michael Yip, for his expert mentoring and continuous support throughout our association. He has been an immense source of motivation, and I learned a great deal from him; which includes the technical, professional, and social skills required to progress in the world of academia.

I would also like to thank my committee members Prof. Henrik Christensen, Prof. Sonia Martinez, Prof. Vikash Gilja, and Prof. Nuno Vasconcelos for their support and for always taking out time for me from their busy schedules.

Furthermore, I offer my acknowledgement and thanks to Prof. Byron Boots for his profound guidance and technical support. He is also a co-author in two of my papers. I would also like to express my deepest gratitude to Dr. Arsalan Mousavian, Dr. Chris Paxton, and Prof. Dieter Fox. They are co-authors in one of my papers, and I thank them for their involvement and assistance during every step of the way while I conducted research in their robotics research group at NVIDIA. I would also like to acknowledge and thank Prof. Dmitry Berenson for his time and insightful discussion over the kinematic constraints in motion planning and their practical applications.

Moreover, I would also like to extend my gratitude to all the members of ARCLab, especially Dr. Ryan Orosco, Fei Liu, Nikhil Das, Florian Richter, Dimitri Schreiber, Jacob Johnson, Yuheng Zhi, Zih-Yun Chiu, Jingpei Lu, Nikhil Shinde, Jiangeng Dong, Linjun Li, Yinglong Miao, Asfiya Baig, Mrinal Verghese, Anthony Simeonov, Mayur Bency, Taylor Henderson, Harleen Singh, Yuzhe Qin, and Ojash Neopane, for their valuable company and support. I am also thankful to my colleagues and friends at Contextual Robotics Institute and UCSD, especially Maryam Pourabedi, Anwesan Pal, Carlos Nieto, Ehsan Ziaeikajbaf, Priyam Parashar, Ruffin White, Shengye Wang, Vikas Dhiman, Angelique Taylor, and Maria Harris, for their helpfulness.

Lastly, I am thankful to all members of the ECE student support office for aiding me in navigating through the administrative requirements of my degree.

This dissertation, in part, is a reprint of this dissertation author's publications. Chapter 1 and 6, in part, are reprint of dissertation author's publications.

Chapter 2, in part, is a reprint of **A.H.Qureshi**, B. Boots, and M.C.Yip,"Adversarial Imitation Via Variational Inverse Reinforcement Learning", *International Conference on Representation Learning (ICLR)*, 2019. The dissertation author is the primary author of this paper.

Chapter 3.1, in part, is a reprint of the following papers. The dissertation author is the primary author of these papers.

- **A.H.Qureshi**, Y.Miao, A.Simeonov, and M.C.Yip,"Motion Planning Networks: Bridging the Gap Between Learning-based and Classical Motion Planners", *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48-66, 2021.
- **A.H.Qureshi**, A.Simeonov, M.J.Bency, and M.C.Yip, "Motion Planning Networks", *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2118-2124, Montreal, Canada, 2019.
- **A.H.Qureshi** and Michael.C.Yip, "Deeply Informed Neural Sampling For Robot Motion Planning", *IEEE International Conference on Intelligent Robot and Systems (IROS)*, pp. 6582-6588, 2018.

Chapter 3.2, in part, is a reprint of L. Li, Y. Miao, **A. H. Qureshi** and M. C. Yip,"MPC-MPNet: Model-Predictive Motion Planning Networks for Fast, Near-Optimal Planning Under Kinodynamic Constraints", *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4496-4503, July 2021, doi: 10.1109/LRA.2021.3067847. The dissertation author is the co-author of this paper.

Chapter 3.3, in part, is a reprint of the following papers. The dissertation author is the primary author of these papers.

- **A.H.Qureshi**, J.Dong, A.Baig, and M.C.Yip,"Constrained Motion Planning Networks X", *IEEE Transactions on Robotics*, 2021.

- **A.H.Qureshi**, J.Dong, A.Choe, and M.C.Yip, “Neural Manipulation Planning on the Constraint Manifolds”, *IEEE Robotics and Automation Letters*, 2020.

Chapter 4, in part, is a reprint of **A.H.Qureshi**, J. J. Johnson, Y. Qin, T. West, B. Boots, and M.C.Yip. “Composing Task-Agnostic Policies via Deep Reinforcement Learning”, *International Conference on Representation Learning (ICLR)*, 2020. The dissertation author is the primary author of this paper.

Chapter 5, in part, is a reprint of **A.H.Qureshi**, A.Mousavian, C.Paxton, M.C.Yip, and D.Fox, “NeRP: Neural Rearrangement Planning for Unknown Objects”, *Robotics: Science and Systems 2021*. The dissertation author is the primary author of this paper.

VITA

2010-2014	B. S. in Electrical Engineering, National University of Sciences and Technology, Pakistan.
2015-2017	M. S. in Engineering, Osaka University, Japan.
2017-2021	Ph. D. in Electrical Engineering (Intelligent Systems, Robotics and Control), University of California San Diego, USA.

PUBLICATIONS

A.H.Qureshi, J.Dong, A.Baig, and M.C.Yip, "Constrained Motion Planning Networks X", *IEEE Transactions on Robotics*, 2021.

A.H.Qureshi, A.Mousavian, C.Paxton, M.C.Yip, and D.Fox, "NeRP: Neural Rearrangement Planning for Unknown Objects", *Robotics: Science and Systems*, 2021.

L.Li, Y.Miao, **A.H.Qureshi**, and M.C.Yip, "MPC-MPNet: Model-Predictive Motion Planning Networks for Fast, Near-Optimal Planning under Kinodynamic Constraints", *IEEE Robotics and Automation Letters*, 2021.

A.H.Qureshi, J.Dong, A.Choe, and M.C.Yip, "Neural Manipulation Planning on the Constraint Manifolds", *IEEE Robotics and Automation Letters*, 2020.

A.H.Qureshi, Y.Miao, A.Simeonov, and M.C.Yip, "Motion Planning Networks: Bridging the Gap Between Learning-based and Classical Motion Planners", *IEEE Transactions on Robotics*, 2020.

J.Johnson, L.Li, F.Liu, **A.H.Qureshi**, and M.C.Yip, "Dynamically Constrained Motion Planning Networks for Non-Holonomic Robots", *IEEE International Conference on Intelligent Robot and Systems (IROS)*, pp. 6937-6943, Las Vegas, USA (Virtual) 2020.

A.H.Qureshi, J. J. Johnson, Y. Qin, T. West, B. Boots, and M.C.Yip, "Composing Task-Agnostic Policies with Deep Reinforcement Learning", *International Conference on Representation Learning (ICLR)*, 2020.

A.H.Qureshi, B. Boots, and M.C.Yip, "Adversarial Imitation Via Variational Inverse Reinforcement Learning", *International Conference on Representation Learning (ICLR)*, 2019.

M.C.Yip, M.J.Bency, **A.H.Qureshi**, "Machine Learning based Fixed-Time Optimal Path Generation", *US Patent App. 16/222,706*, 2019.

A.H.Qureshi, A.Simeonov, M.J.Bency, and M.C.Yip, "Motion Planning Networks", *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2118-2124, Montreal, Canada, 2019.

M.J.Bency, **A.H.Qureshi**, M.C.Yip. “Neural Path Planning: Fixed Time, Near-Optimal Path Generation via Oracle Imitation”, *IEEE International Conference on Intelligent Robot and Systems (IROS)*, pp. 3965-3972, Macau, 2019.

A.H.Qureshi and Michael.C.Yip, “Deeply Informed Neural Sampling For Robot Motion Planning”, *IEEE International Conference on Intelligent Robot and Systems (IROS)*, pp. 6582-6588, 2018.

ABSTRACT OF THE DISSERTATION

Differentiable Neural Motion Planning under Task Constraints

by

Ahmed H. Qureshi

Doctor of Philosophy in Electrical Engineering (Intelligent Systems, Robotics and Control)

University of California San Diego, 2021

Professor Michael C. Yip, Chair

Autonomous robots will soon play a significant role in various domains, such as search-and-rescue, agriculture farms, homes, offices, transportation, and medical surgery, where fast, safe, and optimal response to different situations will be critical. However, to do so, these robots need fast algorithms to plan their motion sequences in real-time with limited perception and battery life. The field of motion planning and control addresses this challenge of coordinating robot motions and enabling them to interact with their environments for performing various challenging tasks under constraints.

Planning algorithms for robot control have a long history ranging from methods with complete to probabilistically complete worst-case theoretical guarantees. However, despite having deep roots in artificial intelligence and robotics, these methods tend to be computationally inefficient in high-dimensional problems. On the other hand, machine learning advancements have led toward systems that can directly perform complex decision-making from raw sensory information. This thesis introduces a new class of planning methods called Neural Motion Planners that emerged from the cross-fertilization of classical motion planning and machine learning techniques. These methods can achieve unprecedented speed and robustness in planning robot motion sequences in complex, cluttered, and partially observable environments. They exhibit worst-case theoretical guarantees and solve a broad range of motion planning problems under geometric collision-avoidance, kinodynamic, non-holonomic, and hard kinematic manifold constraints.

Another challenge towards deploying robots into our natural world is the tedious process of defining objective functions for underlying motion planners and transferring and composing their motion skills into new skills for a combinatorial outburst in robot's skillset for solving unseen practical problems. To address these challenges, this thesis introduces novel methods, i.e., variational inverse reinforcement learning and compositional reinforcement learning approaches. These methods learn unknown constraint functions and their motion skills directly from expert demonstrations for NMPs and compose them into new complex skills for solving more complicated problems across different domains. Finally, this thesis also presents a model-free neural task planning algorithm that works with never-before-seen objects and generalizes to real world environments. It generates task plans for underlying motion planning and control approaches and solves challenging rearrangement tasks in unknown environments.

Chapter 1

Introduction

As an infant over time, their surrounding environments demonstrate them a variety of concepts. For example, they observe people walking, talking, eating, and doing other social interactions within their surroundings. These abstract concepts are translated into various behavioral skills such as they learn to sit, crawl, stand, and walk.These skills are then transferred to new domains such as crawling over flat ground to crawling upstairs and composed together into more complex behaviors such as running, rock climbing, cycling, etc. In all such scenarios, our brain also layouts a task plan and decomposes a given high-level objective into sub-goals and achieves them by using over-the-time acquired skills.

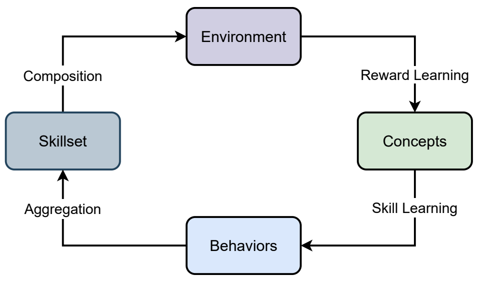


Figure 1.1: A development process includes understanding demonstrated concepts and learning corresponding skills, and composing them into new skills for new tasks.

Inspired from our cognitive and motor skill development process, this dissertation introduces novel deep learning-based algorithms to enable robots to acquire their motion skills from demonstrated concepts, and compose them together into complex, new skills (Fig. 1.1). It also highlights that our learning-based algorithms naturally form a mutualistic relationship with existing task planning methods that breaks a high-level task into a sequence of subtasks. In addition, this dissertation also introduces a novel model-free neural task planner that can find multi-step, long-horizon, intermediate sub-task sequences from high-dimensional sensory information.

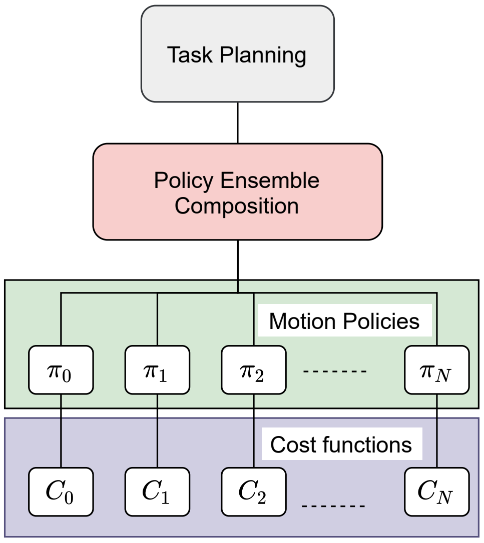


Figure 1.2: This dissertation follows a bottom-up approach from learning concepts (e.g., task constraints or cost functions) and their behavioral skills to composing them into new skills under a high-level task planner.

During execution, a task planner observes a high-level task and decomposes it into a sequence of sub-tasks. The composition framework observes a given sub-task and achieves it by composing available motion skills. These motion skills and their objective functions are learned from behaviors demonstrated by an expert. This dissertation follows a bottom-up approach (Fig. 1.2):

- First, it introduces a novel technique to learn an optimizable objective function from demonstrations, encapsulating concepts and task constraints.
- Second, given an optimizable objective function, it presents a new class of motion planning

methods called neural motion planners that operate efficiently under various task constraints.

- Third, it offers a novel composition framework that can transfer given motion skills to new domains and combine them sequentially and concurrently to accomplish a given task.
- Fourth, it introduces a novel task planner to predict multi-step intermediate task sequences from raw sensory information to solve rearrangement planning problems in real-world environments.

1.1 Task Constraints Learning

Reinforcement learning (RL) has emerged as a promising tool for solving complex decision-making and control tasks from predefined high-level reward functions [SB$^+$98]. However, defining an optimizable reward function that inculcates the desired behavior can be challenging for many robotic applications, which include learning social-interaction skills [QNYI18, QNYI17], dexterous manipulation [FLA16], and autonomous driving [KGB15].

Inverse reinforcement learning (IRL) [NR$^+$00] addresses the problem of learning constraint functions, also known as cost or reward functions, from expert demonstrations, and it is often considered as a branch of imitation learning [ACVB09]. The prior work in IRL includes maximum-margin [AN04, RBZ06] and maximum-entropy [ZMBD08] formulations. Currently, maximum entropy (MaxEnt) IRL is a widely used approach towards IRL, and has been extended to use non-linear function approximators such as neural networks in scenarios with unknown dynamics by leveraging sampling-based techniques [BKP11, FLA16, KPRS13]. However, designing the IRL algorithm is usually complicated as it requires, to some extent, hand engineering such as deciding domain-specific regularizers [FLA16].

Chapter 2 of this dissertation presents a novel IRL approach known as Empowerment-regularized Adversarial Inverse Reinforcement Learning (EAIRL) [QBY19]. EAIRL learns both reward and policy functions directly from expert demonstrations. The recovered rewards, encapsulating task constraints, are shown to be near-optimal and transferable to new environments with different structures and agent dynamics.

1.2 Motion Planning

Motion planning is among the core research problems in robotics and artificial intelligence with its application spanning from autonomous driving to space exploration. It aims to find a collision-free, low-cost path connecting a start and goal states for an agent under task constraints [LaV06] [Lat12]. These task constraints forms an abstract space within robot state-space that comprises a forbidden space and a feasible space (Fig. 1.3). An ideal motion planning algorithm finds a global solution in the feasible space for solving given problems and offers following key features: (i) completeness and optimality guarantees - implying that a solution will be found if one exists and that the solution will be globally optimal satisfying all given constraints, (ii) computational efficiency - finding a solution in either real-time or in sub-second times or better while being memory efficient, and (iii) insensitivity to environment complexity - the algorithm is effective and efficient in finding solutions regardless of the constraints of the environment. Decades of research have produced many significant milestones for motion planning include resolution-complete planners such artificial potential fields [Kha86], sample-based motion planners such as Rapidly-Exploring Random Trees (RRT) [LaV06] and its optimal variant RRT* [KF11], heuristically biased solvers such as [GSB15] and lazy search methods [HMP+18]. However, each planner and their variants have tradeoffs amongst the ideal features of motion planners and often consider simple collision avoidance constraints. Thus, no single motion planner has emerged above all others to solve a broad range of problems.

Chapter 3 of this dissertation presents a new class of motion planning methods called Neural Motion Planners (NMPs) [QSBY19, QY18, QMSY20, QDCY20, QDBY21, LMQY21] with all key features of an ideal planner. NMPs bridge the gap between machine learning and

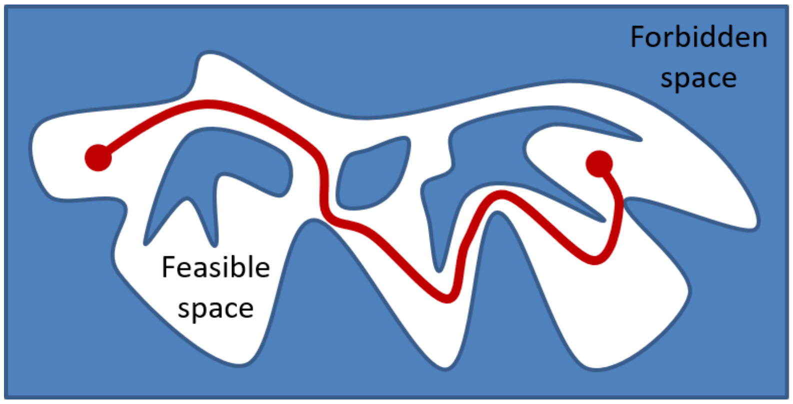


Figure 1.3: An abstract feasible and forbidden space within robot state space formed by the task constraints. A motion planner determines a path in the feasible space that connects the given start and goal.

classical motion planning, leveraging advancement in the latter and the algorithm structures in the former field to present methods that solve various problems under all sort of task constraints at nearly real-time speed. These methods are shown to learn with high data efficiency from streaming data, under a life-long learning setting, and solve a wide range of practical robot motion planning problems in seconds where other traditional methods take up to several minutes.

1.3 Composition

Compositionality is the integration of primitive functions into new complex functions that can further be composed into even more complex functions to solve novel problems [KLP17]. Evidence from neuroscience and behavioral biology research shows that humans and animals have the innate ability to transfer their basic skills to new domains and compose them hierarchically into complex behaviors [RFG01]. For instance, studies on the kicking motion of frogs in water and the flapping motion of birds in flight revealed the composition of multiple muscle units to achieve the overall complex behavior [MIB00]. In robotics and general-purpose artificial intelligence, compositionality corresponds to the combinations of different motion policies/controllers, visual representations or language models [LUTG17]. In robotics, the primary focus is on acquiring new behaviors rather than composing and re-using the already acquired skills to solve novel, unseen tasks [LUTG17].

Most real-world robotics control problems can efficiently be solved if there exists a composition method that can compose primitive behaviors into complex behaviors. For instance, instead of learning a single policy for robot locomotion, a neurologically motivated and simplistic approach could be to acquire basic skills such as moving left, right, up, and down, and compose those skills to solve complex locomotion tasks such as reaching a particular location. Similarly, in the case of autonomous driving, the low-level policies could also include operations such as braking, acceleration, obstacle avoidance, over-taking, etc. However, a fundamental problem that emerges after basic skills acquisition is to build a system that can compose such skills to achieve any given objective. Chapter 4 of this dissertation presents a novel deep reinforcement learning-based approach [QJQ$^+$20] for composing various, task-agnostic motion policies into complex skills for solving new tasks.

1.4 Task Planning

Task planning is a crucial component in robotics, especially in unstructured environments where robots would have to perform various intermediate tasks before achieving a given abstract goal. For instance, our cognitive process decomposes a given task (e.g., cleaning) into subtasks (e.g., moving objects to their designated places). It accomplishes them sequentially or concurrently by sending motor commands to the body for physical interaction with the environment under the task-specific constraints [CS00, ZSS$^+$07]. In robotics, decomposing a high-level task, e.g., cleaning, into a sequence of sub-tasks, e.g., opening the cabinet, moving objects into the cabinet, and closing the cabinet, is known as Task Planning. Formally, a task planner decomposes a given task into a sequence of sub-tasks, and a motion planner achieves those sub-tasks by planning feasible robot motion sequences. This dissertation shows that the NMPs form a mutual symbiotic relationship with existing learning-based task planners. The learning-based task planners generate intermediate task sequences and task representations for NMPs. NMPs directly leverage those representations to quickly solve corresponding intermediate tasks, leading to a next environment observation for the task planners.

A subset problem of task planning known as the rearrangement of unknown objects has recently been identified as a major challenge problem for embodied AI, especially robotics [BCC$^+$20]. Since many real-world robotic tasks boil down to pick-and-place, but with a much wider diversity of objects and scenarios than we typically see in the lab. Robots may also encounter cluttered scenes and blocked goals, cases where typically we might need to perform much longer horizon planning to clear objects and move things out of the way. Chapter 5 of this dissertation addresses this challenge by introducing a new rearrangement task planner [AQF21] that, unlike prior work, operates directly in high-dimensional spaces and solves long-horizon rearrangement problems in never-seen-before, partially observable environments.

1.5 Acknowledgements

Chapter 1, in part, is a reprint of the following publications:

- **A.H.Qureshi**, B. Boots, and M.C.Yip,"Adversarial Imitation Via Variational Inverse Reinforcement Learning", *International Conference on Representation Learning (ICLR)*, 2019. The dissertation author is the primary author of this paper.

- **A.H.Qureshi**, Y.Miao, A.Simeonov, and M.C.Yip,"Motion Planning Networks: Bridging the Gap Between Learning-based and Classical Motion Planners", *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48-66, Feb. 2021, doi: 10.1109/TRO.2020.3006716. The dissertation author is the main of this paper.

- **A.H.Qureshi**, A.Simeonov, M.J.Bency, and M.C.Yip, "Motion Planning Networks", *IEEE*

International Conference on Robotics and Automation (ICRA), pp. 2118-2124, Montreal, Canada, 2019. The dissertation author is the primary author of this paper.

- **A.H.Qureshi** and Michael.C.Yip, "Deeply Informed Neural Sampling For Robot Motion Planning", *IEEE International Conference on Intelligent Robot and Systems (IROS)*, pp. 6582-6588, 2018. The dissertation author is the primary author of this paper.

- L. Li, Y. Miao, **A. H. Qureshi** and M. C. Yip,"MPC-MPNet: Model-Predictive Motion Planning Networks for Fast, Near-Optimal Planning Under Kinodynamic Constraints", *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4496-4503, July 2021, doi: 10.1109/LRA.2021.3067847. The dissertation author is the co-author of this paper.

- **A.H.Qureshi**, J.Dong, A.Baig, and M.C.Yip,"Constrained Motion Planning Networks X", *IEEE Transactions on Robotics*, 2021. The dissertation author is the primary author of this paper.

- **A.H.Qureshi**, J.Dong, A.Choe, and M.C.Yip, "Neural Manipulation Planning on the Constraint Manifolds", *IEEE Robotics and Automation Letters*, 2020. The dissertation author is the primary author of this paper.

- **A.H.Qureshi**, J. J. Johnson, Y. Qin, T. West, B. Boots, and M.C.Yip. "Composing Task-Agnostic Policies via Deep Reinforcement Learning", *International Conference on Representation Learning (ICLR)*, 2020. The dissertation author is the primary author of this paper.

  A.H.Qureshi, A.Mousavian, C.Paxton, M.C.Yip, and D.Fox, "NeRP: Neural Rearrangement Planning for Unknown Objects", *Robotics: Science and Systems 2021*. The dissertation author is the primary author of this paper.

Chapter 2

Task Constraints Learning

Constraint or reward learning is challenging as there can be many optimal policies explaining a set of demonstrations and many reward functions inducing an optimal policy [NR$^+$00, ZMBD08]. Rather than learning reward functions and solving the IRL problem, imitation learning (IL) learns a policy directly from expert demonstrations. Prior work addressed the IL problem through behavior cloning (BC), which learns a policy from expert trajectories using supervised learning [Pom91]. Although BC methods are simple solutions to IL, these methods require a large amount of data because of compounding errors induced by covariate shift [RGB11]. To overcome BC limitations, a generative adversarial imitation learning (GAIL) algorithm [HE16] was proposed. GAIL uses the formulation of Generative Adversarial Networks (GANs) [GPAM$^+$14], i.e., a generator-discriminator framework, where a generator is trained to generate expert-like trajectories while a discriminator is trained to distinguish between generated and expert trajectories. Although GAIL is highly effective and efficient framework, it does not recover transferable/portable reward functions along with the policies, thus narrowing its use cases to similar problem instances in similar environments. Reward function learning is ultimately preferable, if possible, over direct imitation learning as rewards are portable functions that represent the most basic and complete representation of agent intention, and can be re-optimized in new environments and new agents.

Recently, an Adversarial Inverse Reinforcement Learning (AIRL) framework [FLL17], an extension of GAIL, was proposed that offers a solution to transferable reward learning by exploiting the maximum entropy Inverse Reinforcement Learning (IRL) method [ZMBD08]. However, it learns transferable reward functions by modeling the reward as a function of state only instead of both state and action. This makes AIRL fail to recover the ground truth reward when the ground truth reward is a function of both state and action. For example, the reward function in any locomotion or ambulation tasks contains a penalty term that discourages actions with large magnitudes. This need for action regularization is well known in optimal control literature and limits the use cases of a state-only reward function in most practical real-life applications. A more generalizable and useful approach would be to formulate reward as a function of both states and actions, which induces action-driven reward shaping that has been shown to play a vital role in quickly recovering the optimal policies [NHR99].

In this work, we discuss our Variational IRL approach, i.e., Empowerment-regularized Adversarial Inverse Reinforcement Learning (EAIRL) algorithm[1] [QBY19]. Empowerment [SGP14] is a mutual information-based theoretic measure, like state- or action-value functions, that assigns a value to a given state to quantify the extent to which an agent can influence its environment. Our method uses variational information maximization [MR15] to learn empowerment in parallel to learning the reward and policy from expert data. Empowerment acts as a regularizer to policy updates to prevent overfitting the expert demonstrations, which in practice leads to learning robust rewards. Our experimentation shows that the proposed method recovers not only near-optimal policies but also recovers robust, transferable, disentangled, state-action based reward functions that are near-optimal. The results on reward learning also show that EAIRL outperforms several state-of-the-art IRL methods by recovering reward functions that leads to optimal, expert-matching behaviors. On policy learning, results demonstrate that policies learned

[1]Supplementary material is available at `https://sites.google.com/view/eairl`

through EAIRL perform comparably to GAIL and AIRL with non-disentangled (state-action) reward function but significantly outperform policies learned through AIRL with disentangled reward (state-only) and GAN interpretation of Guided Cost Learning (GAN-GCL) [FCAL16].

2.1 Preliminaries

We consider a Markov decision process (MDP) represented as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)$ where $\mathcal{S}$ denotes the state-space, $\mathcal{A}$ denotes the action-space, $\mathcal{P}$ represents the transition probability distribution, i.e., $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$, $\mathcal{R}(s,a)$ corresponds to the reward function, ρ_0 is the initial state distribution $\rho_0 : \mathcal{S} \rightarrow \mathbb{R}$, and $\gamma \in (0,1)$ is the discount factor. Let $q(a|s,s')$ be an inverse model that maps current state $s \in \mathcal{S}$ and next state $s' \in \mathcal{S}$ to a distribution over actions $\mathcal{A}$, i.e., $q : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$. Let π be a stochastic policy that takes a state and outputs a distribution over actions such that $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$. Let τ and τ_E denote a set of trajectories, a sequence of state-action pairs $(s_0, a_0, \cdots s_T, a_T)$, generated by a policy π and an expert policy π_E, respectively, where T denotes the terminal time. Finally, let $\Phi(s)$ be a potential function that quantifies a utility of a given state $s \in \mathcal{S}$, i.e., $\Phi : \mathcal{S} \rightarrow \mathbb{R}$. In our proposed work, we use an empowerment-based potential function $\Phi(\cdot)$ to regularize policy update under MaxEnt-IRL framework. Therefore, the following sections provide a brief background on MaxEnt-IRL, adversarial reward and policy learning, and variational information-maximization approach to learn the empowerment.

MaxEnt-IRL

MaxEnt-IRL [ZMBD08] models expert demonstrations as Boltzmann distribution using parametrized reward $r_\xi(\tau)$ as an energy function, i.e.,

$$p_\xi(\tau) = \frac{1}{Z} \exp(r_\xi(\tau)) \tag{2.1}$$

where $r_\xi(\tau) = \sum_{t=0}^{T} r_\xi(s_t, a_t)$ is a commutative reward over given trajectory τ, parameterized by ξ, and Z is the partition function. In this framework, the demonstration trajectories are assumed to be sampled from an optimal policy π^*, therefore, they get the highest likelihood whereas the suboptimal trajectories are less rewarding and hence, are generated with exponentially decaying probability. The main computational challenge in MaxEnt-IRL is to determine Z. The initial work in MaxEnt-IRL computed Z using dynamic programming [ZMBD08] whereas modern approaches [FLA16, FCAL16, FLL17] present importance sampling technique to approximate Z under unknown dynamics.

Adversarial Inverse Reinforcement Learning

This section briefly describes Adversarial Inverse Reinforcement Learning (AIRL) [FLL17] algorithm which forms a baseline of our proposed method. AIRL is the current state-of-the-art IRL method that builds on GAIL [HE16], maximum entropy IRL framework [ZMBD08] and GAN-GCL, a GAN interpretation of Guided Cost Learning [FLA16, FCAL16].

GAIL is a model-free adversarial learning framework, inspired from GANs [GPAM$^+$14], where the policy π learns to imitate the expert policy behavior π_E by minimizing the Jensen-Shannon divergence between the state-action distributions generated by π and the expert state-action distribution by π_E through following objective

$$\min_{\pi} \max_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_{\pi}[\log D(s,a)] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi) \tag{2.2}$$

where D is the discriminator that performs the binary classification to distinguish between samples generated by π and π_E, λ is a hyper-parameter, and $H(\pi)$ is an entropy regularization term $\mathbb{E}_{\pi}[\log \pi]$. Note that GAIL does not recover reward; however, [FCAL16] shows that the discriminator can be modeled as a reward function. Thus AIRL [FLL17] presents a formal implementation of [FCAL16] and extends GAIL to recover reward along with the policy by imposing a following structure on the discriminator:

$$D_{\xi,\varphi}(s,a,s') = \frac{\exp[f_{\xi,\varphi}(s,a,s')]}{\exp[f_{\xi,\varphi}(s,a,s')] + \pi(a|s)} \tag{2.3}$$

where $f_{\xi,\varphi}(s,a,s') = r_\xi(s) + \gamma h_\varphi(s') - h_\varphi(s)$ comprises a disentangled reward term $r_\xi(s)$ with training parameters ξ, and a shaping term $F = \gamma h_\varphi(s') - h_\varphi(s)$ with training parameters φ. The entire $D_{\xi,\varphi}(s,a,s')$ is trained as a binary classifier to distinguish between expert demonstrations τ_E and policy generated demonstrations τ. The policy is trained to maximize the discriminative reward $\hat{r}(s,a,s') = \log(D(s,a,s') - \log(1 - D(s,a,s')))$. Note that the function $F = \gamma h_\varphi(s') - h_\varphi(s)$ consists of free-parameters as no structure is imposed on $h_\varphi(\cdot)$, and as mentioned in [FLL17], the reward function $r_\xi(\cdot)$ and function F are tied upto a constant $(\gamma - 1)c$, where $c \in \mathbb{R}$; thus the impact of F, the shaping term, on the recovered reward r is quite limited and therefore, the benefits of reward shaping are not fully realized.

Empowerment as Maximal Mutual Information

Mutual information (MI), an information-theoretic measure, quantifies the dependency between two random variables. In intrinsically-motivated reinforcement learning, a maximal of mutual information between a sequence of K actions a and the final state s' reached after the execution of a, conditioned on current state s is often used as a measure of internal reward [MR15], known as Empowerment $\Phi(s)$, i.e.,

$$\Phi(s) = \max I(a,s'|s) = \max \mathbb{E}_{p(s'|a,s)w(a|s)} \left[\log \left(\frac{p(a,s'|s)}{w(a|s)p(s'|s)} \right) \right] \tag{2.4}$$

where $p(s'|a,s)$ is a K-step transition probability, $w(a|s)$ is a distribution over a, and $p(a,s'|s)$ is a joint-distribution of K actions a and final state s'[2]. Intuitively, the empowerment $\Phi(s)$ of a state

[2]In our proposed work, we consider only immediate step transitions i.e., $K = 1$, hence variables s, a and s' will be represented in non-bold notations.

s quantifies an extent to which an agent can influence its future. Thus, maximizing empowerment induces an intrinsic motivation in the agent that enforces it to seek the states that have the highest number of future reachable states.

Empowerment, like value functions, is a potential function that has been previously used in reinforcement learning but its applications were limited to small-scale cases due to computational intractability of MI maximization in higher-dimensional problems. Recently, however, a scalable method [MR15] was proposed that learns the empowerment through the more-efficient maximization of variational lower bound, which has been shown to be equivalent to maximizing MI [Aga04]. The lower bound was derived (for complete derivation see Section 2.5) by representing MI in term of the difference in conditional entropies $H(\cdot)$ and utilizing the non-negativity property of KL-divergence, i.e.,

$$I^{w}(s) = H(a|s) - H(a|s', s) \geq H(a) + \mathbb{E}_{p(s'|a,s)w_{\theta}(a|s)}[\log q_{\phi}(a|s', s)] = I^{w,q}(s) \tag{2.5}$$

where $H(a|s) = -\mathbb{E}_{w(a|s)}[\log w(a|s)]$, $H(a|s', s) = -\mathbb{E}_{p(s'|a,s)w(a|s)}[\log p(a|s', s)]$, $q_{\phi}(\cdot)$ is a variational distribution with parameters ϕ and $w_{\theta}(\cdot)$ is a distribution over actions with parameters θ. Finally, the lower bound in Eqn. 2.5 is maximized under the constraint $H(a|s) < \eta$ (prevents divergence, see [MR15]) to compute empowerment as follow:

$$\Phi(s) = \max_{w,q} \mathbb{E}_{p(s'|a,s)w(a|s)}[-\frac{1}{\beta}\log w_{\theta}(a|s) + \log q_{\phi}(a|s', s)] \tag{2.6}$$

where β is η dependent temperature term.

[MR15] also applied the principles of Expectation-Maximization (EM) [Aga04] to learn empowerment, i.e., alternatively maximizing Eqn. 2.6 with respect to $w_{\theta}(a|s)$ and $q_{\phi}(a|s', s)$. Given a set of training trajectories τ, the maximization of Eqn. 2.6 w.r.t $q_{\phi}(\cdot)$ is shown to be a supervised maximum log-likelihood problem whereas the maximization w.r.t $w_{\theta}(\cdot)$ is determined through the functional derivative $\partial I/\partial w = 0$ under the constraint $\sum_a w(a|s) = 1$. The optimal w^* that maximizes Eqn. 2.6 turns out to be $\frac{1}{Z(s)}\exp(\beta\mathbb{E}_{p(s'|s,a)}[\log q_\phi(a|s,s')])$, where $Z(s)$ is a normalization term. Substituting w^* in Eqn. 2.6 showed that the empowerment $\Phi(s) = \frac{1}{\beta}\log Z(s)$ (for full derivation, see Section 2.5).

Note that $w^*(a|s)$ is implicitly unnormalized as there is no direct mechanism for sampling actions or computing $Z(s)$. [MR15] introduced an approximation $w^*(a|s) \approx \log\pi(a|s) + \Phi(s)$ where $\pi(a|s)$ is a normalized distribution which leaves the scalar function $\Phi(s)$ to account for the normalization term $\log Z(s)$. Finally, the parameters of policy π and scalar function Φ are optimized by minimizing the discrepancy, $l_I(s,a,s')$, between the two approximations $(\log\pi(a|s) + \Phi(s))$ and $\beta\log q_\phi(a|s',s))$ through either absolute ($p = 1$) or squared error ($p = 2$), i.e.,

$$l_I(s,a,s') = \left|\beta\log q_\phi(a|s',s) - (\log\pi_\theta(a|s) + \Phi_\varphi(s))\right|^p \tag{2.7}$$

2.2 Variational Inverse Reinforcement Learning

We present an inverse reinforcement learning algorithm that learns a robust, transferable reward function and policy from expert demonstrations. Our proposed method comprises (i) an inverse model $q_\phi(a|s',s)$ that takes the current state s and the next state s' to output a distribution over actions $\mathcal{A}$ that resulted in s to s' transition, (ii) a reward $r_\xi(s,a)$, with parameters ξ, that is a function of both state and action, (iii) an empowerment-based potential function $\Phi_\varphi(\cdot)$ with parameters φ that determines the reward-shaping function $F = \gamma\Phi_\varphi(s') - \Phi_\varphi(s)$ and also regularizes the policy update, and (iv) a policy model $\pi_\theta(a|s)$ that outputs a distribution over actions given the current state s. All these models are trained simultaneously based on the objective functions described in the following sections to recover optimal policies and generalizable reward functions concurrently.

Inverse model $q_\phi(a|s,s')$ optimization

As mentioned in Section 2.1, learning the inverse model $q_\phi(a|s,s')$ is a maximum log-likelihood supervised learning problem. Therefore, given a set of trajectories $\tau \sim \pi$, where a single trajectory is a sequence states and actions, i.e., $\tau_i = \{s_0, a_0, \cdots, s_T, a_T\}_i$, the inverse model $q_\phi(a|s',s)$ is trained to minimize the mean-square error between its predicted action $q(a|s',s)$ and the action a taken according to the generated trajectory τ, i.e.,

$$l_q(s,a,s') = (q_\phi(\cdot|s,s') - a)^2 \tag{2.8}$$

Empowerment $\Phi_\varphi(s)$ optimization

Empowerment will be expressed in terms of normalization function $Z(s)$ of optimal $w^*(a|s)$, i.e., $\Phi_\varphi(s) = \frac{1}{\beta}\log Z(s)$. Therefore, the estimation of empowerment $\Phi_\varphi(s)$ is approximated by minimizing the loss function $l_I(s,a,s')$, presented in Eqn. 2.7, w.r.t parameters φ, and the inputs (s,a,s') are sampled from the policy-generated trajectories τ.

Reward function $r_\xi(s,a)$

To train the reward function, we first compute the discriminator as follow:

$$D_{\xi,\varphi}(s,a,s') = \frac{\exp[r_\xi(s,a) + \gamma\Phi_{\varphi'}(s') - \Phi_\varphi(s)]}{\exp[r_\xi(s,a) + \gamma\Phi_{\varphi'}(s') - \Phi_\varphi(s)] + \pi_\theta(a|s)} \tag{2.9}$$

where $r_\xi(s,a)$ is the reward function to be learned with parameters ξ. We also maintain the target φ' and learning φ parameters of the empowerment-based potential function. The target parameters φ' are a replica of φ except that the target parameters φ' are updated to learning parameters φ after every n training epochs. Note that keeping a stationary target $\Phi_{\varphi'}$ stabilizes the learning as also mentioned in [MKS$^+$15]. Finally, the discriminator/reward function parameters ξ are trained via binary logistic regression to discriminate between expert τ_E and generated τ trajectories, i.e.,

$$\mathbb{E}_\tau[\log D_{\xi,\varphi}(s,a,s')] + \mathbb{E}_{\tau_E}[(1-\log D_{\xi,\varphi}(s,a,s'))] \tag{2.10}$$

Policy optimization policy $\pi_\theta(a|s)$

We train our policy $\pi_\theta(a|s)$ to maximize the discriminative reward and to minimize the loss function $l_I(s,a,s')$ which accounts for empowerment regularization. Hence, the overall policy training objective is:

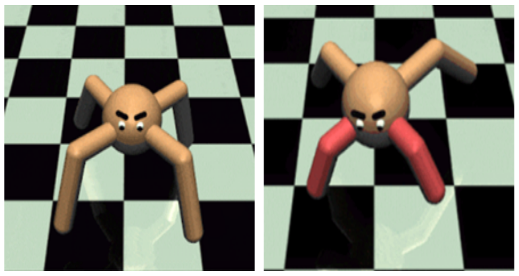
$$\mathbb{E}_\tau[\log \pi_\theta(a|s)\hat{r}(s,a,s')] + \lambda_I \mathbb{E}_\tau\left[l_I(s,a,s')\right] \tag{2.11}$$

where policy parameters θ are updated using any policy optimization method such as TRPO [SLA+15] or an approximated step such as PPO [SWD+17].

Algorithm 1: Empowerment-based Adversarial Inverse Reinforcement Learning

```
Initialize parameters of policy $\pi_\theta$, and inverse model $q_\phi$
Initialize parameters of target $\Phi_{\varphi'}$ and training $\Phi_\varphi$ empowerment, and reward $r_\xi$
  functions
Obtain expert demonstrations $\tau_E$ by running expert policy $\pi_E$
for $i \leftarrow 0$ to $N$ do
    Collect trajectories $\tau$ by executing $\pi_\theta$
    Update $\phi_i$ to $\phi_{i+1}$ with the gradient $\mathbb{E}_\tau[\bigtriangledown_{\phi_i} l_q(s,a,s')]$
    Update $\varphi_i$ to $\varphi_{i+1}$ with the gradient $\mathbb{E}_\tau[\bigtriangledown_{\varphi_i} l_I(s,a,s')]$
    Update $\xi_i$ to $\xi_{i+1}$ with the gradient:
        $\mathbb{E}_\tau[\bigtriangledown_{\xi_i} \log D_{\xi_i,\varphi_{i+1}}(s,a,s')] + \mathbb{E}_{\tau_E}[\bigtriangledown_{\xi_i}(1-\log D_{\xi_i,\varphi_{i+1}}(s,a,s'))]$
    Update $\theta_i$ to $\theta_{i+1}$ using natural gradient update rule (i.e., TRPO/PPO) with the
      gradient:
        $\mathbb{E}_\tau\left[\bigtriangledown_{\theta_i} \log \pi_{\theta_i}(a|s)\hat{r}_{\xi_{i+1}}(s,a,s')\right] + \lambda_I \mathbb{E}_\tau\left[\bigtriangledown_{\theta_i} l_I(s,a,s')\right]$
    After every $n$ epochs sync $\varphi'$ with $\varphi$
```

Algorithm 1 outlines the overall training procedure to train all function approximators


(a) Ant environment

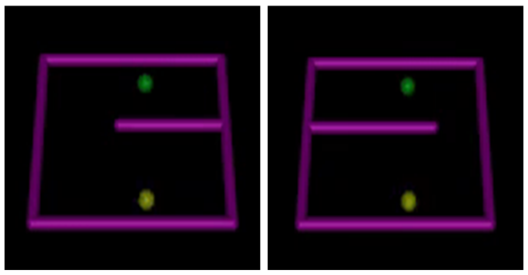

(b) Pointmass-maze environment

Figure 2.1: Transfer learning problems: reward transfer from (a) a quadruped-ant to a crippled-ant. (b) from a left-passage maze to a right-passage maze.

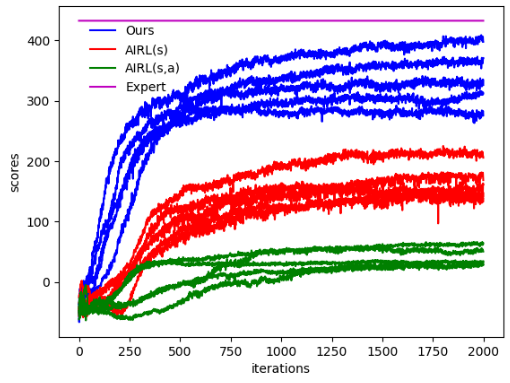
simultaneously. Note that the expert samples τ_E are seen by the discriminator only, whereas all other models are trained using the policy generated samples τ. Furthermore, the discriminating reward $\hat{r}(s,a,s')$ boils down to the following expression, as shown in Section 2.5:

$$\hat{r}(s,a,s') = f(s,a,s') - \log \pi(a|s)$$

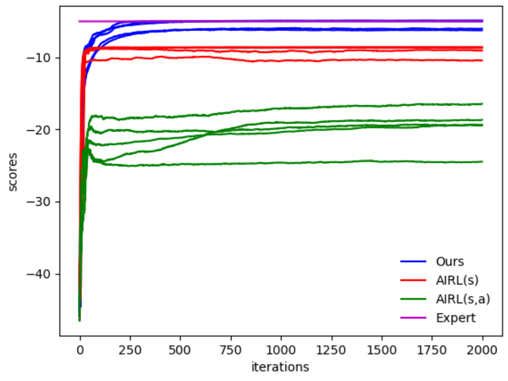
where $f(s,a,s') = r_\xi(s,a) + \gamma\Phi_{\varphi'}(s') - \Phi_\varphi(s)$. Thus, an alternative way to express our policy training objective is $\mathbb{E}_\tau[\log \pi_\theta(a|s) r_\pi(s,a,s')]$, where $r_\pi(s,a,s') = \hat{r}(s,a,s') - \lambda_I l_I(s,a,s')$, which would undoubtedly yield the same results as Eqn. 2.11, i.e., maximize the discriminative reward and minimize the loss l_I. The analysis of this alternative expression is given in Section 2.5 to highlight that our policy update rule is equivalent to MaxEnt-IRL policy objective [FCAL16] except that it also maximizes the empowerment, i.e.,

$$r_\pi(s,a,s') = r_\xi(s,a,s') + \gamma\Phi(s') + \lambda\hat{H}(\cdot) \tag{2.12}$$

where, λ and γ are hyperparameters, and $\hat{H}(\cdot)$ is the entropy-regularization term depending on $\pi(\cdot)$ and $q(\cdot)$. Hence, our policy is regularized by the empowerment which induces generalized behavior rather than locally overfitting to the limited expert demonstrations.



(a) Ant environment



(b) Pointmass-maze environment

Figure 2.2: The performance of policies obtained from maximizing the learned rewards in the transfer learning problems over five trials.

2.3 Results

Our proposed method, EAIRL, learns both reward and policy from expert demonstrations. Thus, for comparison, we evaluate our method against both state-of-the-art policy and reward learning techniques on several control tasks in OpenAI Gym. In case of policy learning, we compare our method against GAIL, GAN-GCL, AIRL with state-only reward, denoted as AIRL(s), and an augmented version of AIRL we implemented for the purposes of comparison that has state-action reward, denoted as AIRL(s,a). In reward learning, we only compare our method against AIRL(s) and AIRL(s,a) as GAIL does not recover rewards, and GAN-GCL is shown to exhibit inferior performance than AIRL [FLL17]. Furthermore, in the comparisons, we also include the expert performances which represents a policy learned by optimizing a ground-truth reward using TRPO [SLA$^+$15]. The performance of different methods are evaluated in term of mean and standard deviation of total rewards accumulated (denoted as score) by an agent during the trial, and for each experiment, we run five randomly-seeded trials.

Table 2.1: The evaluation of reward learning on transfer learning tasks. Mean scores (higher the better) with standard deviation are presented over 5 trials.

Algorithm	States-Only	Pointmass-Maze	Crippled-Ant
Expert	N/A	-4.98 ± 0.29	432.66 ± 14.38
AIRL	Yes	-8.07 ± 0.50	175.51 ± 27.31
AIRL	No	-19.28 ± 2.03	46.12 ± 14.37
EAIRL(Ours)	**No**	-7.01 ± 0.61	348.43 ± 43.17

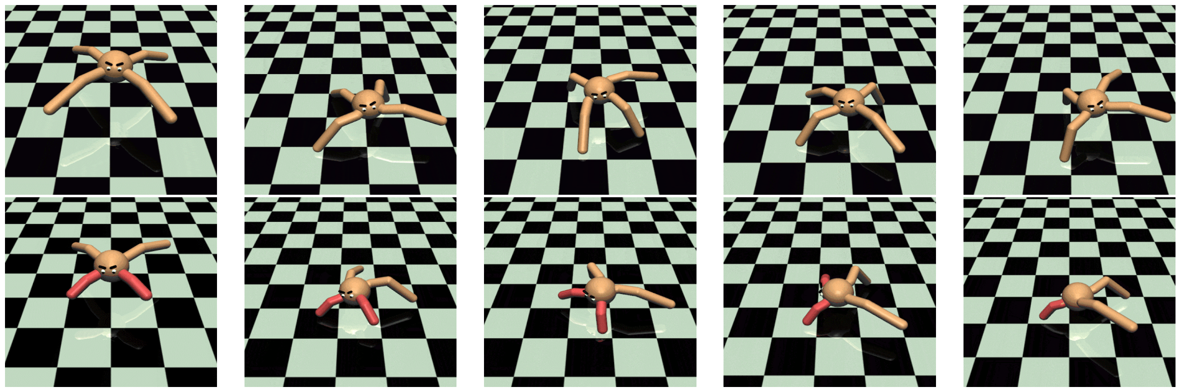


Figure 2.3: The top and bottom rows show the gait of standard and crippled ant, respectively.

Reward learning performance (Transfer learning experiments)

To evaluate the learned rewards, we consider a transfer learning problem in which the testing environments are made to be different from the training environments. More precisely, the rewards learned via IRL in the training environments are used to re-optimize a new policy in the testing environment using standard RL. We consider two test cases shown in the Fig. 2.1.

In the first test case, as shown in Fig. 2.1(a), we modify the agent itself during testing. We trained a reward function to make a standard quadruped ant to run forward. During testing, we disabled the front two legs (indicated in red) of the ant (crippled-ant), and the learned reward is used to re-optimize the policy to make a crippled-ant move forward. Note that the crippled-ant cannot move sideways. Therefore, the agent has to change the gait to run forward. In the second test case, shown in Fig 2.1(b), we change the environment structure. The agent learns to navigate a 2D point-mass to the goal region in a simple maze. We re-position the maze central-wall during

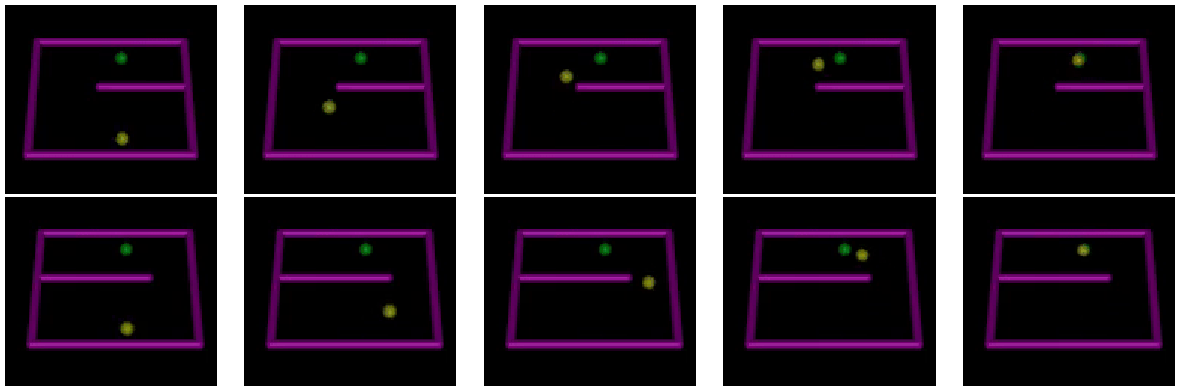


Figure 2.4: The top and bottom rows show the path followed by a 2D point-mass agent (yellow) to reach the target (green) in training and testing environment, respectively.

testing so that the agent has to take a different path, compared to the training environment, to reach the target (see Section 2.5).

Fig. 2.2 compares the policy performance scores over five different trials of EAIRL, AIRL(s) and AIRL(s,a) in the aforementioned transfer learning tasks. Fig .2.3 and Fig. 2.4 shows EAIRL execution traces in these scenarios. The expert score is shown as a horizontal line to indicate the standard set by an expert policy. Table 2.1 summarizes the means and standard deviations of the scores over five trials. It can be seen that our method recovers near-optimal reward functions as the policy scores almost reach the expert scores in all five trials even after transferring to unseen testing environments. Furthermore, our method performs significantly better than both AIRL(s) and AIRL(s,a) in matching an expert's performance, thus showing no downside to the EAIRL approach.

Policy learning performance (Imitation learning)

Next, we considered the performance of the learned policy specifically for an imitation learning problem in various control tasks. The tasks, shown in Fig. 2.5, include (i) making a 2D halfcheetah robot to run forward, (ii) making a 3D quadruped robot (ant) to move forward, (iii) making a 2D swimmer to swim, and (iv) keeping a friction less pendulum to stand vertically up. For each algorithm, we provided 20 expert demonstrations generated by a policy trained on

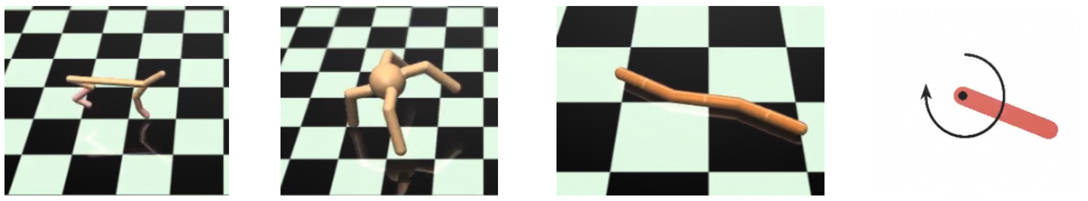

(a) HalfCheetah (b) Ant (c) Swimmer (d) Pendulum

Figure 2.5: Benchmark control tasks for imitation learning

a ground-truth reward using TRPO (Schulman et al., 2015). Table 2.2 presents the means and standard deviations of policy learning performance scores, over the five different trials. It can be seen that EAIRL, AIRL(s,a) and GAIL demonstrate similar performance and successfully learn to imitate the expert policy, whereas AIRL(s) and GAN-GCL fails to recover a policy.

Table 2.2: The evaluation of imitation learning on benchmark control tasks. Mean scores (higher the better) with standard deviation are presented over 5 trials for each method.

Methods	Environments			
	HalfCheetah	Ant	Swimmer	Pendulum
Expert	2139.83 ± 30.22	935.12 ± 10.94	76.21 ± 1.79	-100.11 ± 1.32
GAIL	1880.05 ± 15.72	738.72 ± 9.49	50.21 ± 0.26	-116.01 ± 5.45
GCL	-189.90 ± 44.42	16.74 ± 36.59	15.75 ± 7.32	-578.18 ± 72.84
AIRL(s,a)	1826.26 ± 19.64	645.90 ± 41.75	49.52 ± 0.48	-118.13 ± 11.33
AIRL(s)	121.10 ± 42.31	271.31 ± 9.35	33.21 ± 2.40	-134.82 ± 10.89
EAIRL	1870.10 ± 17.86	641.12 ± 25.92	49.55 ± 0.29	-116.26 ± 8.313

2.4 Discussion

This section highlights the importance of empowerment-regularized MaxEnt-IRL and modeling rewards as a function of both state and action rather than restricting to state-only formulation on learning rewards and policies from expert demonstrations.

In the scalable MaxEnt-IRL framework [FCAL16, FLL17], the normalization term is approximated by importance sampling where the importance-sampler/policy is trained to minimize the KL-divergence from the distribution over expert trajectories. However, merely minimizing the divergence between expert demonstrations and policy-generated samples leads to localized policy behavior which hinders learning generalized reward functions. In our proposed work, we regularize the policy update with empowerment i.e., we update our policy to reduce the divergence from expert data distribution as well as to maximize the empowerment (Eqn.2.12). The proposed regularization prevents premature convergence to local behavior which leads to robust state-action based rewards learning. Furthermore, empowerment quantifies the extent to which an agent can control/influence its environment in the given state. Thus the agent takes an action a on observing a state s such that it has maximum control/influence over the environment upon ending up in the future state s'.

Our experimentation also shows the importance of modeling discriminator/reward functions as a function of both state and action in reward and policy learning under GANs framework. The reward learning results show that state-only rewards (AIRL(s)) does not recover the action dependent terms of the ground-truth reward function that penalizes high torques. Therefore, the agent shows aggressive behavior and sometimes flips over after few steps (see the accompanying video), which is also the reason that crippled-ant trained with AIRL's disentangled reward function reaches only the half-way to expert scores as shown in Table 2.1. Therefore, the reward formulation as a function of both states and actions is crucial to learning action-dependent terms required in most real-world applications, including any autonomous driving, robot locomotion or manipulation task where large torque magnitudes are discouraged or are dangerous. The policy learning results further validate the importance of the state-action reward formulation. Table 2.2 shows that methods with state-action reward/discriminator formulation can successfully recover expert-like policies. Hence, our empirical results show that it is crucial to model reward/discriminator as a function of state-action as otherwise, adversarial imitation learning fails to learn ground-truth rewards and expert-like policies from expert data.

2.5 Derivations

For completeness, we present a derivation of presenting mutual information (MI) as variational lower bound and maximization of lower bound to learn empowerment.

Variational Information Lower Bound

As mentioned in section 2.1, the variational lower bound representation of MI is computed by defining MI as a difference in conditional entropies, and the derivation is formalized as follow.

$$\begin{aligned}
I^{w,q}(s) &= H(a|s) - H(a|s',s) \\
&= H(a|s) + \mathbb{E}_{p(s'|a,s)w(a|s)}[\log p(a|s',s)] \\
&= H(a|s) + \mathbb{E}_{p(s'|a,s)w(a|s)}[\log \frac{p(a|s',s)q(a|s',s)}{q(a|s',s)}] \\
&= H(a|s) + \mathbb{E}_{p(s'|a,s)w(a|s)}[\log q(a|s',s)] + \mathbb{E}_{p(s'|a,s)w(a|s)}[\log \frac{p(a|s',s)}{q(a|s',s)}] \\
&= H(a|s) + \mathbb{E}_{p(s'|a,s)w(a|s)}[\log q(a|s',s)] + \mathrm{KL}[p(a|s',s)||q(a|s',s)] \\
&\geq H(a|s) + \mathbb{E}_{p(s'|a,s)w(a|s)}[\log q(a|s',s)] \\
&\geq -\mathbb{E}_{w(a|s)} \log w(a|s) + \mathbb{E}_{p(s'|a,s)w(a|s)}[\log q(a|s',s)]
\end{aligned}$$

Variational Information Maximization

The empowerment is a maximal of MI and it can be formalized as follow by exploiting the variational lower bound formulation (for details see [MR15]).

$$\Phi(s) = \max_{w,q} \mathbb{E}_{p(s'|a,s)w(a|s)}[-\frac{1}{\beta}\log w(a|s) + \log q(a|s',s)] \tag{2.13}$$

As mentioned in section 2.1, given a training trajectories, the maximization of Eqn. 2.13 w.r.t inverse model $q(a|s',s)$ is a supervised maximum log-likelihood problem. The maximization of Eqn. 2.13 w.r.t $w(a|s)$ is derived through a functional derivative $\partial I^{w,q}/\partial w = 0$ under the constraint $\sum_a w(a|s) = 1$. For simplicity, we consider discrete state and action spaces, and the derivation is as follow:

$$\begin{aligned}\hat{I}^w(s) &= \mathbb{E}_{p(s'|a,s)w(a|s)}[-\frac{1}{\beta}\log w(a|s) + \log q(a|s',s)] + \lambda\big(\sum_a w(a|s) - 1\big)\\ &= \sum_a\sum_{s'} p(s'|a,s)w(a|s)\{-\frac{1}{\beta}\log w(a|s) + \log q(a|s',s)\} + \lambda\big(\sum_a w(a|s) - 1\big)\end{aligned}$$

$$\begin{aligned}&\frac{\partial \hat{I}^w(s)}{\partial w} = \sum_a \{(\lambda - \beta) - \log w(a|s) + \beta\mathbb{E}_{p(s'|a,s)}[\log q(a|s',s)]\} = 0\\ &w(a|s) = e^{\lambda-\beta}e^{\beta\mathbb{E}_{p(s'|a,s)}[\log q(a|s',s)]}\end{aligned}$$

By using the constraint $\sum_a w(a|s) = 1$, it can be shown that the optimal solution $w^*(a|s) = \frac{1}{Z(s)}\exp(u(s,a))$, where $u(s,a) = \beta\mathbb{E}_{p(s'|a,s)}[\log q(a|s',s)]$ and $Z(s) = \sum_a u(s,a)$. This solution maximizes the lower bound since $\partial^2 I^w(s)/\partial w^2 = -\sum_a \frac{1}{w(a|s)} < 0$.

Empowerment-regularized MaxEnt-IRL Formulation.

In this section we derive the Empowerment-regularized formulation of maximum entropy IRL. Let τ be a trajectory sampled from expert demonstrations D and

$$p_\xi(\tau) \propto p(s_0)\Pi_{t=0}^{T-1} p(s_{t+1}|s_t,a_t)\exp^{r_\xi(s_t,a_t)} \tag{2.14}$$

be a distribution over τ. As mentioned in Section 2.1, the IRL objective is to maximize the likelihood:

$$\max_{\xi} J(\xi) = \max_{\xi} \mathbb{E}_D[\log p_{\xi}(\tau)]$$

Furthermore, as derived in [FLL17], the gradient of above equation w.r.t ξ can be written as:

$$\begin{aligned}\max_{\xi} J(\xi) &= \mathbb{E}_D[\sum_{t=0}^{T} \frac{\partial}{\partial \xi} r_{\xi}(s_t, a_t)] - \mathbb{E}_{p_{\xi}}[\sum_{t=0}^{T} \frac{\partial}{\partial \xi} r_{\xi}(s_t, a_t)] \\ &= \sum_{t=0}^{T} \mathbb{E}_D[\frac{\partial}{\partial \xi} r_{\xi}(s_t, a_t)] - \mathbb{E}_{p_{\xi,t}}[\frac{\partial}{\partial \xi} r_{\xi}(s_t, a_t)]\end{aligned}$$

where $r_{\xi}(\cdot)$ is a parametrized reward to be learned, and $p_{\xi,t} = \int_{s_{t'} \neq t, a_{t'} \neq t} p_{\xi}(\tau)$ denotes marginalization of state-action at time t. Since, it is unfeasible to draw samples from p_{ξ}, [FCAL16] proposed to train an importance sampling distribution $\mu(\tau)$ whose varience is reduced by defining $\mu(\tau)$ as a mixture of polices, i.e., $\mu(a|s) = \frac{1}{2}(\pi(a|s) + \hat{p}(a|s))$, where $\hat{p}$ is a rough density estimate over demonstrations. Thus the above gradient becomes:

$$\frac{\partial}{\partial \xi} J(\xi) = \sum_{t=0}^{T} \mathbb{E}_D[\frac{\partial}{\partial \xi} r_{\xi}(s_t, a_t)] - \mathbb{E}_{\mu_t}[\frac{p_{\xi,t}(s_t, a_t)}{\mu_t(s_t, a_t)} \frac{\partial}{\partial \xi} r_{\xi}(s_t, a_t)] \tag{2.15}$$

We train our importance-sampler/policy π to maximize the empowerment $\Phi(\cdot)$ for generalization and to reduce divergence from true distribution by minimizing $D_{\mathrm{KL}}(\pi(\tau) \| p_{\xi}(\tau))$. Since, $\pi(\tau) = p(s_0)\Pi_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(s_t, a_t)$, the matching terms of $\pi(\tau)$ and $p_{\xi}(\tau)$ cancel out, resulting into entropy-regularized policy update. Furthermore, as we also include the empowerment $\Phi(\cdot)$ in the policy update to be maximized, hence the overall objective becomes:

$$\max_{\pi} \mathbb{E}_{\pi}[\sum_{t=0}^{T-1} r_{\xi}(s_t, a_t) + \Phi(s_{t+1}) - \log \pi(a_t|s_t)] \tag{2.16}$$

Our discriminator is trained to minimize cross entropy loss as mention in Eqn. 2.10, and for the proposed structure of our discriminator Eqn. 2.9, it can be shown that the discriminator's gradient w.r.t its parameters turns out to be equal to Equation 2.15 (for more details, see [FLL17]). On the other hand, our policy training objective is

$$r_{\pi}(s,a,s') = \log(D(s,a,s')) - \log(1 - D(s,a,s')) - l_I(s,a,s') \tag{2.17}$$

In the next section, we show that the above policy training objective is equivalent to Equation 2.16.

Policy Objective

We train our policy to maximize the discriminative reward $\hat{r}(s,a,s') = \log(D(s,a,s') - \log(1 - D(s,a,s')))$ and minimize the information-theoretic loss function $l_I(s,a,s')$. The discriminative reward $\hat{r}(s,a,s')$ simplifies to:

$$\begin{aligned}
\hat{r}(s,a,s') &= \log(D(s,a,s')) - \log(1 - D(s,a,s')) \\
&= \log \frac{e^{f(s,a,s')}}{e^{f(s,a,s')} + \pi(a|s)} - \log \frac{\pi(a|s)}{e^{f(s,a,s')} + \pi(a|s)} \\
&= f(s,a,s') - \log \pi(a|s)
\end{aligned}$$

where $f(s,a,s') = r(s,a) + \gamma\Phi(s') - \Phi(s)$. The entropy-regularization is usually scaled by the hyperparameter, let say $\lambda_h \in \mathbb{R}$, thus $\hat{r}(s,a,s') = f(s,a,s') - \lambda_h \log \pi(a|s)$. Hence, assuming single-sample (s,a,s'), absolute-error for $l_I(s,a,s') = |\log q_{\phi}(a|s,s') - (\log \pi(a|s) + \Phi(s))|$, and

$l_i > 0$, the policy is trained to maximize following:

$$\begin{aligned} r_\pi(s,a,s') &= f(s,a,s') - \lambda_h \log \pi(a|s) - l_I(s,a,s') \\ &= r(s,a) + \gamma\Phi(s') - \Phi(s) - \lambda_h \log \pi(a|s) - \log q(a|s,s') + \log \pi(a|s) + \Phi(s) \\ &= r(s,a) + \gamma\Phi(s') - \lambda_h \log \pi(a|s) - \log q(a|s,s') + \log \pi(a|s) \end{aligned}$$

Note that, the potential function $\Phi(s)$ cancels out and we scale the leftover terms of l_I with a hyperparameter λ_I. Hence, the above equation becomes:

$$r_\pi(s,a,s') = r(s,a,s') + \gamma\Phi(s') + (\lambda_I - \lambda_h) \log \pi(a|s) - \lambda_I \log q(a|s,s')$$

We combine the log terms together as:

$$r_\pi(s,a,s') = r(s,a) + \lambda_I \Phi(s') + \lambda \hat{H}(\cdot) \tag{2.18}$$

where λ is a hyperparameter, and $\hat{H}(\cdot)$ is an entropy regularization term depending on $q(a|s,s')$ and $\pi(a|s)$. Therefore, it can be seen that the Eqn. 2.18 is equivalent/approximation to Eqn. 2.16.

2.6 Implementation Details

Network Architectures

We use two-layer ReLU network with 32 units in each layer for the potential function $h_\varphi(\cdot)$ and $\Phi_\varphi(\cdot)$, reward function $r_\xi(\cdot)$, discriminators of GAIL and GAN-GCL. Furthermore, policy $\pi_\theta(\cdot)$ of all presented models and the inverse model $q_\phi(\cdot)$ of EAIRL are presented by two-layer RELU network with 32 units in each layer, where the network's output parametrizes the Gaussian distribution, i.e., we assume a Gaussian policy.

Hyperparameters

For all experiments, we use the temperature term $\beta = 1$. We evaluated both mean-squared and absolute error forms of $l_I(s,a,s')$ and found that both lead to similar performance in reward and policy learning. We set entropy regularization weight to 0.1 and 0.001 for reward and policy learning, respectively. The hyperparameter λ_I was set to 1.0 for reward learning and 0.001 for policy learning. The target parameters of the empowerment-based potential function $\Phi_{\varphi'}(\cdot)$ were updated every 5 and 2 epochs during reward and policy learning respectively. Although reward learning hyperparameters are also applicable to policy learning, we decrease the magnitude of entropy and information regularizers during policy learning to speed up the policy convergence to optimal values. Furthermore, we set the batch size to 2000- and 20000-steps per TRPO update for the pendulum and remaining environments, respectively. For the methods [FLL17, HE16] presented for comparison, we use their suggested hyperparameters. We also use policy samples from previous 20 iterations as negative data to train the discriminator of all IRL methods presented in this work to prevent the parametrized reward functions from overfitting the current policy samples.

2.7 Acknowledgements

Chapter 2, in part, is a reprint of **A.H.Qureshi**, B. Boots, and M.C.Yip,"Adversarial Imitation Via Variational Inverse Reinforcement Learning", *International Conference on Representation Learning (ICLR)*, 2019. The dissertation author is the primary author of this paper.

Chapter 3

Neural Motion Planning

In this work, we discuss a new wave in Motion Planning called Neural Motion Planners that have emerged from the cross-fertilization of motion planning and deep learning to solve planning problems under various collision avoidance, kinodynamic, and manifold kinematic constraints.

3.1 Collision Avoidance Constraints

This chapter introduces our initial work towards neural motion planners under collision avoidance constraints [QSBY19, QY18, QMSY20] . We highlight that merging both machine

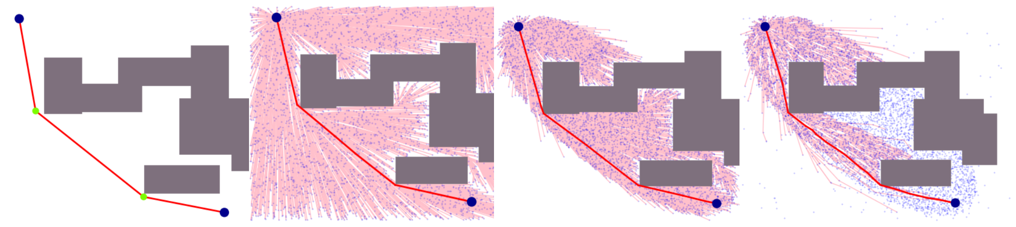

(a) MPNet (b) RRT* (c) Informed-RRT* (d) BIT*

Figure 3.1: MPNet greedily outputs a near-optimal path, whereas classical planning methods such as RRT* [KF11], Informed-RRT* [GSB14], and BIT* [GSB15] need to expand their planning spaces through an exhaustive search.

learning and classical motion planning techniques holds great potential to build motion planning methods with all key features of an ideal planner ranging from theoretical guarantees to computational efficiency. In this respect, this chapter formally presents our Motion Planning Networks, or MPNet, and its features corresponding to an ideal planner and its merits in solving complex robotic motion planning problems. MPNet is a deep neural network-based bidirectional iterative planning algorithm that comprises two modules, an encoder network and planning network. The encoder network takes the environment information, such as the raw or voxelized output of a depth camera or LIDAR, and embeds them into a latent space. The planning network takes the environment encoding, robot's current and goal state, and outputs a next state of the robot that would lead it closer to the goal region. MPNet can very effectively generate steps from start to goal that are likely to be part of the optimal solution with minimal-to-no branching required (Fug. 3.1). Being a neural network approach, we also propose three learning strategies to train MPNet: i) offline batch learning which assumes the availability of all training data, ii) continual learning with episodic memory which assumes that the expert demonstrations come in streams and the global training data distribution is unknown, and iii) active continual learning that incorporates MPNet into the learning process and asks for expert demonstrations only when needed. The following are the major contributions of MPNet:

- MPNet can learn from streaming data which is crucial for real-world scenarios in which the expert demonstrations usually come in streams, such as in semi self-driving cars. However, as MPNet uses deep neural networks, it can suffer from catastrophic forgetting when given the data in streams. To retain MPNet prior knowledge, we use a continual learning approach based on episodic memory and constraint optimization.
- The active continual learning approach that asks for demonstrations only when needed, hence improving the overall training data efficiency. This strategy is in response to practical and data-efficient learning where planning problems come in streams, and MPNet attempts to plan a motion for them. In case MPNet fails to find a path for a given problem, only then

an Oracle Planner is called to provide an expert demonstration for learning.

- MPNet plans paths with a low computational complexity and exhibits a mean computation time of less than 1 second in all presented experiments.
- MPNet can generate informed samples, thanks to its stochastic planning network, for sampling-based motion planners such as RRT* without incurring any additional computational load. The MPNet informed sampling based RRT* exhibits mean computation time of less than a second while ensuring asymptotic optimality and completeness guarantees.
- A hybrid planning approach that combines MPNet with classical planners to provide worst-case guarantees of our approach. MPNet plans motion through divide-and-conquer since it first outputs a set of critical states and recursively finds paths between them. Therefore, it is straightforward to outsource a segment of a planning problem to a classical planner, if needed, while retaining the computational benefits of MPNet.
- MPNet generalizes to similar but unseen environments that were not in the training examples.

3.1.1 Related work

The quest for solving the motion planning problem originated with the development of complete algorithms which suffer from computational inefficiency, and thus led to more efficient methods with resolution- and probabilistic- completeness [LaV06]. The algorithms with full completeness guarantees [SS83] [LPW79] find a path solution, if one exists, in a finite-time. However, these methods are computationally intractable for most practical applications as they require the complete environment information such as obstacle geometry which is not usually available in real-world scenarios [Can88]. The algorithms with resolution-completeness [Kha86] [BLP85] also find a path, if one exists, in a finite-time, but require tedious fine-tuning of resolution parameters for every given planning problem. To overcome the limitations of complete and resolution-complete algorithms, the probabilistically complete methods, also known as Sampling-based Motion Planners (SMPs), were introduced [LaV06]. These methods rely on sampling techniques to generate rapidly-exploring trees or roadmaps in robot's obstacle-free state-space. The feasible path is determined by querying the generated graph through shortest path-finding methods such as Dijkstra's method (Fig. 3.2). These methods are probabilistically complete, since the probability of finding a path solution, if one exists, approaches one as the number of samples in the graph approaches infinity [LaV06].

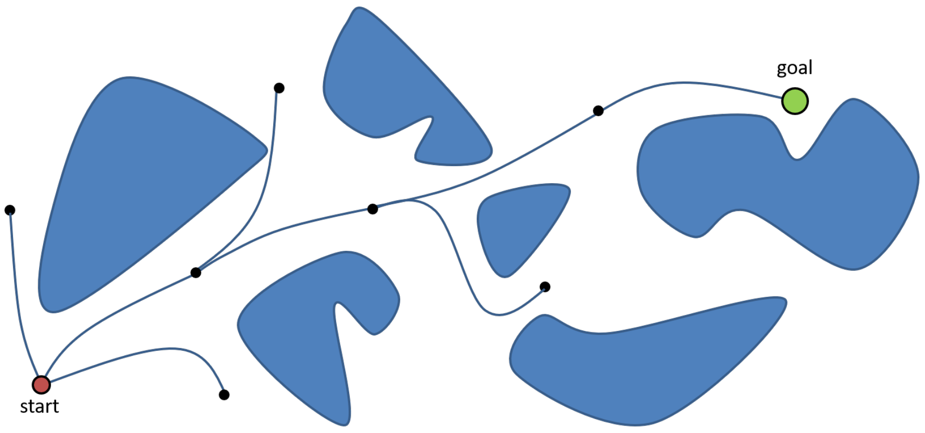


Figure 3.2: RRT [LaV98] algorithm randomly sampling obstacle-free space to generate a tree, and find a path connecting the given start and goal configurations.

The prominent and widely used SMPs include single-query rapidly-exploring random trees (RRT) [LaV98] and multi-query probabilistic roadmaps (PRM) [KL98]. In practice, single-query methods are preferred since most multi-query problems can be solved as a series of single-query problems [LaV98] [KF11]. Besides, PRM based methods require pre-computation of a roadmap which is usually expensive to determine in online planning problems [KF11]. Therefore, RRT and its variants have now emerged as a promising tools for motion planning that finds a path irrespective of obstacles geometry. Although the RRT algorithm rapidly finds a path solution, if one exists, they fail to find the shortest path solution [KF11]. An optimal variant of RRT called RRT* asymptotically guarantees to find the shortest path, if one exists [KF11]. However, RRT* becomes computationally inefficient as the dimensionality of the planning problem increases. Furthermore, studies show that to determine a ε-near optimal path in $d \in \mathbb{N}$ dimensions, nearly $O(1/\varepsilon^d)$ samples are required. Thus, RRT* is no better than grid search methods in higher dimensional spaces [Hau15]. Several methods have been proposed to mitigate limitations in current asymptotically optimal SMPs through different heuristics such as biased sampling [QA16] [QMI+13] [GSB14] [GSB15], lazy edge evaluation [HMP+18], and bidirectional tree generations [QA15] [TQAN18].

Biased sampling heuristics adaptively sample the robot state-space to overcome limitations caused by random uniform exploration in underlying SMP methods. For instance, P-RRT* [QA16] [QMI+13] incorporates artificial potential fields [Kha86] into RRT* to generate goal directed trees for rapid convergence to an optimal path solution. In similar vein, Gammell et al. proposed the Informed-RRT* [GSB14] and BIT* (Batch Informed Trees) [GSB15]. Informed-RRT* defines an ellipsoidal region using RRT*'s initial path solution to adaptively sample the configuration space for optimal path planning. Despite improvements in computation time, Informed-RRT* suffers in situations where finding an initial path is itself challenging. On the other hand, BIT* is an incremental graph search method that instantiates a dynamically-changing ellipsoidal region for batch sampling to compute paths. Despite some improvements in computation speed, these biased sampling heuristics still suffer from the curse of dimensionality.

Lazy edge evaluation methods, on the other hand, have shown to exhibit significant improvements in computation speeds by evaluating edges only along the potential path solutions. However, these methods are critically dependent on the underlying edge selector and tend to exhibit limited performance in cluttered environments [HS]. Bidirectional path generation improves the algorithm performance in narrow passages but still inherits the limitations of baseline SMPs [QA15][TQAN18].

Reinforcement learning (RL) [SB+98] has also emerged as a prominent tool to solve continuous control and planning problems [LHP+15]. RL considers Markov Decision Processes

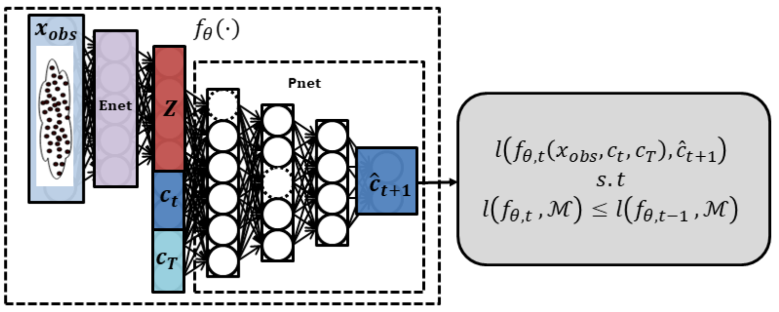
(MDPs) where an agent interacts with the environment by observing a state and taking an action which leads to a next state and reward. The reward signal encapsulates the underlying task and provides feedback to the agent on how well it is performing. Therefore, the objective of an agent is to learn a policy to maximize the overall expected reward function. In the past, RL was only employed for simple problems with low-dimensions [DNP$^+$13] [KBP13]. Recent advancements have led to solving harder, higher dimensional problems by merging the traditional RL with expressive function approximators such neural networks, now known as Deep RL (DRL) [MKS$^+$15] [DCH$^+$16]. DRL has solved various challenging robotic tasks using both model-based [LFDA16] [LPK$^+$18] [YLK$^+$17] and model-free [CKY$^+$17] [GHLL17] [PHL$^+$17] approaches. Despite considerable progress, solving practical problems which have weak rewards and long-horizons remain challenging [QJQ$^+$20].

There also exist approaches that apply various learning strategies such as imitation learning to mitigate the limitations of motion planning methods. Zucker et al. [ZKB08] proposed to adapt the sampling for the SMPs using REINFORCE algorithm [Wil92] in discretized workspaces. Berenson et al. [BAG12] use a learned heuristic to store new paths, if needed, or to recall and repair the existing paths. Coleman et al. [CŞM$^+$15] store experiences in a graph rather than individual trajectories. Ye and Alterovitz [YA17] use human demonstrations in conjunction with SMPs for path planning. While improved performance was noted compared to traditional planning algorithms, these methods lack generalizability and require tedious hand-engineering for every new environment. Therefore, modern techniques use efficient function approximators such as neural networks to either embed a motion planner or to learning auxiliary functions for SMPs such as sampling heuristic to speed up planning in complex cluttered environments.

Neural Motion Planning has been a recent addition to the motion planning literature. Bency et al. [BQY19] introduced recurrent neural networks to embed a planner based on its demonstrations. While useful for learning to navigate static environments, their method does not use environment information and therefore, is not meant to generalize to other environments.

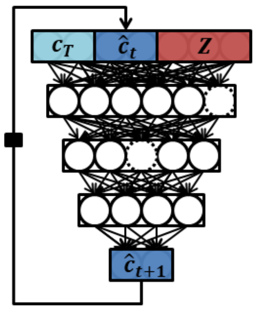
Ichter et al. [IHP18] proposed conditional variational autoencoders that contextualize on environment information to generate samples through decoder network for the SMPs such as FMT* [JP16]. The SMPs use the generated samples to create a graph for finding a feasible or optimal path for the robot to follow. In a similar vein, Zhang et. al [ZHL18] learns a rejection sampling policy that rejects or accepts the given uniform samples before making them the part of the SMP graph. The rejection sampling policy is learned using past experiences from the similar environments. Note that a policy for rejection sampling implicitly learns the sampling distribution whereas Icheter et. al [IHP18] explicitly generates the guided samples for the SMPs. Bhardwaj et al [BCS17] proposed a method called SAIL that learns a deterministic policy which guides the graph expansion of underlying graph-based planner towards the promising areas that potentially contains the path solution. SAIL learns the guiding policy using the oracle Q-values (encapsulates the cost-to-go function), argmin regression, and full environment information. Like these adaptive sampling methods, MPNet can also generate informed samples for SMPs, but in addition, our approach is also outputs feasible trajectories with worst-case theoretical guarantees.

There has also been attempts towards building learning-based motion planners. For instance, Value Iteration Networks (VIN) [TWT$^+$16] approximates a planner by emulating value iteration using recurrent convolutional neural networks and max-pooling. However, VIN is only applicable for discrete planning tasks. Universal Planning Networks (UPN) [SJA$^+$18] extends VIN to continuous control problems using gradient descent over generated actions to find a motion plan connecting the given start and goal observations. However, these methods do not generalize to novel environments or tasks and thus require frequent retraining. The most relevant approach to our neural planner (MPNet) is L2RRT [IP19] that plans motion in learned latent spaces using RRT method. L2RRT learns state-space encoding model, agent's dynamics model, and collision checking model. However, it is unclear that existence of a path solution in configuration space will always imply the existence of a path in the learned latent space and vice versa.



x_{obs}
Enet
Z
c_t
c_T
$f_\theta(\cdot)$
Pnet
$\hat{c}_{t+1}$
$l(f_{\theta,t}(x_{obs}, c_t, c_T), \hat{c}_{t+1})$
$s.t$
$l(f_{\theta,t}, \mathcal{M}) \leq l(f_{\theta,t-1}, \mathcal{M})$

(a)



(b)

Figure 3.3: MPNet consists of encoder network (Enet) and planning network (Pnet). (a) shows their end-to-end training under a continual learning setting. Fig (b) shows the online execution of MPNet.

3.1.2 Motion Planning Networks (MPNet)

MPNet comprises of two neural networks: an encoder network (Enet) and a planning network (Pnet). Enet takes the robot's surrounding information such as a raw point-cloud or point-cloud converted to voxel depending on the underlying neural architecture that could be a fully-connected neural network or a 3D convolutional neural network (CNN), respectively. The output of the Enet is a latent space embedding of the given information. Pnet takes the encoding of the environment, the robot's current state and goal state to output samples for either a path or tree generation. In remaining section, we describe the notations necessary to outline MPNet.

Let robot configuration space (C-space) be denoted as $\mathcal{C} \subset \mathbb{R}^d$ comprising of obstacle space $\mathcal{C}_{\text{obs}}$ and obstacle-free space $\mathcal{C}_{\text{free}} = \mathcal{C} \backslash \mathcal{C}_{\text{obs}}$, where d is the C-space dimensionality. Let robot's surrounding environment, also known as workspace, be denoted as $\mathcal{X} \subset \mathbb{R}^m$, where m is a workspace dimension. Like C-space, the workspace also comprise of obstacle, $\mathcal{X}_{\text{obs}}$, and obstacle-free, $\mathcal{X}_{\text{free}} = \mathcal{X} \backslash \mathcal{X}_{\text{obs}}$, regions. The workspaces could be up to 3-dimensions whereas the C-space can have higher dimensions depending on the robot's degree-of-freedom (DOF). Let robot initial and goal configuration space be $c_{\text{init}} \in \mathcal{C}_{\text{free}}$ and $c_{\text{goal}} \subset \mathcal{C}_{\text{free}}$, respectively. Let $\sigma = \{c_0, \cdots, c_T\}$ be an ordered list of length T. We assume σ_i corresponds to the i-th state in σ, where $i = [0, T]$. For instance σ_0 corresponds to state c_0. Furthermore, we consider σ_{end} corresponds to the last element of σ, i.e., $\sigma_{\text{end}} = c_T$. A motion planning problem can be concisely denoted as $\{c_{\text{init}}, c_{\text{goal}}, \mathcal{C}_{\text{obs}}\}$ where the aim of a planning algorithm is to find a feasible path solution, if one exists, that connects c_{init} to c_{goal} while completely avoiding obstacles in $\mathcal{C}_{\text{obs}}$. Therefore, a feasible path can be represented as an ordered list $\sigma = \{c_0, \cdots, c_T\}$ such that $c_0 = c_{\text{init}}$, $c_T = c_{\text{goal}}$, and a path constituted by connecting consecutive states in σ lies entirely in $\mathcal{C}_{\text{free}}$. In practice, C-space representation of obstacles $\mathcal{C}_{\text{obs}}$ is unknown and rather a collision-checker is available that takes workspace information, $\mathcal{X}_{\text{obs}}$, and robot configuration, c, and determines if they are in collision or not. Another important problem in motion planning is to find an optimal path solution for a given cost function. Let a cost function be defined as $J(\cdot)$. An optimal planner provides guarantees, either weak or strong, that if given enough running time, it would eventually converge to an optimal path, if one exists. The optimal path is a solution that has the lowest possible cost w.r.t. $J(\cdot)$.

We consider a practical scenario, where MPNet plans feasible, near-optimal paths using raw point-cloud/voxel data of obstacles $x_{\text{obs}} \subset \mathcal{X}_{\text{obs}}$. However, like other planning algorithms, we do assume an availability of a collision-checker that verifies the feasibility of MPNet generated paths based on $\mathcal{X}_{\text{obs}}$. Precisely, Enet, with parameters θ^e, takes the environment information x_{obs} and compresses them into a latent space Z,

$$Z \leftarrow \text{Enet}(x_{\text{obs}}; \theta^e) \tag{3.1}$$

Pnet, with parameters θ^p, takes the environment encoding Z, robot's current or initial configuration $c_t \in \mathcal{C}_{\text{free}}$, and goal configuration $c_{\text{goal}} \subset \mathcal{C}_{\text{free}}$ to produce a trajectory through incremental generation of states $\hat{c}_{t+1}$ (Fig. 3.3 (b)),

$$\hat{c}_{t+1} \leftarrow \text{Pnet}(Z, c_t, c_{\text{goal}}; \theta^p) \tag{3.2}$$

3.1.3 MPNet: Training

In this section, we present three training methodologies for MPNet neural models, Enet and Pnet: i) offline batch learning, ii) continual learning, and iii) active continual learning.

Offline batch learning method assumes the availability of complete data to train MPNet offline before running it online to plan motions for unknown/new planning problems. Continual learning enables MPNet to learn from streaming data without forgetting past experiences. Active continual learning incorporates MPNet into the continual learning process where MPNet actively asks for an expert demonstration when needed for the given problem. Further details on training approaches are presented as follow.

Offline batch learning

The offline batch learning requires all the training data to be available in advance for MPNet [QSBY19] [QY18]. As mentioned earlier, MPNet comprises of two modules, Enet and Pnet. In this training approach, both modules can either be trained together in an end-to-end fashion using planning network loss function or separately with their individual loss functions.

To train Enet and Pnet together, we back-propagate the gradient of Pnet's loss function in Equation 3.4 through both modules. For the standalone training of Enet, we use an encoder-decoder architecture whenever there is an ample amount of environmental point-cloud data available for unsupervised learning. There exist several techniques for encoder-decoder training such as variational auto-encoders [KW13] and their variants [HMP$^+$17] or the class of contractive autoencoders (CAE) [RVM$^+$11]. For our purpose, we observed that the contractive autoencoders learn robust feature spaces desired for planning and control, and give better performance than other available encoding techniques. The CAE uses the usual reconstruction loss and a regularization over encoder parameters,

$$l_{\mathrm{AE}}(\theta^e, \theta^d) = \frac{1}{N_{\mathrm{obs}}} \sum_{x \in D_{\mathrm{obs}}} ||x - \hat{x}||^2 + \lambda \sum_{ij} (\theta^e_{ij})^2 \tag{3.3}$$

where θ^e, θ^d are the encoder and decoder parameters, respectively, and λ denotes regularization coefficient. The variable $\hat{x}$ represents reconstructed point-cloud. The training dataset of obstacles' point-cloud $x \subset \mathcal{X}_{\mathrm{obs}}$ is denoted as D_{obs} which contains point cloud from $N_{\mathrm{obs}} \in \mathbb{N}$ different workspaces.

The planning module (Pnet) is a stochastic feed-forward deep neural network with parameters θ^p. Our demonstration trajectory is a list of waypoints, $\sigma = \{c_0, c_1, \cdots, c_T\}$, connecting the start and goal configurations such that the fully connected path lies in obstacle-free space. To train Pnet either end-to-end with Enet or separately for the given expert trajectory, we consider one-step look ahead prediction strategy. Therefore, MPNet takes the obstacles' point-cloud embedding Z, robot's current state c_t and goal state c_T as an input to output the next waypoint $\hat{c}_{t+1}$ towards the goal-region. The training objective is a mean-squared-error (MSE) loss between the predicted $\hat{c}_{t+1}$ and target waypoints c_{t+1}, i.e.,

$$l_{\mathrm{Pnet}}(\theta) = \frac{1}{N_p} \sum_{j}^{\hat{N}} \sum_{i=0}^{T_j - 1} ||\hat{c}_{j,i+1} - c_{j,i+1}||^2, \tag{3.4}$$

where T_j is the length of j-th trajectory, $\hat{N} \in \mathbb{N}$ is the total number of training trajectories, and N_p is the averaging term. Although we use MSE loss, one can consider other choices such as adversarial loss function [GPAM$^+$14]. In rigid body planning, such as planning in SE(3), if the rotations are represented as quaternions, we use the following loss:

$$l_{\mathrm{Pnet}}(\theta) = l_p + \beta l_q, \tag{3.5}$$

where l_p and l_q correspond to the MSE loss between positional p and quaternion q components of the waypoints in σ, and β is a scaling factor. Since quaternion needs to be in a unit sphere the loss l_q is defined as (for more details refer to [KC17]):

$$l_{\text{q}} = \frac{1}{N_p} \sum_{j}^{\hat{N}} \sum_{i=0}^{T-1} \left\| \frac{\hat{q}_{j,i+1}}{||\hat{q}_{j,i+1}||} - q_{j,i+1} \right\|^2 \tag{3.6}$$

Continual Learning

In continual learning settings, both modules of MPNet (Enet and Pnet) are trained in an end-to-end fashion, since both neural models need to adapt to the incoming data (Fig. 3.3(a)). We consider a supervised learning problem. Therefore, the data comes with targets, i.e.,

$$(s_1, y_1, \cdots, s_i, y_i, \cdots, s_N, y_N)$$

where $s = (c_t, c_T, x_{\text{obs}})$ is the input to MPNet comprising of the robot's current state c_t, the goal state c_T, and obstacles information x_{obs}. The target y is the next state c_{t+1} in the expert trajectory given by an oracle planner. Generally, continual learning using neural networks suffers from the issue of catastrophic forgetting since taking a gradient step on a new datum could erase the previous learning. The problem of catastrophic forgetting is also described in terms of knowledge transfer in the backward direction.

To describe backward transfer, we introduce few new notations as follow. Let a mean model success rate on a dataset $\mathcal{M}$ after learning from an expert example at time $t-1$ and t be denoted as $\mathcal{A}_{\mathcal{M},t-1}$ and $\mathcal{A}_{\mathcal{M},t}$, respectively. A metric described as a backward transfer can be written as $\mathcal{B} = \mathrm{A}_{\mathcal{M},t} - \mathrm{A}_{\mathcal{M},t-1}$. A positive backward transfer $\mathcal{B} \geq 0$ indicates that after learning from a new experience, the learning-based planner performed better on the previous tasks represented as $\mathcal{M}$. A negative backward transfer $\mathcal{B} < 0$ indicates that on learning from new experience the model on previous tasks deteriorated.

To overcome negative backward transfer leading to catastrophic forgetting, we employ the

Gradient Episodic Memory (GEM) method for lifelong learning [LPR17]. GEM uses the episodic memory $\mathcal{M}$ that has a finite set of continuum data seen in the past to ensure that the model update doesn't lead to negative backward transfer while allowing only the positive backward transfer. For MPNet, we adapt GEM for the regression problem using the following optimization objective function.

$$\begin{aligned} &\min_{\theta} l(f_{\theta}^{t}(s), y) \text{ s.t} \\ &\qquad \hat{\mathbb{E}}_{(s,y)\sim\mathcal{M}}[l(f_{\theta}^{t}(s), y)] \leq \hat{\mathbb{E}}_{(s,y)\sim\mathcal{M}}[l(f_{\theta}^{t-1}(s), y)] \end{aligned} \tag{3.7}$$

where $l = \|f_{\theta}^{t}(s) - y\|^2$ is a squared-error loss, f_{θ}^{t} is the MPNet model at time step t (see Fig. 3.3). Furthermore, note that, if the angle between proposed gradient (g) at time t, and the gradient over $\mathcal{M}$ $(g_{\mathcal{M}})$ is positive, i.e., $\langle g, g_{\mathcal{M}} \rangle \geq 0$, there is no need to maintain the old function parameters f_{θ}^{t-1} because the above equation can be formulated as:

$$\langle g, g_{\mathcal{M}} \rangle := \left\langle \nabla_{\theta} l\big(f_{\theta}(s), y\big), \hat{\mathbb{E}}_{(s,y)\sim\mathcal{M}} \nabla_{\theta} l\big(f_{\theta}(s), y\big) \right\rangle \tag{3.8}$$

where $\hat{\mathbb{E}}$ denotes empirical mean.

In most cases, the proposed gradient g violates the constraint $\langle g, g_{\mathcal{M}} \rangle \geq 0$, i.e., the proposed gradient update g will cause increase in the loss over previous data. To avoid such violations, David and Razanto [LPR17] proposed to project the gradient g to the nearest gradient g' that keeps $\langle g', g_{\mathcal{M}} \rangle \geq 0$, i.e,

$$\min_{g'} \frac{1}{2}\|g - g'\|_2^2 \text{ s.t } \langle g', g_{\mathcal{M}} \rangle \geq 0 \tag{3.9}$$

The projection of proposed gradient g to g' can be solved efficiently using Quadratic

Algorithm 2: Continual Learning

```
Initialize memories: episodic $\mathcal{M}$ and replay $\mathcal{B}^*$
Set the replay period $r$
Set the replay batch size $N_B$
Initialize MPNet $f_\theta$ with parameters $\theta$.
for $t = 0$ to $T$ do
    $\{\mathcal{X}_{\text{obs}}, c_{\text{init}}, c_{\text{goal}}\}_t \leftarrow$ GetPlanningProblem()
    $\sigma \leftarrow$ GetExpertDemo($\mathcal{X}_{\text{obs}}, c_{\text{init}}, c_{\text{goal}}$)
    $\mathcal{M} \leftarrow$ UpdateEpisodicMemory($\sigma, \mathcal{M}$)
    $\mathcal{B}^* \leftarrow \mathcal{B}^* \cup \sigma$
    $g \leftarrow \hat{\mathbb{E}}_{(s,y)\sim\sigma} \nabla_\theta l(f_\theta(s), y)$
    $g_{\mathcal{M}} \leftarrow \hat{\mathbb{E}}_{(s,y)\sim\mathcal{M}} \nabla_\theta l(f_\theta(s), y)$
    Project $g$ to $g'$ using QP based on Equation 3.9
    Update parameters $\theta$ w.r.t. $g'$
    if $\mathcal{B}^*$.size() $> N_B$ and not $t$ mod $r$ then
        $\mathcal{B} \leftarrow$ SampleReplayBatch($\mathcal{B}^*$)
        $g \leftarrow \hat{\mathbb{E}}_{(s,y)\sim\mathcal{B}} \nabla_\theta l(f_\theta(s), y)$
        $g_{\mathcal{M}} \leftarrow \hat{\mathbb{E}}_{(s,y)\sim\mathcal{M}} \nabla_\theta l(f_\theta(s), y)$
        Project $g$ to $g'$ using QP based on Equation 3.9
        Update parameters $\theta$ w.r.t. $g'$
```

Programming (QP) based on the duality principle, for details refer to [LPR17].

Various data parsing methods are available to update the episodic memory $\mathcal{M}$. These sample selection strategies for episodic memory play a vital role in the performance of continual/lifelong learning methods such as GEM [IC18]. There exist several selection metrics such as surprise, reward, coverage maximization, and global distribution matching [IC18]. In our continual learning framework, we use a global distribution matching method to select samples for the episodic memory. For details and comparison of different sample selection approaches, please refer to Section 3.17 (Sample Selection Strategies). The global distribution matching method, also known as reservoir sampling, uses random sampling techniques to populate the episodic memory. The aim is to approximately capture the global distribution of the training dataset since it is unknown in advance. There are several ways to implement reservoir sampling. The simplest approach accepts the new sample at i-th step with probability $\frac{|\mathcal{M}|}{i}$ to replace a randomly selected old sample from the memory. In other words, it rejects the new sample at i-the step with probability $1 - \frac{|\mathcal{M}|}{i}$ to keep the old samples in the memory.

In addition to episodic memory $\mathcal{M}$, we also maintain a replay/rehearsal memory $\mathcal{B}^*$. The replay buffer lets MPNet rehearse by learning again on the old samples (Line 14-19). We found that rehearsals further mitigate the problem of catastrophic forgetting and leads to better performance, as also reported by Rolnick et al. [RAS+19] in reinforcement learning setting. Note that replay or rehearsal on the past data is done with the interval of replay period $r \in \mathbb{N}_{\geq 0}$.

Finally, Algorithm 2 presents the continual learning framework where an expert provides a single demonstration at each time step, and MPNet model parameters are updated according to the projected gradient g'.

Active Continual Learning

Active continual learning (ACL) is our novel data-efficient learning strategy which is practical in the sense that the planning problem comes in streams and our method asks for expert demonstrations only when needed. It builds on the framework of continual learning presented in the previous section. ACL introduces a two-level of sample selection strategy. First, ACL gathers the training data by actively asking for the demonstrations on problems where MPNet failed to find a path. Second, it employs a sample selection strategy that further prunes the expert demonstrations to fill episodic memory so that it approximates the global distribution from streaming data. The two-level of data selection in ACL improves the training samples efficiency while learning the generalized neural models for the MPNet.

Algorithm 3 presents the outline of ACL method. At every time step t, the environment generates a planning problem $(c_{\text{init}}, c_{\text{goal}}, \mathcal{X}_{\text{obs}})$ comprising of the robot's initial c_{init} and goal c_{goal} configurations and the obstacles' information $\mathcal{X}_{\text{obs}}$ (Line 7). Before asking MPNet to plan a

Algorithm 3: Active Continual Learning

```
Initialize memories: episodic $\mathcal{M}$ and replay $\mathcal{B}^*$
Initialize MPNet $f_\theta$ with parameters $\theta$.
Set the number of iterations $C$ to pretrain MPNet.
Set the replay period $r$
Set the replay batch size $N_B$
for $t = 0$ to $T$ do
    $\{\mathcal{X}_{\text{obs}}, c_{\text{init}}, c_{\text{goal}}\}_t \leftarrow$ GetPlanningProblem()
    $\sigma \leftarrow \varphi$ \\ an empty list to store path solution
    if $t > N_c$ then
        $x_{\text{obs}} \subset \mathcal{X}_{\text{obs}}$
        $\sigma \leftarrow f_\theta(x_{\text{obs}}, c_{\text{init}}, c_{\text{goal}})$
    if not $\sigma$ then
        $\sigma \leftarrow$ GetExpertDemo$(\mathcal{X}_{\text{obs}}, c_{\text{init}}, c_{\text{goal}})$
        $\mathcal{B}^* \leftarrow \mathcal{B}^* \cup \sigma$
        $\mathcal{M} \leftarrow$ UpdateEpisodicMemory$(\sigma, \mathcal{M})$
        $g \leftarrow \hat{\mathbb{E}}_{(s,y)\sim\sigma} \nabla_\theta l\big(f_\theta(s), y\big)$
        $g_{\mathcal{M}} \leftarrow \hat{\mathbb{E}}_{(s,y)\sim\mathcal{M}} \nabla_\theta l\big(f_\theta(s), y\big)$
        Project $g$ to $g'$ using QP based on Equation 3.9
        Update parameters $\theta$ w.r.t. $g'$
    if $\mathcal{B}^*$.size() $> N_B$ and not $t$ mod $r$ then
        $\mathcal{B} \leftarrow$ SampleReplayBatch$(\mathcal{B}^*)$
        $g \leftarrow \hat{\mathbb{E}}_{(s,y)\sim\mathcal{B}} \nabla_\theta l\big(f_\theta(s), y\big)$
        $g_{\mathcal{M}} \leftarrow \hat{\mathbb{E}}_{(s,y)\sim\mathcal{M}} \nabla_\theta l\big(f_\theta(s), y\big)$
        Project $g$ to $g'$ using QP based on Equation 3.9
        Update parameters $\theta$ w.r.t. $g'$
```

motion for a given problem, we let it learn from the expert demonstrations for up to $N_c \in \mathbb{N}_{\geq 0}$ iterations (Line 9-10). If MPNet is not called or failed to determine a solution, an expert-planner is executed to determine a path solution σ for a given planning problem (Line 13). The expert trajectory σ is stored into a replay buffer $\mathcal{B}^*$ and an episodic $\mathcal{M}$ memory based on their sample selection strategies (Line 14-15). MPNet is trained (Line 16-19) on the given demonstration using the constraint optimization mentioned in Equations 3.7-3.9. Finally, similar to continual learning, we also perform rehearsals on the old samples (Line 20-25).

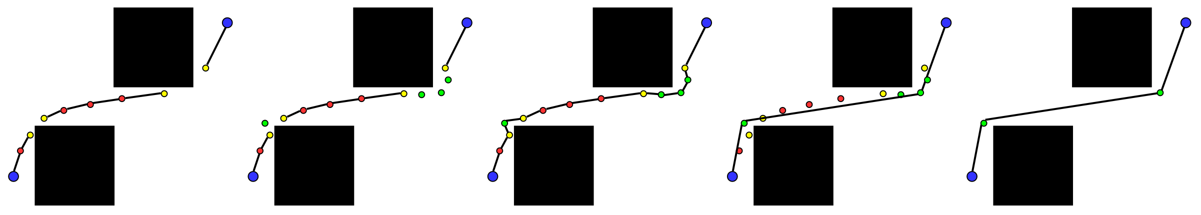


Figure 3.4: Online execution of MPNet: (a) global planning to output a coarse path. (b-c) a neural replanning step. (d) lazy states contraction (LSC) method to prune redundant states. (e) output feasible path.

3.1.4 MPNet: Online Planning

MPNet can generate both end-to-end collision-free paths and informed samples for the SMPs. We denote our path planner and sample generator as MPNetPath and MPNetSMP, respectively. MPNet uses two trained modules, Enet and Pnet.

Enet takes the obstacles' information x_{obs} and encodes them into a latent-space embedding Z. Enet is either trained end-to-end with Pnet or separately with encoder-decoder structure. Pnet, is a stochastic model as during execution it uses Dropout in almost every layer. The layers with Dropout [SHK$^+$14] get their neurons dropped with a probability $p \in [0,1]$. Therefore, every time MPNet calls Pnet, due to Dropout, it gets a sliced/thinner model of the original planning network which leads to a stochastic behavior. The stochasticity helps in recursive, divide-and-conquer based path planning, and also enables MPNet to generate samples for SMPs. The input to Pnet is a concatenated vector of obstacle-space embedding Z, robot current configuration $\hat{c}_t$, and goal configuration c_T. The output is the robot configuration $\hat{c}_{t+1}$ for time step $t+1$ that would bring the robot closer to the goal configuration. We iteratively execute Pnet (Fig. 3.3(b)), i.e., the new state $\hat{c}_{t+1}$ becomes the current state $\hat{c}_t$ in the next time step and therefore, the path grows incrementally.

Path Planning with MPNet

Path planning with MPNet uses Enet and Pnet in conjunction with our iterative, recursive, and bi-directional planning algorithm. Our planning strategy has the following salient features.

Forward-Backward Bi-directional Planning: Our algorithm is bidirectional as we run our neural models to plan forward, from start to goal, as well as to plan backward, from goal to start, until both paths meet each other. To connect forward and backward paths we use a greedy RRT-Connect [KL00] like heuristic.

Recursive Divide-and-Conquer Planning: Our planner is recursive and solves the given path planning problem through divide-and-conquer. MPNet begins with global planning (Fig. 3.4 (a)) that results in critical, collision-free states that are vital to generating a feasible trajectory. If any of the consecutive critical nodes in the global path are not connectable (Beacon states), MPNet takes them as a new start and goal, and recursively plans a motion between them (Figs. 3.4 (b-c)). Hence, MPNet decomposes the given problem into sub-problems and recursively executes itself on those sub-problems to eventually find a path solution.

In the remainder of this section, we describe the essential components of MPNetPath algorithm followed by the overall algorithm execution and outline.

Bidirectional Neural Planner (BNP): In this section, we formally outline our forward-backward, bidirectional neural planning method, outlined in Algorithm 4. BNP takes the environment representation Z, the robot's initial c_{init} and target c_{goal} configurations as an input. The bidirectional path is generated as two paths, from start to goal (σ_a) and from goal to start (σ_b), incrementally march towards each other. The paths σ_a and σ_b are initialized with the robot's start configuration c_{init} and goal configuration c_{goal}, respectively (Line 1). We expand paths in an alternating fashion, i.e., if at any iteration i, a path σ_a is extended, then in the next iteration, a path σ_b will be extended, and this is achieved by swapping the roles of σ_a and σ_b at the end of every iteration (Line 10). Furthermore, after each expansion step, the planner attempts to connect both paths through a straight-line, if possible. We use steerTo (described later) to perform straight-line connection which makes our connection heuristic greedy. Therefore, like RRT-Connect [KL00], our method makes the best effort to connect both paths after every path expansion. In case both paths σ^a and σ^b are connectable, BNP returns a concatenated path σ that comprises of states from σ_a and σ_b (Line 8-9).

Algorithm 4: BidirectionalNeuralPlanner($c_{\text{init}}, c_{\text{goal}}, Z$)

```
$\sigma^{\text{a}} \leftarrow \{c_{\text{init}}\}, \sigma^{\text{b}} \leftarrow \{c_{\text{goal}}\}$
$\sigma \leftarrow \varnothing$
for $i \leftarrow 0$ to $N$ do
    $c_{\text{new}} \leftarrow \text{Pnet}(Z, \sigma^{\text{a}}_{\text{end}}, \sigma^{\text{b}}_{\text{end}})$
    $\sigma^{\text{a}} \leftarrow \sigma^{\text{a}} \cup \{c_{\text{new}}\}$
    Connect $\leftarrow \text{steerTo}(\sigma^{\text{a}}_{\text{end}}, \sigma^{\text{b}}_{\text{end}})$
    if Connect then
        $\sigma \leftarrow \text{concatenate}(\sigma^{\text{a}}, \sigma^{\text{b}})$
        return $\sigma$
    SWAP($\sigma^{\text{a}}, \sigma^{\text{b}}$)
return $\varnothing$
```

Replanning: The bidirectional neural planner outputs a coarse path of critical points σ (Fig. 3.4 (a)). If all consecutive nodes in σ are connectable, i.e., the trajectories connecting them are in obstacle-free space then there is no need to perform any further planning. However, if there are any beacon nodes in σ, a replanning is performed to connect them (Fig. 3.4 (b-c)). The replanning procedure is presented in Algorithm 5. The trajectory σ consists of states $\sigma = \{c_0, c_1, \cdots, c_T\}$ given by BNP. The algorithm uses steerTo function and iterates over every connective state pairs σ_i and σ_{i+1} in a given path σ to check if there exists a collision-free straight trajectory between them. If a collision-free trajectory does not exist between any of the given consecutive states, a new path is determined between them through replanning. To replan, the beacon nodes are presented to the replanner as a new start and goal pair together with the obstacles information. We propose two replanning methods:

Algorithm 5: Replan(σ, Z, plan_oracle)

```
$\sigma_{\text{new}} \leftarrow \varnothing$
**for** $i \leftarrow 0$ *to* $size(\sigma) - 1$ **do**
    **if** steerTo($\sigma_i, \sigma_{i+1}$) **then**
        $\sigma_{\text{new}} \leftarrow \sigma_{\text{new}} \cup \{\sigma_i, \sigma_{i+1}\}$
    **else**
        **if** plan_oracle **then**
            $\sigma' \leftarrow$ OraclePlanner($\sigma_i, \sigma_{i+1}, \mathcal{X}_{\text{obs}}$)
        **else**
            $\sigma' \leftarrow$ BNP($\sigma_i, \sigma_{i+1}, Z$)
        **if** $\sigma'$ **then**
            $\sigma_{\text{new}} \leftarrow \sigma_{\text{new}} \cup \sigma'$
        **else**
            **return** $\varnothing$
**return** $\sigma_{\text{new}}$
```

(i) Neural replanning (NP): NP takes the beacon states and makes a limited number of attempts (N_r) to find a path solution between them using BNP. In case of NP, the plan_oracle in Algorithm 5 is set to False.

(ii) Hybrid replanning (HP): HP combines NP and oracle planner. HP takes beacon states and tries to find a solution using NP. If NP fails after a fixed number of trials N_r, an oracle planner is executed to find a solution, if one exists, between the given states (Line 9). HP is performed if boolean plan_oracle is set True in Algorithm 5.

Lazy States Contraction (LSC): This process is often known as smoothing or shortcutting [HNTH10]. The term contraction was coined in graph theory literature [Ski]. We implement LSC as a recursive function that when executed on a given path $\sigma = \{c_0, c_1, \cdots, c_T\}$, leaves no redundant state by directly connecting, if possible, the non-consecutive states, i.e., c_i and $c_{>i+1}$, where $i \in [0, T-1]$, and removing the lousy/lazy states (Fig. 3.4 (d)). This method improves the computational efficiency as the algorithm will have to process fewer nodes during planning.

*Steering (*SteerTo*):* The steerTo function, as it name suggests, walks through a straight line connecting the two given states to validate if their connecting trajectory lies entirely in the collision-free space or not. To evaluate the connecting trajectory, the steerTo function discretize the straight line into small steps and verifies if each discrete node is collision-free or not. The discretization is done as $\sigma(\delta) = (1-\delta)c_1 + \delta c_2; \forall \delta \in [0,1]$ between nodes c_1 and c_2 using a small step-size. In our algorithm, we use different step-sizes for the global planning, replanning, and for final feasibility checks.

isFeasible: This function uses the steerTo to check whether the given path $\sigma = \{c_0, \cdots, c_T\}$ lies entirely in collision-free space or not.

Algorithm 6: MPNetPath($c_{\text{init}}, c_{\text{goal}}, x_{\text{obs}}$)

```
$Z \leftarrow$ Enet($x_{\text{obs}}$)
$\sigma \leftarrow$ BNP($c_{\text{init}}, c_{\text{goal}}, Z$)// BidirectionalNeuralPlanner
$\sigma \leftarrow$ LazyStatesContraction($\sigma$)
if IsFeasible($\sigma$) then
    return $\sigma$
else
    plan_oracle = False// use NeuralReplanning
    for $i \leftarrow 0$ to $N_r$ do
        $\sigma \leftarrow$ Replan($\sigma, Z$, plan_oracle)
        $\sigma \leftarrow$ LazyStatesContraction($\sigma$)
        if IsFeasible($\sigma$) then
            return $\sigma$
    plan_oracle = True// use OraclePlanner
    $\sigma \leftarrow$ Replan($\sigma, Z$, plan_oracle)// HybridReplanning
    $\sigma \leftarrow$ LazyStatesContraction($\sigma$)
    if IsFeasible($\sigma$) then
        return $\sigma$
    return $\varnothing$
```

MPNetPath Execution Summary: Algorithm 6 outlines our MPNetPath framework. Enet encodes the raw point-cloud x_{obs} into latent embedding Z (Line 1). BNP takes the given planning problem $(c_{\text{init}}, c_{\text{goal}}, Z)$ and outputs a coarse path σ (Line 2). The LSC function takes σ to remove the redundant nodes and leaves only critical states crucial for finding a feasible path solution (Line 3). If a path constituted by connecting the consecutive nodes in σ does not belong to the collision-free region, a neural replanning (NP), followed by LSC, is performed for a fixed number of iterations N_r (Line 8-12). The neural-replanning recursively gets deeper and finer in connecting the beacon states whereas LSC function keeps on pruning out the redundant states after every replanning step. In most case, the neural replanner is able to compute a path solution. However, for some hard cases, where neural replanner fails to find a path between beacon states in σ, we employ hybrid planning (HP) using an oracle planner (Line 13-14). Note that, in case of hybrid replanning, the oracle planner is executed only for a small segment of the overall problem which helps MPNetPath retains its computational benefits while being able to determine a path solution if one exists.

Informed Sampling with MPNet

As mentioned earlier, our planning network (Pnet) is a stochastic model as it has Dropout in almost every layer during the execution. We exploit the stochasticity of Pnet to generate multiple informed samples that would be used as a sample generator for a classical SMP. Because samples are informed in such a way that has high probability to be part of a connectable and near-optimal path, it allows the underlying SMP to find solutions efficiently and quickly.

Algorithm 7 outlines the procedure to integrate our MPNetSMP with any SMP. MPNet's BNP performs one-step-ahead prediction, i.e., given obstacles representation Z, robot current state $\hat{c}_t$, and goal state c_T, it predicts a next state $\hat{c}_{t+1}$ closer to the goal than previous state $\hat{c}_t$. We execute MPNetSMP to incrementally generate samples from start to goal in a loop for a fixed number of iterations N_{smp} before switching to uniform random sampling (Line 5-8). The underlying SMP takes the informed samples (Line 9) and builds a tree starting from an initial configuration. Due to informed sampling, the tree, in the beginning, is biased towards a subspace that potentially contains a path solution and after certain iterations, the tree begins to expand

Algorithm 7: MPNetSMP($c_{\text{init}}, c_{\text{goal}}, x_{\text{obs}}$)

```
Initialize SMP($c_{init}, c_{goal}, \mathcal{X}_{obs}$)
$c_{rand} \leftarrow c_{init}$
$Z \leftarrow$ Enet($x_{obs}$)
for $i \leftarrow 0$ to $n$ do
    if $i < N_{smp}$ then
        $c_{rand} \leftarrow$ BNP$(Z, c_{rand}, c_{goal})$
    else
        $c_{rand} \leftarrow$ RandomSampler()
    $\sigma \leftarrow$ SMP$(c_{rand})$
    if $c_{rand} \in c_{goal}$ then
        $c_{rand} \leftarrow c_{init}$
if $\sigma_{end} \in c_{goal}$ then
    return $\sigma$
return $\varnothing$
```

uniformly. Hence, our MPNetSMP perform both exploitation and exploration. Once a path solution σ is found, it is either returned as a solution or further optimized for a given cost function.

We also propose that our MPNetSMP can be adapted to generate samples for bidirectional SMPs [QA15]. MPNetSMP generates informed samples incrementally between the given start and goal, and just like the bidirectional heuristic of MPNetPath, it can be queried to generate bidirectional samples. A simple modification for bidirectional sampling in Algorithm 7 would be to initialize two random variable c_{randS} and c_{randG} with start and goal states, respectively, in place of Line 2. Then use c_{randG} instead of c_{goal} and swap the roles of c_{randS} and c_{randG} at the end of every iteration. The proposed modification would allow informed bidirectional generation of trees that are crucial for solving the narrow passage planning problems.

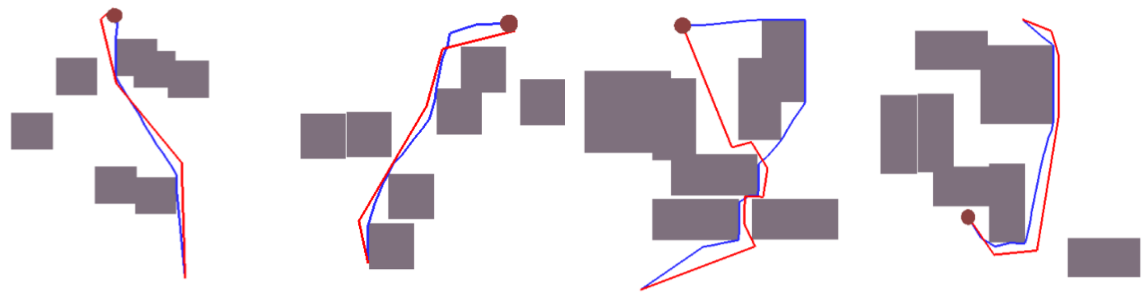
3.1.5 Implementation details

Data Collection:

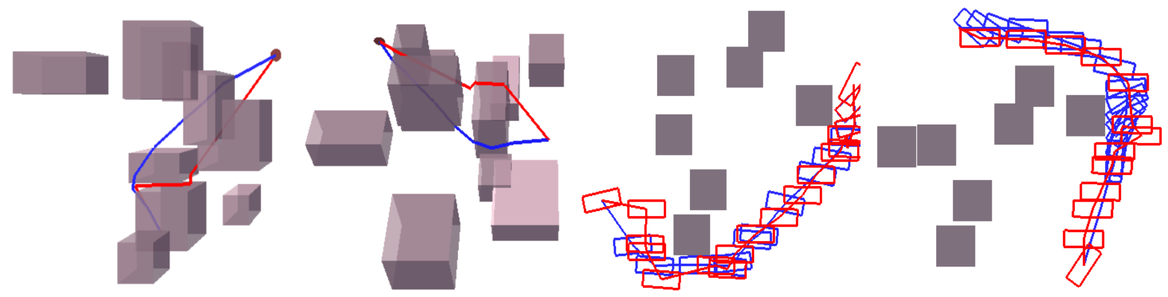
Environment settings: In the case of point-mass and rigid-body-SE2, we randomly place several quadrilateral blocks in the operating region of 40×40 and $40 \times 40 \times 40$ to generate different 2D and 3D workspaces, respectively. Each random placement of the obstacle blocks leads to a different environment/workspace. By following this procedure, 110 unique workspaces were generated for point-mass and rigid-body-SE2 cases, i.e., simple 2D (s2D), rigid-body-SE2, complex 2D (c2D) and complex 3D (c3D). The rigid-body-SE3 environment is a standard OMPL's [ŞMK12] home-like scenario. For Baxter robot, we created an L-shaped table top on which 5 different objects representing obstacles were randomly placed to constitute 10 different challenging environments in ROS Gazebo.

Point Cloud Generation: Since in point-mass and rigid-body-SE2 settings, the environments contained quadrilateral blocks as obstacles, we generate the point-cloud for each environment by randomly sampling points within those blocks. The point cloud for simple 2D, complex 2D and rigid-body-SE2 contained 1400 points in 2D space obstacles, i.e., 2×1400. The point cloud for complex 3D environment also contained 1400 points but in 3D space, i.e., 3×1400. These points were flattened to feed into the feed-forward neural encoder. For real-world complex scenarios, like in Baxter cases and rigid-body-SE3 home-like environment, we get a raw point-cloud using KINECT depth camera with the PCL [RC11] and pclros (http://wiki.ros.org/pcl). The point-cloud dimensions for Baxter and rigid-body-SE3 were 3×5351 and 3×150008, respectively. These point clouds can either be flattened or voxelized if Enet is a feed-forward neural network or a 3D convolutional neural network (CNN), respectively. For 3D CNN, we voxelize the point-cloud to $32 \times 32 \times 32$ dimensions.

Demonstration Trajectories: To create trajectories, we randomly sample a fixed number of obstacle-free states to create a list of start and goal pairs without repetition. We then use an oracle planner, in our case RRT*, to generate feasible near-optimal trajectories between the given start and goal pairs which forms the expert data for training and testing. In case of 2D/3D point-mass and rigid-body-SE2, the training dataset comprised 4000 paths per workspace with a total of 100 workspaces. We also created two test datasets, seen-$\mathcal{X}_{\text{obs}}$ and unseen-$\mathcal{X}_{\text{obs}}$, to evaluate our models. The seen-$\mathcal{X}_{\text{obs}}$ dataset comprised of 100 environments that were seen by the model during training but with 200 unseen start and goal pairs per environment. The unseen-$\mathcal{X}_{\text{obs}}$ dataset contained 10 unseen environments each of which contained 2000 unseen start and goal pairs. For Baxter robot, we run RRT* to collect 900 training and 100 testing paths for each of the ten environments between a fixed start configuration and a randomly selected goal configuration near the table top. For rigid-body-SE3, we collect 2000 training and 500 testing trajectories between randomly selected, unique collision-free poses of the rigid-body in the home environment.



(a) $t_r = 4.96s, t_m = 0.02s$ (b) $t_r = 5.35s, t_m = 0.02s$ (c) $t_r = 6.83s, t_m = 0.03s$ (d) $t_r = 6.22s, t_m = 0.04s$

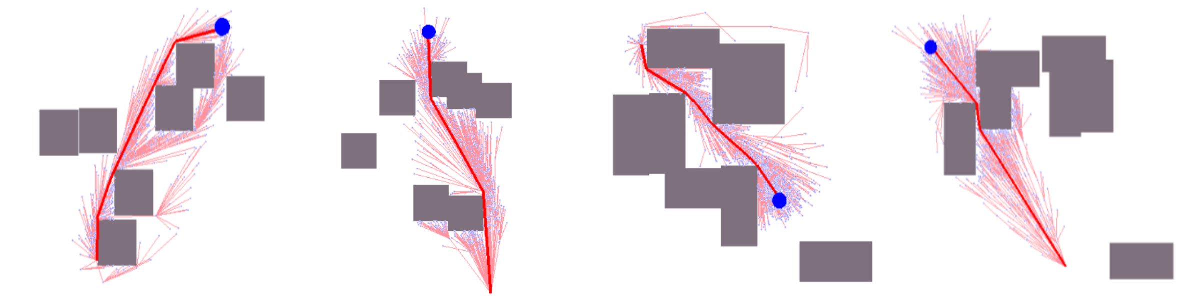


(e) $t_r = 8.37s, t_m = 0.07s$ (f) $t_r = 9.70s, t_m = 0.08s$ (g) $t_r = 26.61s, t_m = 0.38s$ (h) $t_r = 27.81s, t_m = 0.37s$

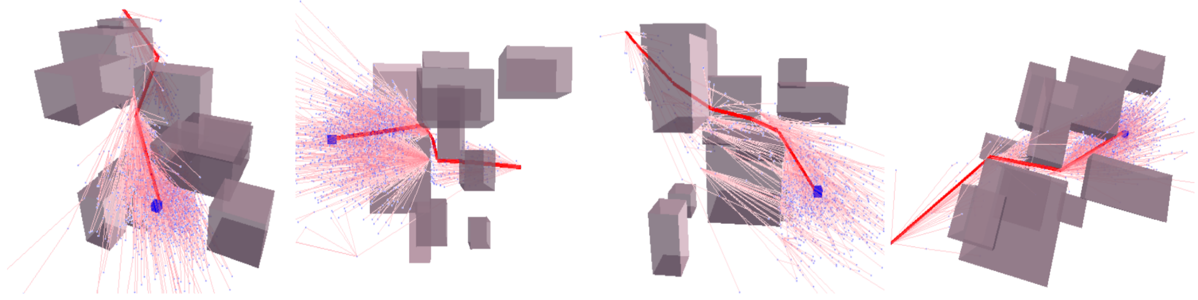
Figure 3.5: Time comparison of MPNetPath:NP (Red, t_m) and RRT* (Blue, t_r) for computing the near-optimal paths in environments such as simple 2D (a-b), complex 2D (c-d), complex 3D (e-f), and rigid-body-SE2 (g-h).

3.1.6 Results

In this section, we present results evaluating MPNet against state-of-the-art classical planners: RRT* [KF11], Informed-RRT* [GSB14], and BIT* [GSB15]. We use standard and efficient OMPL [ŞMK12] implementations for classical planners. MPNet models were trained



(a) $t = 0.13s, c = 32.46$ (b) $t = 0.13s, c = 38.02$ (c) $t = 0.25s, c = 27.57$ (d) $t = 0.21s, c = 37.23$



(e) $t = 0.22s, c = 40.73$ (f) $t = 0.18s, c = 36.48$ (g) $t = 0.29s, c = 39.51$ (h) $t = 0.21s, c = 40.26$

Figure 3.6: MPNetSMP generating informed samples for RRT* to plan motions in simple 2D (a-b), complex 2D (c-d) and complex 3D (e-h). The number of samples and time required to compute the path are denoted by n and t, respectively.

with the PyTorch Python API, exported using TorchScript [1] so that they could be loaded in OMPL for execution. Furthermore, for MPNet, we present the results of MPNetPath and MPNetSMP trained with offline batch learning (B), continual learning (C), and active continual learning (AC). The MPNetPath:NP and MPNetPath:HP uses neural and hybrid replanning, respectively. The hybrid replanning exploits neural replanner for a fixed number of iterations N_r. The system used for experiments has 3.40GHz$\times$ 8 Intel Core i7 processor with 32 GB RAM and GeForce GTX 1080 GPU.

Training & Testing Environments

We consider problems requiring the planning of 2D/3D point-mass robot, rigid-body-SE2, rigid-body-SE3, and a 7DOF Baxter robot manipulator. We used encoder-decoder based

[1]https://pytorch.org/tutorials/advanced/cpp_export.html

Table 3.1: Mean computation times with standard deviations are presented for MPNet (M) (all variations), Informed-RRT* and BIT* on two test datasets, i.e., seen and unseen (shown inside brackets), in four different environments.

Methods	Environments			
	Simple 2D	Complex 2D	Complex 3D	Rigid-body-SE2
IRRT*	1.06 ± 0.33 (1.10 ± 0.09)	1.60 ± 0.47 (1.49 ± 0.16)	2.99 ± 0.82 (2.76 ± 0.20)	15.58 ± 2.85 (14.80 ± 2.83)
BIT*	0.59 ± 0.28 (0.65 ± 0.30)	1.79 ± 1.35 (1.61 ± 0.53)	0.19 ± 0.12 (0.20 ± 0.04)	7.16 ± 1.95 (6.52 ± 1.65)
MSMP (B)	0.13 ± 0.01 (0.13 ± 0.00)	0.29 ± 0.05 (0.23 ± 0.07)	0.25 ± 0.05 (0.23 ± 0.06)	0.49 ± 0.05 (0.39 ± 0.04)
MPath:NP (C)	0.02 ± 0.00 (0.02 ± 0.00)	0.05 ± 0.01 (0.05 ± 0.01)	0.07 ± 0.01 (0.08 ± 0.01)	0.41 ± 0.08 (0.39 ± 0.07)
MPath:NP (AC)	0.03 ± 0.01 (0.03 ± 0.01)	0.06 ± 0.01 (0.06 ± 0.01)	0.08 ± 0.01 (0.08 ± 0.01)	0.53 ± 0.12 (0.42 ± 0.08)
MPath:NP (B)	0.02 ± 0.00 (0.02 ± 0.00)	0.04 ± 0.00 (0.04 ± 0.01)	0.06 ± 0.01 (0.07 ± 0.01)	0.38 ± 0.04 (0.37 ± 0.02)
MPath:HP (B)	0.04 ± 0.03 (0.07 ± 0.04)	0.13 ± 0.07 (0.14 ± 0.09)	0.09 ± 0.03 (0.12 ± 0.04)	0.42 ± 0.30 (0.41 ± 0.40)

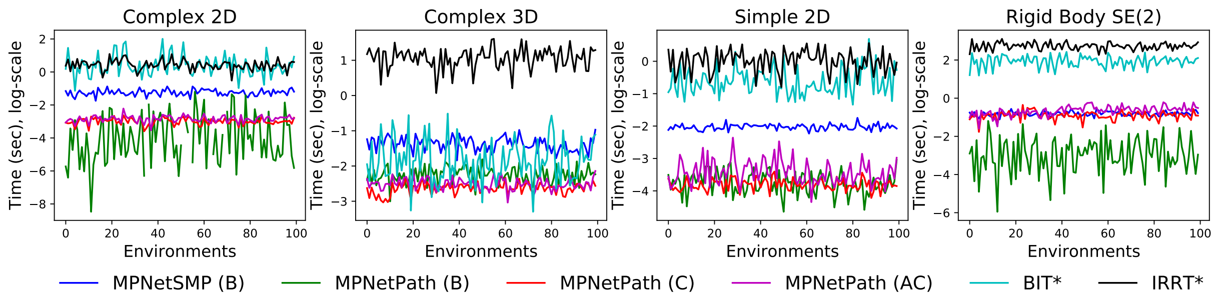


Figure 3.7: Mean computation time (log-scale) comparisons of MPNetPath:NP, MPNetSMP-RRT*, Informed-RRT* (IRRT*) and BIT* on seen-$\mathcal{X}_{\text{obs}}$ dataset.

unsupervised learning in the case of point-mass robots, and rigid-body-SE2. In Baxter and rigid-body-SE3 settings, we train Enet and Pnet together in an end-to-end setting.

In planning of 2D/3D point-mass and rigid-body-SE2, we train MPNet over one hundred workspaces with each containing four thousand trajectories, and it is evaluated on two test datasets, i.e., seen-$\mathcal{X}_{\text{obs}}$ and unseen-$\mathcal{X}_{\text{obs}}$. The seen-$\mathcal{X}_{\text{obs}}$ comprises of one hundred workspaces seen by MPNet during the training but two hundred unseen start and goal pairs in each environment.

Table 3.2: Success rates of all MPNet variants in the four environments on both test datasets, seen and unseen (shown inside brackets).

Methods	Environments			
	Simple 2D	Complex 2D	Complex 3D	Rigid-body-SE2
MPNetPath:NP (C)	93.33 (93.18)	83.44 (83.78)	89.88 (90.86)	83.770 (86.18)
MPNetPath:NP (AC)	96.70 (97.83)	84.36 (84.08)	96.60 (95.28)	87.08 (87.64)
MPNetPath:NP (B)	99.30 (98.30)	99.71 (98.80)	99.11 (97.76)	94.21 (95.18)
MPNetPath:HP (B)	100.0 (100.0)	100.0 (100.0)	100.0 (100.0)	100.0 (100.0)
MPNetSMP	100.0 (100.0)	100.0 (100.0)	100.0 (100.0)	100.0 (100.0)

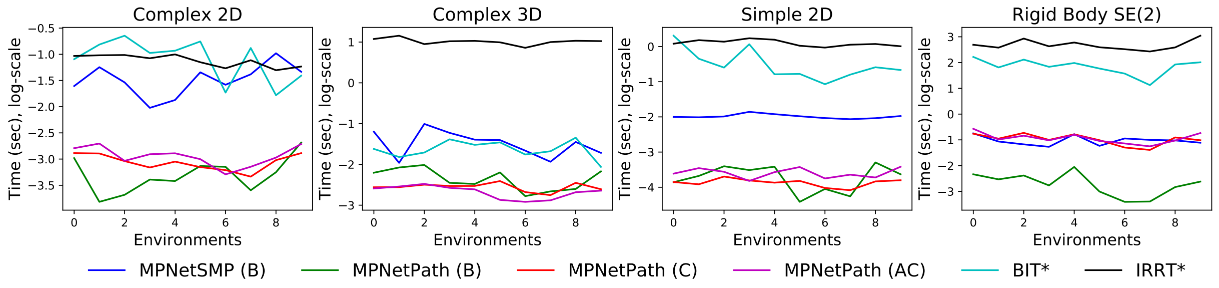


Figure 3.8: Mean computation time (log-scale) comparisons of MPNetPath:NP and MPNetSMP (with underlying RRT*) against Informed-RRT* (IRRT*) and BIT* on unseen-$\mathcal{X}_{\text{obs}}$ dataset.

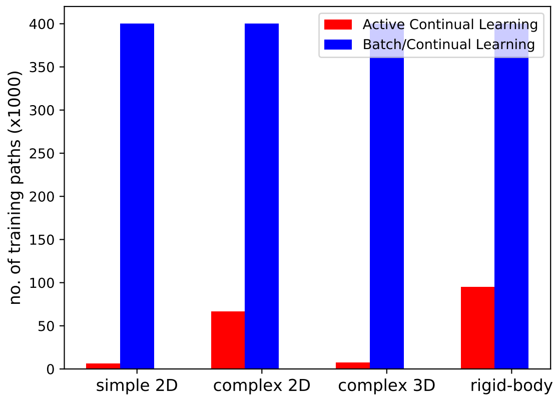


Figure 3.9: The number of training paths required by MPNet when trained with active continual learning and traditional learning.

The unseen-$\mathcal{X}_{\text{obs}}$ consists of ten new workspaces, not seen by MPNet during training, with each containing two thousand start and goal pairs. For the 7DOF Baxter manipulator, our training dataset containing ten cluttered environments with each having nine hundred trajectories, and our test dataset contained the same ten workspaces as in training but one hundred new start and goal pairs for each scenario. In rigid-body-SE3 planning, we consider OMPL's [ŞMK12] home-like environment, with training and testing dataset comprising two thousand trajectories and five-hundred unseen start and goal 3D poses of the rigid-body, respectively. For all planners presented in our experiments, we evaluate multiple trials on the given test dataset, and their mean performance metrics are reported.

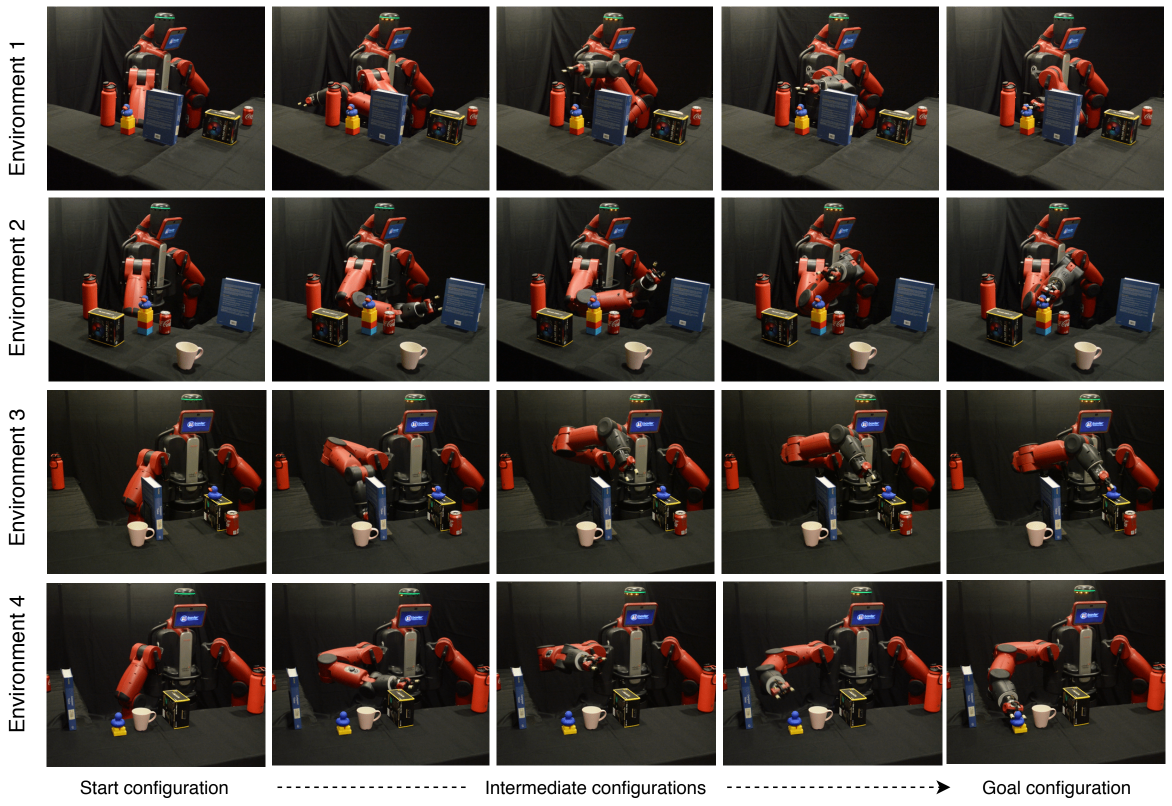


Figure 3.10: MPNetPath:NP plans motion for a Baxter robot in ten challenging environments, out of which four are shown. The left- and right-most indicate robot's initial and goal configurations, respectively, and the blue duck shows the target.

MPNet Comparison with its Expert Demonstrator (RRT*)

We compare MPNet against its expert demonstrator RRT*. Fig. 3.5 shows the paths generated by MPNetPath (red), and its expert demonstrator RRT* (blue). The average computations times of MPNetPath and RRT* are denoted as t_{R} and t_{MP}, respectively. The average Euclidean path cost of MPNetPath and RRT* solutions are denoted as c_{R} and c_{MP}, respectively. It can be seen that MPNet finds paths of similar lengths as its expert demonstrator RRT* while retaining consistently low computational time. Furthermore, the computation times of RRT* are not only higher than MPNetPath computation time but also grows exponentially with the dimensionality of the planning problems. Overall, we observed that MPNetPath is at least 100$\times$ faster than RRT* and finds paths that are within a 10% range of RRT*'s paths cost.

Fig. 3.6 presents informed trees generated by MPNetSMP with an underlying RRT* algorithm. We report average path computation times and Euclidean costs denoted as t and c, respectively. The generated trees by MPNetSMP are in a subspace of given configuration space that most of the time contains path solutions. It should be noted that MPNetSMP paths are almost optimal and are observed to be better than MPNetPath and its expert demonstrator RRT* solutions. Moreover, our sampler not only finds near-optimal/optimal paths but also exhibit a consistent computation time of less than a second in all environments which is much lower than the computation time of RRT* algorithm.

MPNet Comparison with Advanced Motion Planners

We further extend our comparative studies to evaluate MPNet against advanced classical planners such as Informed-RRT* [GSB14] and BIT* [GSB15].

Table 3.1 presents the comparison results over the four different scenarios, i.e., simple 2D (Fig. 3.5 (a-b)), complex 2D (Fig. 3.5 (c-d)), complex 3D (Fig. 3.5 (e-f)) and rigid-body-SE2 (Fig. 3.5 (g-h)). Each scenario comprises of two test datasets, seen-$\mathcal{X}_{\text{obs}}$ and unseen-$\mathcal{X}_{\text{obs}}$. We let Informed-RRT* and BIT* run until they find a solution of Euclidean cost within 5% range of the cost of MPNetPath solution. We report mean computation times with standard deviation over five trials. In all planning problems, MPNetPath:NP (with neural replanning), MPNetPath:HP (with hybrid replanning), and MPNetSMP exhibit a mean computation time of less than a second. The state-of-the-art classical planners, Informed-RRT* and BIT*, not only exhibit higher computation times than all versions of MPNet but, just-like RRT*, their computation times increase with the planning problem dimensionality. In simplest case (Fig. 3.5 (a-b)), MPNetPath:NP stand out to be atleast 80$\times$ and 30$\times$ faster than BIT* and Informed-RRT*, respectively. On the other hand, MPNetSMP provides about 8$\times$ and 5$\times$ computation speed improvements compared to BIT* and Informed-RRT*, respectively, in simple 2D environments. Furthermore, it can be observed that the speed gap between classical planner and MPNet (MPNetPath and MPNetSMP) keeps increasing with planning problem dimensions.

Fig. 3.7 and Fig. 3.8 present the mean computation time, in log-scale, of all MPNet variants, Informed-RRT* (IRRT*), and BIT* on seen-$\mathcal{X}_{\text{obs}}$ and unseen-$\mathcal{X}_{\text{obs}}$ test datasets, respectively. Note that MPNetPath trained with offline batch learning (B), continual learning (C), and active continual learning (AC) show similar computation times as can be seen by their plots in all planning problems. Furthermore, in Figs. 3.7-3.8, it can be seen that MPNetPath and MPNetSMP computation times remain consistently less than one second in all planning problems irrespective of their dimensions. The computation times of IRRT* and BIT* are not only high but also lack consistency and exhibits high variations over different planning problems. Although the computation speed of MPNetSMP is slightly lower than that of MPNetPath, it performs better than all other methods in terms of path optimality for a given cost function.

MPNet Data Efficiency & Performance Evaluation

Fig. 3.9 shows the number of training paths consumed by all MPNet versions and Table 3.2 presents their mean success rates on the test datasets in four scenarios- simple 2D, complex 2D, complex 3D, and rigid-body-SE2. The success rate represents the percentage of planning problems solved by a planner in a given test dataset. The models trained with offline batch learning provides better performance in term of a success rate compared to continual/active-continual learning methods. However, in our experiments, the continual/active-continual learning frameworks required about ten training epochs compared to one hundred training epochs of the offline batch learning method. Furthermore, we observed that continual/active-continual learning and offline batch learning show similar success rates if allowed to run for the same number of training epochs. The active continual learning is ultimately preferred as it requires less training data than traditional continual and offline batch learning while exhibiting considerable performance.

MPNet on 7DOF Baxter

Since BIT* performs better than Informed-RRT* in high dimension planning problems, we consider only BIT* as our classical planner in our further comparative analysis. We evaluate MPNet against BIT* for the motion planning of 7DOF Baxter robot in ten different environments where each environment comprised of one hundred new planning problems. Fig. 3.10 presents four out of ten environment settings. Each planning problem scenario contained an L-shape table of half the Baxter height with randomly placed five different real-world objects such as a book, bottle, or mug as shown in the figure. The planner's objective is to find a path from a given robot configuration to target configuration while avoiding any collisions with the environment, or self-collisions. Table 3.3 summarizes the results over the entire test dataset. MPNet on average find paths in less than a second with about 80% success rate. BIT* also finds the initial path in less than a second with similar success rate as MPNet. However, the path costs of BIT* initial solution are significantly higher than the path cost of MPNet solutions. Furthermore, we let BIT* run to improve its initial path solutions so that the cost of the paths are not greater than 140% of MPNet path costs. The results show that BIT* mostly fails to find solutions as optimal as MPNet solutions but demonstrate about 56% success rate in finding solutions that are usually 40% larger in lengths/costs than MPNet path costs. Furthermore, the computation time of BIT* also increases significantly, requiring an average of about 9 seconds to plan such paths. Fig. 3.12 shows the mean computation time and path cost comparison of BIT* and MPNet over ten individual environments. It can be seen that MPNet computes paths in less than a second while giving incredibly optimized solutions and is much faster compared to BIT*.

Fig. 3.11 shows a multi-goal planning problem for 7 DOF Baxter robot. The task is to reach and pick up the target object while avoiding any collision with the environment and transfer it to another target location, shown as yellow block. Note that in previous experiments we let BIT* run until it finds paths within 40% of MPNet path costs. Now, we let BIT* run until it finds a path within 10% cost of the cost of MPNet path solution. Our result shows that MPNetPath:NP

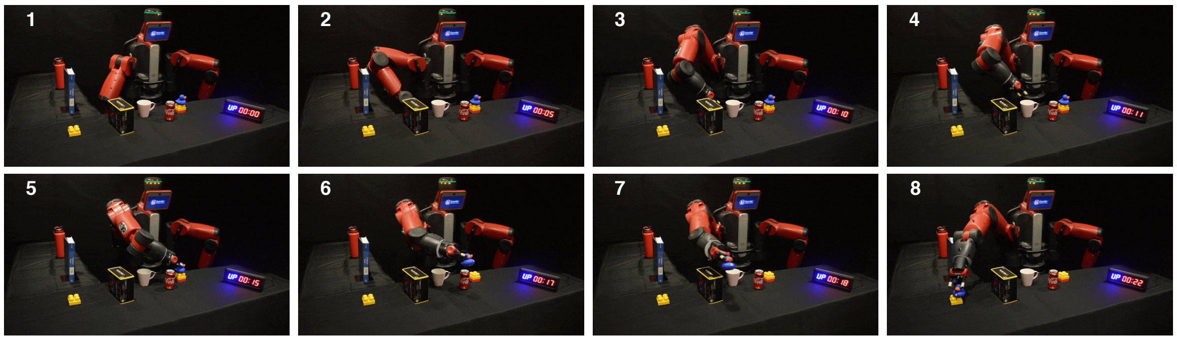


Figure 3.11: MPNetPath plans motion to pick up the blue object (duck), and move it to a new target (yellow block) (Frame 8). Note that the stopwatch indicates the execution time not the planning time.

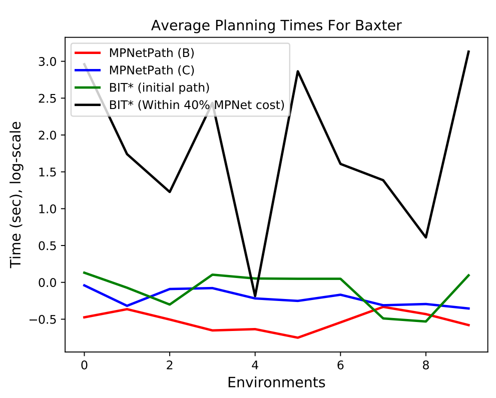
Table 3.3: Computation time, path cost, and success rate comparison of MPNetPath:NP, and BIT* on Baxter test dataset. BIT*'s times for finding the first path and further optimizing it within 40% range of MPNet's path cost are reported.

Methods	Baxter		
	Time	Path cost	Success rate(%)
BIT* (Initial Path)	0.94 ± 0.20	13.91 ± 0.60	83.0
BIT* (±40% MPNet cost)	9.20 ± 7.61	10.78 ± 0.31	56.0
MPNetPath (C)	0.81 ± 0.08	6.98 ± 0.18	78.6
MPNetPath (B)	0.59 ± 0.08	7.86 ± 0.20	87.8

plans the entire trajectory in less than a second whereas BIT* took about 3.01 minutes to find a path of similar cost as our solution. Note that MPNet is never trained on trajectories that connect two configurations on the table. However, thanks to MPNet ability to generalize that it can solve completely unseen problems in no time compared to classical planners that have to perform an exhaustive search before proposing any path solution.

MPNet planning in SE (3)

To further extend our comparison of MPNet and BIT*, we also consider planning in SE (3) for a rigid-body in a cluttered home-like environment with multiple narrow passages [ŞMK12], as shown in the Fig. 3.13. Our test dataset comprised 500 randomly sampled start and goal poses (not seen by MPNet during training), and the Fig. 3.13 shows two of those test cases. In these settings, MPNet exhibits a success rate of about 85%. The mean computation time to find an initial path solution for MPNet (B), MPNet (C), and BIT* were 0.96, 1.61, and 2.84 seconds,



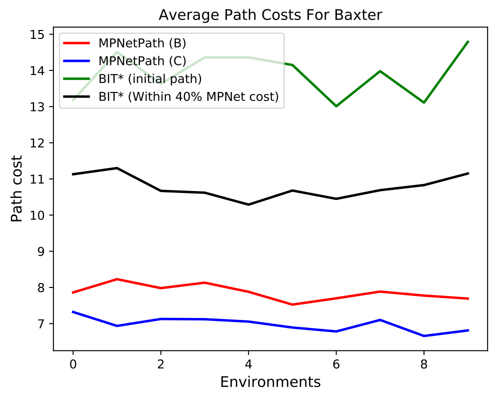


Figure 3.12: Computational time (log-scale) and path length comparison of MPNet and BIT* over ten challenging environments with 7DOF Baxter manipulator.

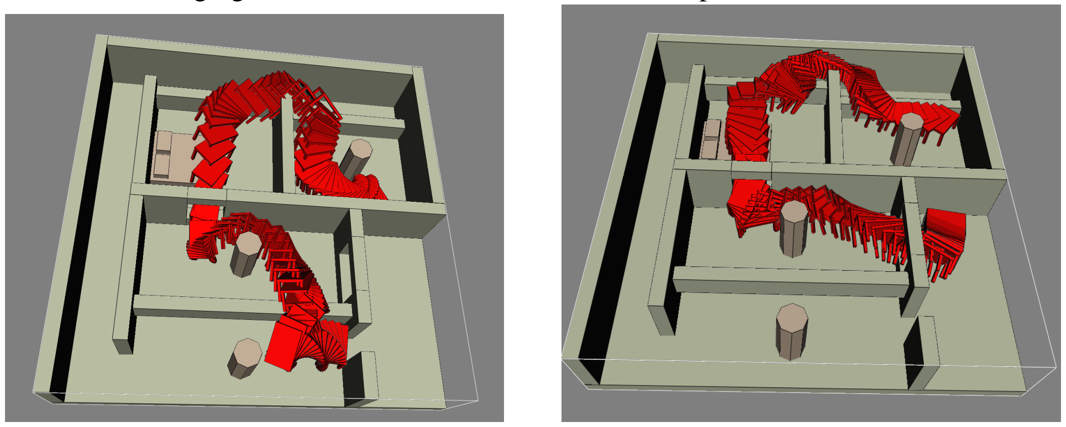


Figure 3.13: MPNet plans the motion of a rigid-body in SE(3) in a cluttered home-like environment with multiple narrow passages for randomly selected start and goal poses.

respectively. We also observed that MPNet paths' cost were significantly lower than the cost of BIT* solutions. The mean path costs of MPNet (B), MPNet (C), and BIT* were 457.80, 512.73, and 836.85, respectively. Furthermore, we noticed that BIT* takes up to several minutes to find a path of similar cost as MPNet's solution.

3.1.7 Discussion

This section presents a discussion on various attributes of MPNet, sample selection methods for continual learning, and a brief analysis of its completeness, optimality, and computational guarantees.

Stochastic Neural Planning & Replanning

In MPNet, we apply Dropout [SHK+14] to almost every layer of the planning network, and it remains active during path planning. The Dropout randomly selects the neurons from its preceding layer with a probability $p \in [0,1]$ and masks them so that they do not affect the model decisions. Therefore, at any planning step, Pnet would have a probabilistically different model structure that results in stochastic behavior.

Yarin and Zubin [GG16] demonstrate the application of Dropout to model uncertainty in the neural networks. We show that the Dropout can also be used in learning-based motion planners to approximate the subspace of a given configuration space that potentially contains several path solutions to a given planning problem. For instance, it is evident from the trees generated by MPNetSMP (Fig. 3.6) that our model approximates the subspace containing path solutions including the optimal path.

The perturbations in Pnet's output, thanks to Dropout, also play a crucial role during our neural replanning (Algorithm 5). In neural replanning, MPNet takes its own global path and uses its stochastic neural planner (Algorithm 6) to replan a motion between any beacon states. Replanning is a crucial component of our planning algorithm. Although the global plan comprises collision-free nodes, the straight-line connection between those nodes might not be collision-free (Fig. 3.4 (b)). We observed that without replanning, MPNet exhibits a low success rate of about $60-70\%$. Since global planning decomposes a given problem into sub-problems, the replanning between beacon nodes to solve sub-problems through our stochastic neural planner helps our method to recover from any failures leading to a high success rate in complex, cluttered environments.

Sample Selection Strategies

In continual learning, we maintain an episodic memory of a subset of past training data to mitigate catastrophic forgetting when training MPNet on new demonstrations. Therefore, it is crucial to populate the episodic memory with examples that are important for learning a generalizable model from streaming data. We compare four MPNet models trained on a simple 2D environment with four different sample selection strategies for the episodic memory. The four selection metrics include surprise, reward, coverage maximization, and global distribution matching. The surprise and reward metrics give higher and lower priority to examples with large prediction errors, respectively. The coverage maximization maintains a memory of k-nearest neighbors. The global distribution matching, as described earlier, uses random selection techniques to approximate the global distribution from streaming data. We evaluate four MPNet models on two test datasets, seen-$\mathcal{X}_{\text{obs}}$ and unseen-$\mathcal{X}_{\text{obs}}$, from simple 2D environment, and their mean success rates are reported in Fig. 3.14. It can be seen that global distribution matching exhibits slightly better performance than reward and coverage maximization metrics. The surprise metric gives poor performance because satisfying the constraint in Equations 3.9 becomes nearly impossible when the episodic memory is populated with examples that have large prediction losses. Since global distribution matching provides the best performance overall, we have used it as our sample selection strategy for the episodic memory in the continual learning settings.

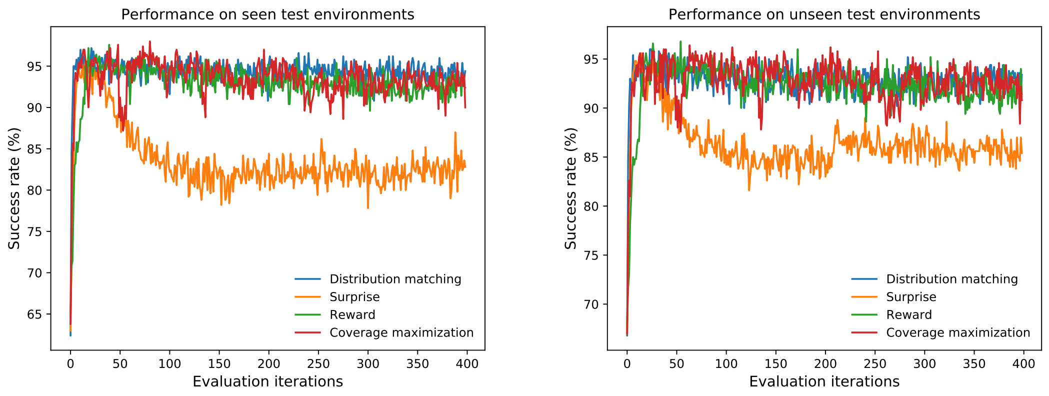


Figure 3.14: Impact of sample selection methods for episode memory on the performance of continual learning for MPNet in simple 2D test datasets seen-$\mathcal{X}_{\text{obs}}$ and unseen-$\mathcal{X}_{\text{obs}}$.

Batch offline & Active Continual Learning

In this section, we briefly highlight the merits of our training approaches. A batch offline learning is preferred when plenty of data is available for offline training, especially in cases where existing planners can be used to generate data. However, in cases where data is expensive to make, an active continual learning approach is preferable as it asks for demonstration only when needed leading to data-efficient training of our models. For instance, in semi-autonomous driving, the planning problems would usually come in streams, and our neural planner will only ask for a human demonstration when needed. Also, to realize classical life-long learning one can combine both batch offline and active continual learning where models are pre-trained with the former and further refined with the latter whenever needed.

Completeness

In this section, we discuss the completeness guarantees of MPNet, formally proposed as follow:

Proposition 3.1 (Feasible Path Planning): *Given a planning problem* $\{c_{\text{init}}, c_{\text{goal}}, x_{\text{obs}}\}$*, and a collision-checker, MPNet finds a path solution* $\sigma : [0, T]$*, if one exists, such that* $\sigma_0 = c_{\text{init}}$*,* $\sigma_T \in c_{\text{goal}}$*, and* $\sigma \subset \mathcal{C}_{\text{free}}$*.*

Proposition 3.1 implies that for a given planning problem, MPNet will eventually find a feasible path, if one exists. We show that the worst-case completeness guarantees of our approach rely on the underlying SMP for both path planning (MPNetPath) and informed neural sampling (MPNetSMP). In our experiments, we use RRT* as our oracle SMP that exhibits *probabilistic completeness*.

The *probabilistic completeness* is described in Definition 3.1 based on the following notations. Let $\mathcal{T}_n^{AL}$ be a connected tree in obstacle-free space, comprising of n vertices, generated by an algorithm AL. We also assume $\mathcal{T}_n^{AL}$ always originates from the robot's initial configuration

c_{init}. An algorithm is probabilistically complete if it satisfies the following definition.

Definition 3.1 (Probabilistic Completeness): *Given a planning problem* $\{c_{\text{init}}, c_{\text{goal}}, \mathcal{X}_{\text{obs}}\}$ *for which there exists a solution, an algorithm AL with a tree* $\mathcal{T}_n^{AL}$ *that originates at* c_{init} *is considered probabilistically complete iff* $\lim_{n\to\infty} \mathbb{P}(\mathcal{T}_n^{AL} \cap c_{\text{goal}} \neq \varnothing) = 1$.

RRT* algorithm exhibits *probabilistic completeness* as it builds a connected tree originating from initial robot configuration and randomly exploring the entire space until it finds a goal configuration. Therefore, as the number of samples in the RRT* approach to infinity the probability of finding a path solution, if one exists, gets to one, i.e.,

$$\lim_{n\to\infty} \mathbb{P}(\mathcal{T}_n^{RRT^*} \cap c_{\text{goal}} \neq \varnothing) = 1 \tag{3.10}$$

In remainder of the section, we present a brief analysis showing that MPNetPath and MPNetSMP also exhibit *probabilistic completeness* like their underlying RRT*.

Probabilistic completeness of MPNetPath: To justify MPNetPath worst-case guarantees, we introduce the following assumptions that are crucial to validate a planning problem.

Assumption 3.1 (Feasibility of States to Connect): *The given start and goal pairs* $(c_{\text{init}}, c_{\text{goal}})$*, for which an oracle-planner is called to find a path, lie entirely in obstacle-free space, i.e.,* $c_{\text{init}} \in \mathcal{C}_{\text{free}}$ *and* $c_{\text{goal}} \subset \mathcal{C}_{\text{free}}$.

Assumption 3.2 (Existence of a Path Solution) *Under Assumption 1, for a given problem* $(c_{\text{init}}, c_{\text{goal}}, \mathcal{X}_{\text{obs}})$*, there exists at least one feasible path* σ *that connects* c_{init} *and* c_{goal}*, and* $\sigma \not\subset \mathcal{X}_{\text{obs}}$.

Based on Definition 3.1 and Assumptions 3.1-3.2, we propose in Theorem 3.1, with a sketch of proof, that MPNetPath also exhibits *probabilistic completeness*, just like its underlying oracle planner RRT*.

Theorem 3.1 (Probabilistic Completeness of MPNetPath): *If Assumptions 1-2 hold, for a given planning problem* $\{c_{\text{init}}, c_{\text{goal}}, \mathcal{X}_{\text{obs}}\}$*, and an oracle RRT* planner, MPNetPath will find a* *path solution, if one exists, with a probability of one as the underlying RRT* is allowed to run till infinity if needed.*

Sketch of Proof: Assumption 3.1 puts a condition that the start and goal states in the given planning problem lie in the obstacle-free space. Assumption 3.2 requires that there exist at least one collision-free trajectory for the given planning problem. During planning, MPNetPath first finds a coarse solution σ that might have beacon (non-connectable consecutive) states (see Fig. 3.4). Our algorithm connects those beacon states through replanning. Assumption 3.1 holds for the beacon states as each generated state of MPNetPath is evaluated by an oracle collision-checker before making it a part of σ. In replanning, we perform neural replanning for a fixed number of trials to further refine σ, and if that fails to conclude a solution, we connect any remaining beacon states of the refined σ by RRT*. Hence, if Assumption 3.1-3.2 hold for beacon states, MPNetPath inherits the convergence guarantees of the underlying planner which in our case is RRT*. Therefore, MPNetPath with an underlying RRT* oracle planner exhibits probabilistic completeness.

Probabilistic completeness of MPNetSMP MPNetSMP generates samples for SMPs such as RRT*. Our method performs exploitation and exploration to build an informed tree. It begins with exploitation by sampling a subspace that potentially contains a solution for a limited time and switches to exploration via uniform sampling to cover the entire configuration space. Therefore, like RRT*, the tree of MPNetSMP is fully connected, originates from the initial robot configuration, and eventually expands to explore the entire space. Hence, the probability that MPNetSMP tree will find a goal configuration, if possible, approaches one as the samples in the tree reaches a large number, i.e.,

$$\lim_{n\to\infty} \mathbb{P}(\mathcal{T}_n^{\text{MPNetSMP}} \cap c_{\text{goal}} \neq \varnothing) = 1 \tag{3.11}$$

Optimality

MPNet learns through imitating the expert demonstrations which could come from either an oracle motion planner or a human demonstrator. In our experiments, we use RRT* for generating expert trajectories, hybrid replanning in MPNetPath, and as a baseline SMP for our MPNetSMP framework. Optimal SMPs such as RRT* exhibits asymptotic optimality, i.e., a probability of finding an optimal path, if one exists, approaches one as the number of random samples in the tree approaches infinity (for proof, refer to [KF11]). RRT* gets weak-optimality guarantees from its rewiring heuristic. The rewiring incrementally updates the tree connections such that over the time, each node in the tree would be connected to a branch that ensures lowest cost path to the root state (initial robot configuration) of the tree.

In our experiments, we show that MPNetPath imitates the optimality of its expert demonstrator. In Fig. 3.5, it can be seen that MPNet path solutions are of similar Euclidean costs as its expert demonstrator RRT* path solutions. Therefore, the quality of the MPNet paths relies on the quality of its training data.

In the case of MPNetSMP, there exists optimality guarantees depending on the underlying SMP. Since we use RRT* as a baseline SMP for MPNetSMP, we formally propose in Proposition 3.2 that our method exhibits asymptotic optimality.

Proposition 3.2 (Optimal Path Planning): *Given a planning problem* $\{c_{\text{init}}, c_{\text{goal}}, x_{\text{obs}}\}$*, a collision-checker, and a cost function* $J(\cdot)$*, MPNet can adaptively sample the configuration space for an asymptotic optimal SMP such that the probability of finding an optimal path solution, if one exists, w.r.t.* $J(\cdot)$ *approaches one as the number of samples in the graph approaches infinity.*

In our sampling heuristic, MPNetSMP generates informed samples for a fixed number of iterations and performs uniform random sampling afterward. Therefore, RRT* tree formed by the samples of MPNetSMP will explore the entire C-space and will be rewired incrementally to ensure asymptotic optimality. Since MPNetSMP only performs intelligent sampling and does not modify the internal machinery of RRT*, the optimality proofs will exactly be the same as provided in [KF11]. Hence, provided that MPNetSMP performs exploitation for fixed steps and pure exploration afterward, and the underlying SMP is RRT*, the optimality of our method is asymptotically guaranteed .

Computational Complexity

In this section, we present the computational complexity of our method. Both MPnetPath and MPNetSMP take the output of Enet and Pnet, which is only a forward-pass through a neural network. The forward pass through a neural network is known to have a constant complexity since it is a matrix multiplication of a fixed size input vector with network weights to generate a fixed size output vector. Since MPNetSMP produces samples by merely forward passing a planning problem through Enet and Pnet, it does not add any complexity to the underlying SMP. Therefore, the computational complexity of MPNetSMP with underlying RRT* is the same as RRT* complexity, i.e., $O(n\log n)$, where n is the number of samples [KF11].

On the other hand, MPNetPath has a crucial replanning component that uses an oracle planner in the cases where neural replanning fails to determine an end-to-end collision-free path. The oracle planner in the presented experiments is RRT*. Therefore, in the worst-case, MPNetPath operates with complexity of $O(n\log n)$. However, we show empirically that MPNetPath succeeds, without any oracle planner, for up to 90% of the cases in a challenging test dataset. For the remaining 10% cases, oracle planner is executed only for a small segment of the overall path (see Algorithm 6). Hence, even for hard planning problems where MPNetPath fails, our method reduces a complicated problem into a simpler problem, thanks to its divide-and-conquer approach, for the underlying oracle planner. Thus, the oracle planner operates at a complexity which is practically much lower than its worst-case theoretical complexity, as is also evident from the results of MPNetPath with hybrid replanning (HB) in Table 3.1.

3.2 Kinodynamic Constraints

As mentioned earlier, the motion planning problem is to find a path connecting the given start and goal states while satisfying all the desired constraints on the robot motion. Geometric motion planning is a simple instance that considers only collision-avoidance constraints with states representing the robot's kinematic variables (e.g., position, joint-angles) [LaV06]. However, in KMP, the state comprises position, velocity, and sometimes acceleration. The robot motions are required to satisfy both kinematic (e.g., collision-avoidance) and dynamics (e.g., velocity and acceleration) constraints, which makes the problem PSPACE-hard and computationally demanding [LaV06].

The KMP has a broad range of applications from torque-constrained robot manipulation to speed racing [AYYS16] and performing acrobatic motions [KLR$^+$20]. The existing state-of-the-art methods solve KMP through Sampling-based Motion Planners (SMPs) [LaV06] by constructing a tree that originates from a start state and expands by searching the entire state space to reach the given goal state. The edges of the tree, connecting any two intermediate states, are formed by a local steering function. The local steering often requires solving a boundary-value problem (BVP) through trajectory optimization, which is known to be NP-hard [Kac02] and fails when the boundary constraints are not satisfied. A recent development led to another sampling-based KMP approach [LLB16], called Stable Sparse-RRT (SST), that circumvents solving BVPs by using a random shooting method for steering and achieves asymptotic optimality. However, it still takes long computation times from tens of seconds to minutes due to uniform exploration of state and control spaces as environments become more complicated.

To overcome the limitation of prior methods and utilize strengths of neural motion planners, we propose an imitation learning-based approach named Model-Predictive Motion Planning Networks (MPC-MPNet)[2] [LMQY21] . It has a deep neural networks-based generator and discriminator, which, once trained using expert data, outputs feasible paths that satisfy the

[2]Supplementary videos and code release details are available at https://sites.google.com/view/mpc-mpnet

given kinodynamic constraints. Our approach extends the Motion Planning Networks (MPNet) [QMSY20] and leverages collision-aware Model Predictive Control (MPC) methods in the planning loop for parallelizable local steering. Note that the original MPNet framework [QMSY20] considered only geometric planning problems and relied on bidirectional path expansions and re-planning steps for finding path solutions to given problems. In KMP, the bidirectional path extensions and re-planning steps to repair in-collision path segments often lead to discontinuity and infeasible paths. Therefore, in MPC-MPNet, we propose novel planning algorithms that perform only forward path expansion and avoid re-planning by building a set of informed possible paths. We evaluate our approach in challenging under-actuated robotics problems in complex, cluttered environments. Our results show that MPC-MPNet outperforms state-of-the-art planning algorithms in computational speed and generalizes to unseen planning problems with high success rates.

3.2.1 Related Work

This section presents a brief overview of related work done in the past for solving KMP problems. KMP algorithms depend on the local steering function to satisfy the motion constraints between any two given states. The local steering methods are usually implemented either as (i) random shooting methods which uniformly samples control sequences [DXCR93] [LKJ01] [LLB16], (ii) predefined motion primitives [SBFP19], or (iii) local trajectory optimization solvers [XvdBPA15] [PPK$^+$12] [GPPK13] [TMTR10].

Sampling random control sequences for shooting method has the advantage of worst-case theoretical guarantees. For instance, the SST approach [LLB16] leverages random exploration of control and state spaces to provide the worst-case asymptotic completeness and optimality guarantees. However, these methods struggle in higher-dimensional and cluttered planning spaces since uniform exploration takes large computation times to determine a feasible path solution. In contrast, the motion primitives methods generate a local database of optimal controllers in their offline stage [SBFP19] and use them for steering under KMP algorithms. However, as the motion primitive set is finite, it usually does not capture the entire control space and lacks completeness guarantees.

The trajectory optimization methods formulate local steering as an optimization problem and iteratively solve them for computing action sequences connecting the given states. For instance, [WVDB13] [GPPK13] locally linearize the system and obtain Linear Quadratic Regulators' (LQR) parameters using optimization for steering. Tedrake et. el [TMTR10] extend LQR methods to construct a tree connecting states within stability regions defined by LQR funnels. However, these methods add a substantial computational burden when constructing trees and do not apply to online planning problems. In a similar vein, Xie et. el [XvdBPA15] formulated local steering as BVP and used optimization to find their solutions. These BVP solvers operate in conjunction with traditional SMPs [LaV06] for finding solutions to KMP problems. However, their approach often collapse when the boundary constraints are not satisfied.

Another class of methods in KMP learn local controllers for steering through reinforcement and imitation learning. For instance, [CHF$^+$19] uses reinforcement learning (RL) to acquire local policy for steering within SMP methods. Similarly, [OSJ$^+$20] [OJO$^+$20] constructs high-level landmarks, which are later connected by local RL policies. However, these methods inherit RL limitations such as requiring exhaustive interactions with their environments for learning. In the imitation learning paradigm, [WBMW18] and [AP19] use expert demonstrations to learn local policies for connecting given states, but they still rely on classical planners to generate those state sequences.

Recent developments overcome the limitations of classical motion planning by incorporating learned planning distributions into their algorithms [QSBY19] [QMSY20] [IHP18] [IP19]. These learning-based planners quickly generate feasible motion sequences online for finding path solutions but mostly consider geometric planning problems. Among them, MPNet [QSBY19] [QMSY20] generates end-to-end paths by exploiting learned distributions in a bidirectional manner throughout its planning and replanning process. MPNet has also been extended to consider kinematic [QDCY20] and non-holonomic [JLL+20] constraints. However, these extensions still rely on bidirectional planning, which often leads to discontinuities and infeasible paths in KMP problems.

3.2.2 Model Predictive Motion Planning Networks

In this section, we formally present MPC-MPNet, an end-to-end learning-based KMP algorithm. Let $\mathcal{C}$ denote a configuration space (C-space) of a mechanical system, where collision and collision-free regions are represented as $\mathcal{C}_{obs}$ and $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$, respectively. Let $\mathcal{X}$ denote the state space in which a state, $x = (c, \dot{c}) \in \mathcal{X}$, contains a configuration $c \in \mathcal{C}$ and the time derivative $\dot{c}$. Like C-space, the state space also contains the collision $\mathcal{X}_{obs}$ and collision-free $\mathcal{X}_{free}$ state spaces. The system's dynamics model, represented by an implicit set of equations, can be formulated as $\dot{x} = f(x, u)$, where u denotes the control input to a system from a feasible control set $\mathcal{U}$. In general, the objective of KMP for the given initial state x_{init} and goal region $\mathcal{X}_{goal} \subset \mathcal{X}_{free}$ is to find a collision-free trajectory $\sigma = [(x, u, \tau)_t]_{t=0}^{T}$, comprising a sequence of states $[x_t]_{t=0}^{T} \mapsto \mathcal{X}_{free}$ and controls $[u_t]_{t=0}^{T} \mapsto \mathcal{U}$ with their corresponding durations $[\tau_t]_{t=0}^{T}$, such that $x(0) = x_{init}$, $x(T) \in \mathcal{X}_{goal}$.

MPC-MPNet iteratively generates waypoints and local steering trajectories to construct collision-free paths connecting the start and goal states under kinodynamic constraints. It includes a neural generator, discriminator, and two novel planning algorithms named MPC-MPNetPath and MPC-MPNetTree. The main components of our approach are described as follows.

Observation Encoder

The observation encoder embeds the workspace information, represented as voxel maps v, into latent features Z containing critical anchor points for the underlying neural generator and discriminator. The voxel maps are volumetric with dimensions $L \times W \times H \times C$, corresponding to length, width, height, and number of channels, respectively, and are usually processed by 3D convolutional neural networks (CNNs). However, we convert the voxel maps into voxel patches with dimensions $L \times W \times \hat{C}$, where $\hat{C} = HC$, and use the 2D-CNNs for learning the embeddings. This is because 3D-CNNs are known to be computationally inefficient as their representations are inherently cubic and usually contain empty volumes [ZLU18].

Neural Generator

The neural generator G, with parameters θ_g, is a stochastic neural model that outputs intermediate waypoints $\hat{x}_{t+1}$ for the given environment encoding Z, robot's current state $x_t \in \mathcal{X}_{free}$ and goal state $x_{goal} \in \mathcal{X}_{goal}$, i.e.,

$$\hat{x}_{t+1} \leftarrow G(Z, x_t, x_{goal}; \theta_g) \tag{3.12}$$

The neural generator adopts Dropout [SHK+14] in almost every layer to generate stochastic samples. The Dropout randomly selects neurons in each trained network layer, resulting in a sliced model in every forward pass. This operation leads to randomness adapted from expert data. In contrast, other techniques, such as input Gaussian noise, are agnostic of underlying data distribution and incurs difficulty in training deep neural models [CWD+18].

Our neural generator is trained end-to-end with the observation encoder by optimizing the following mean square error between the predicted states $\hat{x}$ and the demonstration trajectories' states x^*, i.e.,

$$L_{G_{\theta_g}} = \frac{1}{N_p} \sum_{i=0}^{N} \frac{1}{T_i} \sum_{j=0}^{T_i - 1} \|\hat{x}_{i,j+1} - x^*_{i,j+1}\|^2 \tag{3.13}$$

where N_p is the total number of paths and T_i is the length of each path i in the dataset.

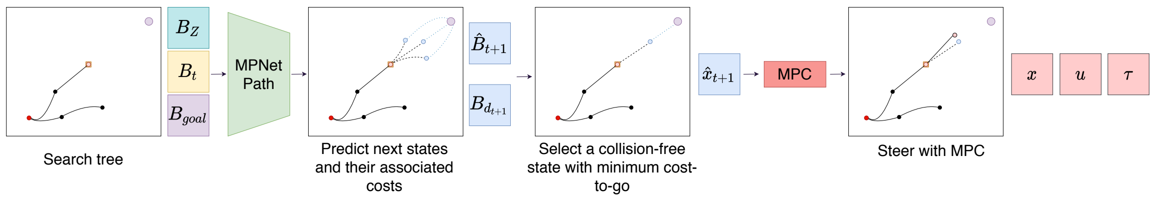


Figure 3.15: MPC-MPNetPath In each iteration, the neural generator predicts a batch of next states from a given current and select a collision-free state with a minimum estimated cost for the tree expansion using MPC.

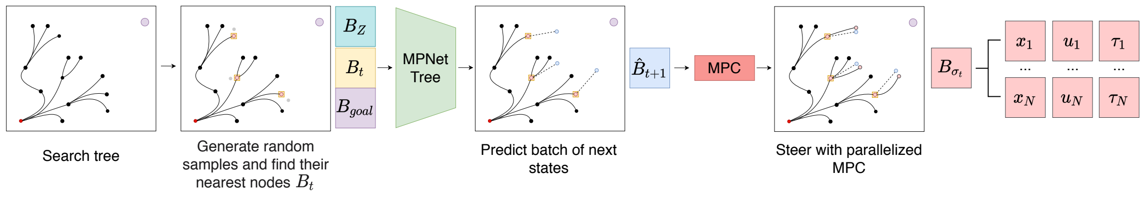

Search tree
Generate random samples and find their nearest nodes B_t
B_Z
B_t
B_{goal}
MPNet Tree
Predict batch of next states
$\hat{B}_{t+1}$
MPC
Steer with parallelized MPC
B_{σ_t}
x_1
u_1
τ_1
x_N
u_N
τ_N

Figure 3.16: MPC-MPNetTree The neural generator predicts a batch of next states from nearest neighbors of random states inside a search tree. The parallelized MPC finds the local controllers between them.

Neural Discriminator

Due to stochasticity in the neural generator, the predictions are usually scattered towards the given target. To select the best state from the given set, we introduce a neural discriminator network that predicts a given state's time-to-reach the desired target. Our planning procedures leverage the time-to-reach predictions as a cost to prune outputs of the neural generator.

Hence, the discriminator D, with parameters θ_d, takes environment embedding Z, the robot state x_t and x_{goal} as input, and predicts the cost as:

$$\hat{d}_t \leftarrow D(Z, x_t, x_{goal}; \theta_d) \tag{3.14}$$

The neural discriminator is trained to minimize the mean square error between the predicted costs and the real costs:

$$L_{D_{\theta_d}} = \frac{1}{N_p}\sum_{i=0}^{N}\frac{1}{T_i}\sum_{j=0}^{T_i-1}\|\hat{d}_{i,j+1} - (\sum_{k=j+1}^{T_i} d^*_{i,k})\|^2 \tag{3.15}$$

where $d^*_{i,k}$ is the cost to goal at waypoint k in the path i.

To balance the positive and negative training samples and enhance the discriminator's performance, we augment the training data by assigning large penalties to in-collision waypoints and unreachable transition pairs. As a result, the trained discriminator predicts high costs for invalid states and eliminates anomalous waypoints.

Model Predictive Control

To satisfy the kinodynamic constraints during tree/path expansion, we utilize MPC as a steering function, a strategy widely used for optimal control problems. We use MPC for several reasons, including implementation simplicity, lower computational complexity, and parallel computation potential. Furthermore, compared to BVP solvers, MPC models the trajectory generation process as an initial-value problem. Therefore, MPC does not collapse when the terminal state is unreachable and instead returns a nearest, valid terminal state. In contrast, BVP solvers fail in such situations because boundary conditions are not satisfied.

MPC takes a current x_t, a target state $\hat{x}_{t+1}$, and generates an optimized trajectory $\sigma_t = [(x,u,\tau)_t]$ minimizing the cost between the propagated terminal state and $\hat{x}_{t+1}$. Our MPC solver is implemented with Cross Entropy Method (CEM) [BKRE] and time-elastic-band approach to optimize both control and their duration sequences [RHB15]. CEM takes advantage of the Monte Carlo method and importance sampling to estimate the optimal trajectories distribution iteratively.

Algorithm 8 outlines our MPC approach. Using sampled controls and durations from a parameterized distribution $(u,\tau)_i \sim \mathcal{D}(u,\tau;\theta_{mpc})$, we propagate and generate the steering trajectory σ_i, from the given starting state x_t. These propagations are ranked through a cost function defined as $d_s = d(\sigma_i, \hat{x}_{t+1}) + d_c(\sigma_i)$, where $d(\cdot)$ is the distance metrics between given states and $d_c(\cdot)$ is a collision penalty function. The cost function selects the elite samples, i.e., the propagated states with the lowest scores. These elite states are used to update MPC parameters θ_{mpc} by minimizing the cross-entropy loss between the distributions of steered terminal states and the target state $\hat{x}_{t+1}$. Note that, the collision penalty discourages in-collision trajectory expansions.

Furthermore, in our implementation, we assume the distribution $\mathcal{D}(u,\tau;\theta_{mpc})$ of controls and their durations at each time step to be Gaussian distributions.

```
Algorithm 8: Model Predictive Control ($x_t, \hat{x}_{t+1}$)
    initialize parameters $\theta_{mpc}$;
    for $iter \leftarrow 1$ to $N_{iter}$ do
        for $u_i, \tau_i \sim \mathcal{D}(u,\tau;\theta_{mpc})$ do
            Propagate $x_t$ with $u_i, \tau_i$ to generate $\sigma_i$;
            Evaluate $d_{si} = d(\sigma_i, \hat{x}_{t+1}) + d_c(\sigma_i)$;
            Select elite samples based on their scores;
            Update $\theta_{mpc}$ with elite samples;
            Update optimal trajectory with the best $\sigma_i^*$
    return $\sigma_i^*$;
```

Parallelization

We use neural networks to process multiple waypoints as a batch to improve performance and implement our MPC algorithm using tensors on Graphic Processing Units (GPUs) for parallel processing. With parallel computation, the neural networks and MPC are accelerated to process up to $N_B \in \mathbb{N}$ samples simultaneously. To represent batch parallel processing, we introduce new notations denoting all vectorized inputs in batch form using the symbol B, i.e.,

$$B_t = \begin{bmatrix} x_t^1 \\ x_t^2 \\ \vdots \\ x_t^{N_B} \end{bmatrix}, B_{goal} = \begin{bmatrix} x_{goal} \\ x_{goal} \\ \vdots \\ x_{goal} \end{bmatrix}, B_Z = \begin{bmatrix} Z \\ Z \\ \vdots \\ Z \end{bmatrix}, \tag{3.16}$$

where B_t, B_{goal}, and B_Z correspond to the batch of current states, desired states, and observation encodings.

During execution, our stochastic generator outputs a variety of next states B_{t+1}, and the discriminator predicts their associated costs $B_{d_{t+1}}$ as:

$$\hat{B}_{t+1} \leftarrow G(B_Z, B_t, B_{goal}; \theta_g) \tag{3.17}$$

$$B_{d_{t+1}} \leftarrow D(B_Z, \hat{B}_{t+1}, B_{goal}; \theta_d) \tag{3.18}$$

For the given start B_t and target $\hat{B}_{t+1}$ states, our parallelized MPC generates their corresponding local kinodynamic trajectories as:

$$B_{\sigma_t} \leftarrow \text{MPC}(B_t, \hat{B}_{t+1}) \tag{3.19}$$

where $B_{\sigma_t} = [(x_i, u_i, \tau_i)_t]_{i=1}^{N_B}$ is a batch of local trajectories at time t.

```
Algorithm 9: MPC-MPNetPath ($Z, x_{init}, x_{goal}$)
    $T \leftarrow \{x_{init}\}$, $B_t \leftarrow x_{init}$;
    $B_Z \leftarrow Z$, $B_{goal} \leftarrow x_{goal}$;
    for $i \leftarrow 1$ to $n$ do
        $\hat{B}_{t+1} \leftarrow G(B_Z, B_t, B_{goal}; \theta_g)$;
        $\hat{x}_{t+1} \leftarrow_{\hat{x}_{t+1}} D(B_Z, \hat{B}_{t+1}, B_{goal}; \theta_d)$;
        $\sigma_t \leftarrow MPC(x_t, \hat{x}_{t+1})$;
        if Invalid($\sigma_t$) then
            $B_t \leftarrow$ randomNode($T$);
        else
            addToTree($\sigma_t, T$);
            set batch $B_t$ with a terminal state of $\sigma_t$;
        if Reached($T, x_{goal}$) then
            return ExtractPath($T$);
    return $\varnothing$;
```

Algorithm 10: MPC-MPNetTree (Z, x_{init}, x_{goal})

```
$T \leftarrow \{x_{init}\}$;
$B_Z \leftarrow Z,\ B_{goal} \leftarrow x_{goal}$;
for $i \leftarrow 1$ to $n$ do
    $B_{rand} \leftarrow$ RandomSample();
    $B_t \leftarrow$ NearestNeighbor($B_{rand}, T$);
    $\hat{B}_{t+1} \leftarrow G(B_Z, B_t, B_{goal}; \theta_g)$;
    $B_{\sigma_t} \leftarrow MPC(B_t, \hat{B}_{t+1})$;
    addToTree($B_{\sigma_t}, T$);
    if Reached($T, x_{goal}$) then
        return ExtractPath($T$);
return $\varnothing$;
```

Planning Algorithms

In this section, we present our planning algorithms that balance exploration-exploitation in their different ways for quickly finding a path solution with a unidirectional tree expansion.

MPC-MPNetPath: Figure 3.15 and Algorithm 9 outline our MPC-MPNetPath algorithm. The procedure begins by generating a batch of new samples using the neural generator G. The discriminator D prunes the generated batch B_t by selecting a sample $\hat{x}_{t+1}$ with a minimum cost $\hat{d}$ to reach the given target x_{goal}. The MPC module takes the current node x_t and selected next state $\hat{x}_{t+1}$ to perform the kinodynamic steering, leading to a local trajectory σ_t. The terminal state of σ_t is the resulting valid state x_{t+1} (as close as possible to $\hat{x}_{t+1}$ while satisfying constraints) after the execution of u on state x_t for time duration τ. The local trajectory is added to the tree if it is valid. In the case of invalid local trajectory, i.e., not collision-free, a random node is selected from the tree and is treated as new current state x_t for the next planning iteration. Once, the goal is reached, an end-to-end path is extracted connecting the given start and goal states under kinodynamical constraints.

MPC-MPNetTree: This method leverages (i) the innate capacity of neural networks to process batches, and (ii) our parallelized MPC framework to expand multiple nodes of the search tree simultaneously, directed towards the given target states B_{goal}. The algorithm is summarized in Figure 3.16 and Algorithm 10. In each iteration, MPC-MPNetTree samples a set of random states B_{rand} using the RandomSample function and finds their corresponding nearest neighbors in the tree by calling the NearestNeighbor function. These nearest nodes are treated as current set of states B_t for the underling MPC-MPNet procedures, i.e., the generator outputs the next batch of samples and MPC computes their local trajectories B_{σ_t}. The valid local trajectories are added to the search tree, and the final path is extracted once the tree reaches to the given goal state.

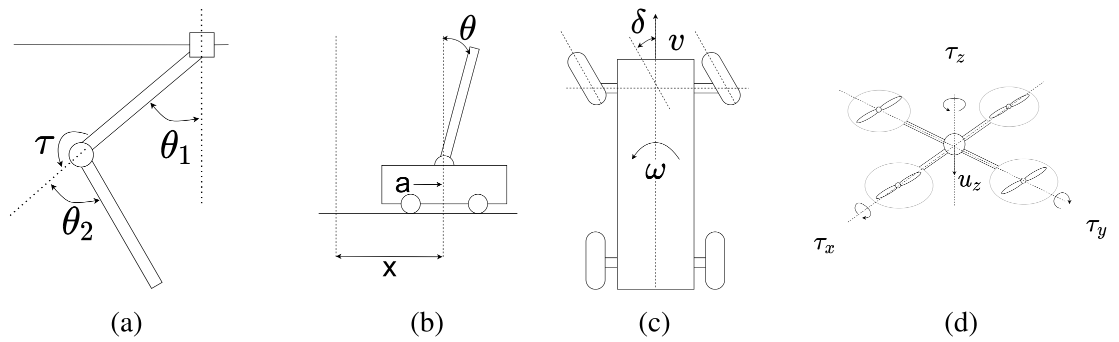


Figure 3.17: We consider the following robotic systems, (a) Acrobot, (b) Cartpole, (c) Car, and (d) Quadrotor, with complex dynamics for our cluttered, kinodynamically constrained environments.

3.2.3 Implementation Details

Table 3.4: The total mean computation times with standard deviations in seen test environments are presented for MPC-MPNetPath (MP-Path), MPC-MPNetTree (MP-Tree), and SST in various kinodynamic planning problems.

Methods	Planning Tasks			
	Acrobot	Cart-Pole	Quadrotor	Car-like
MP-Path	5.09 ± 6.87	4.74 ± 7.37	0.46 ± 2.12	8.96 ± 12.33
MP-Tree	4.19 ± 6.03	4.43 ± 6.07	2.26 ± 3.26	7.51 ± 8.72
SST	28.32 ± 20.53	14.99 ± 14.29	143.69 ± 143.43	41.69 ± 70.31

This section describes the necessary implementation details of our frameworks with their training and testing environments. We implement our neural modules using Pytorch and export

Table 3.5: The total mean computation times with standard deviations in unseen test environments are presented for MPC-MPNetPath (MP-Path), MPC-MPNetTree (MP-Tree), and SST in various kinodynamic planning problems.

Methods	Planning Tasks			
	Acrobot	Cart-Pole	Quadrotor	Car-like
MP-Path	9.88 ± 8.70	7.93 ± 9.13	0.41 ± 1.12	19.10 ± 21.25
MP-Tree	6.13 ± 7.90	4.36 ± 5.55	1.78 ± 2.45	8.40 ± 11.48
SST	21.37 ± 23.02	12.21 ± 10.11	251.03 ± 197.83	28.16 ± 40.00

them to C++ with Torchscript. Our parallelized MPC algorithm follows standard GPU programming. For the training and testing environments, we consider the following kinodynamically constrained, cluttered environments with schematics shown in Fig. 3.17.

Acrobot

We use the acrobot dynamics as specified in [Spo98]. The state space is defined as $[\theta_1, \theta_2.\dot{\theta_1}, \dot{\theta_2}] \in [-\pi, \pi]^2 \times [-6, 6]^2$. The control space is defined as $[-4, 4]$. We generate four rectangular obstacles and restrict the center of the obstacles to lie inside the annulus that affects the acrobot movement.

Cartpole

The cartpole dynamics are used as specified in [PKP14]. The state space is defined as: $[x, \dot{x}, \theta, \dot{\theta}] \in [-30, 30] \times [-40, 40] \times [-\pi, \pi] \times [-2, 2]$. and the control space is defined as $[-300, 300]$. We randomly place seven rectangular obstacles in the environment to challenge and restrict the cartpole motion.

Car

This is a 2D first-order car system, where the state space is of 3DOF, including position and orientation. The control inputs are the position velocity and angular velocity. The state space is defined as $[x, y, \theta] \in [-25, 25] \times [-35, 35] \times [-\pi, \pi]$ and control space bound as $[0, 2] \times [-0.5, 0.5]$.

We randomly place five rectangular obstacles in the workspace, and limit the distance between obstacles to ensure narrow passages.

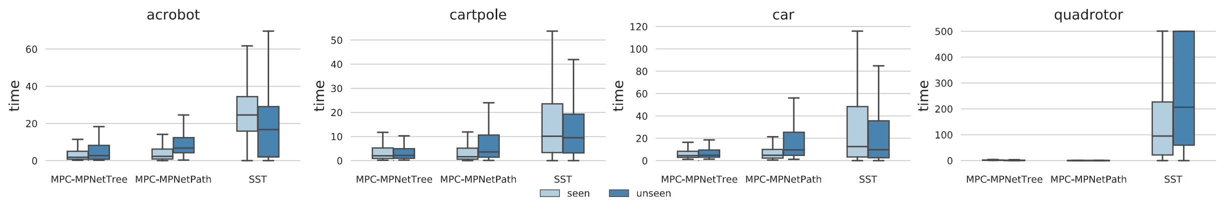
Quadrotor

We define the quadrotor dynamics as in [AOJJ13]. The state space is defined as: $[p, q, \dot{p}, \omega]$, where p and q denote the position and orientation of the quadrotor, respectively, with their corresponding time derivatives, indicating velocity, represented as $\dot{p}$ and ω. The control space is 4 dimensional with bound $[-15, -5] \times [-1, 1]^3$. We randomly place 10 obstacles in the workspace space and ensure the scene is cluttered.

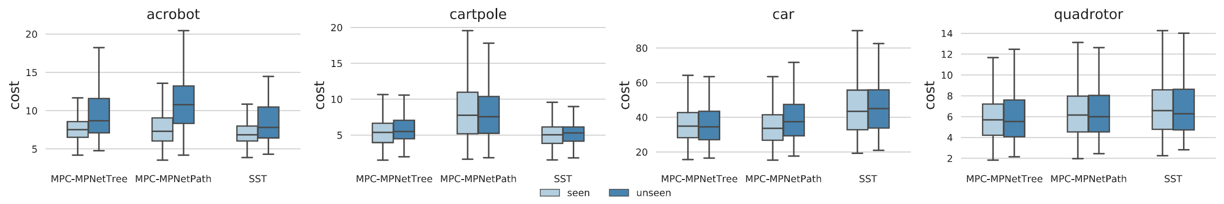
In each of the cases mentioned above, we set up 10 environments by random placement of the obstacles, each with 1000-2000 randomly sampled start and goal state pairs. The $10-20\%$ of data is used for testing, and the remaining for the training. In our test dataset, we also include two unseen environments for each problem, created by random placement of obstacles. We ensure these environments to be different from the ten seen settings, and for each, we randomly sample 100-200 valid start and goal pairs for evaluation. Hence, in total, our test dataset contains 12 environments for the given setups. The demonstration trajectories for the training data are obtained using the SST algorithm. Furthermore, we obtain the point-cloud of the environments, by randomly sampling the obstacle space and processing them into voxel v of size $32 \times 32 \times 32$.

3.2.4 Results

We present a set of experiments to compare the computation time, path quality, and success rate of MPC-MPNetPath, MPC-MPNetTree, and SST planning algorithms. All experiments were performed on the same system with 32GB RAM, GeForce GTX 1080 GPU, and 3.40GHz$\times$8 Intel Core i7 processor.



(a) Computational time (in seconds) comparison for Acrobot, Cart-pole, Car and Quadrotor in seen and unseen environments



(b) Path cost comparison for Acrobot, Cart-pole, Car and Quadrotor in seen and unseen environments

Figure 3.18: The interquartile ranges of computation times and path qualities (time-to-reach the target) for MPC-MPNetPath, MPC-MPNetTree, and SST in Acrobot, Cart-pole, Car and Quadrotor environments.

Comparative Studies

This study compares the computation time, success rate, and path quality (measured by time-to-reach) of MPC-MPNetPath, MPC-MPNetTree, and SST algorithms in Cart-Pole, Acrobot, Car, and Quadrotor environments. All evaluation planning problems were unique and not seen during the training, presenting non-trivial and cluttered environments with underactuated systems.

Table 3.4-3.5 presents the mean computation times with standard deviations across different scenarios in both seen and unseen environments. Figs. 3.18 show the box-plots of all methods comparing their computation time and path quality inter-quartile ranges for solving all kinodynamic planning problems. Fig. 3.19 to Fig. 3.22 show example qualitative results from MPC-MPNet and SST. In all these problems, including unseen scenarios, MPC-MPNetTree and MPC-MPNetPath success rates were between $90-100\%$ and $85-95\%$, respectively, comparable to the SST success rates in the given time limit.

It can be seen that compared to SST, MPC-MPNet methods take significantly lower computation times to find similar quality path solutions with comparable success rates. Our experiments also show that for environments with high-dimensional state and control spaces such

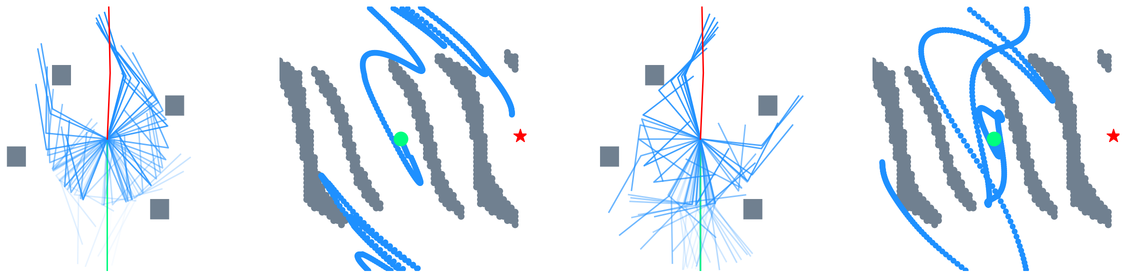

(a) MPC-MPNet: $t = 5.5s$, $c = 10.9$ (b) SST: $t = 52.4s$, $c = 11.2$

Figure 3.19: Acrobot environment: The workspace (left) and state-space (right) trajectories are shown in each subfigure. In this example, the start state is $[0,0,0,0]$, and goal states are randomly distributed around the vertical configuration.

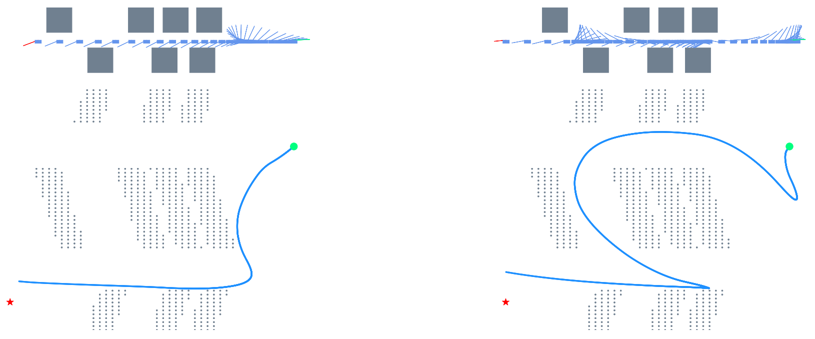
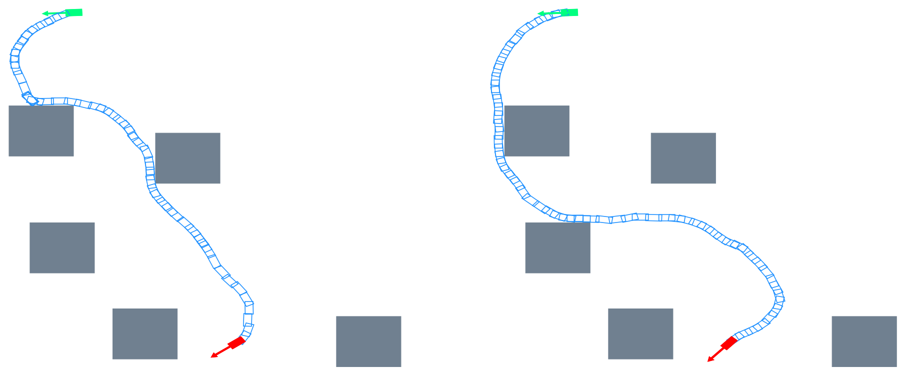

(a) MPC-MPNet: $t = 2.8s$, $c = 4.2$ (b) SST: $t = 29.4s$, $c = 7.9$

Figure 3.20: Cart-Pole environment: The workspace (left) and state-space (right) trajectories are shown in each subfigure.

as in Quadrotor, SST's computation times increase exponentially. In contrast, MPC-MPNet still retains its computational benefits from informative waypoint sampling and outperforms SST by a large margin.

Among MPC-MPNetTree and MPC-MPNetPath, we observed that the former method achieves higher success rates and finds better quality path solutions by exploiting GPU-accelerated, parallel computation. Nevertheless, we present CPU-based MPC-MPNetPath and GPU-based MPC-MPNetTree to highlight that both approaches perform better than traditional gold-standard planning methods. Moreover, our algorithms balance exploration and exploitation through randomly sampling current configurations within trees to provide better completeness and comparable success rates as conventional methods.



(a) Ours: $t = 12.3s$, $c = 57.0$ (b) SST: $t = 27.2s$, $c = 63.8$

Figure 3.21: Car environment: MPC-MPNet and SST finding paths under kinodynamic constraints for a non-holonomic system in an example environment with multiple narrow passages.

Table 3.6: Ablation Study: The total mean computation time with and with out neural discriminator is shown for MPC-MPNetPath, where the path quality, measured by the time-to-reach, is presented in parentheses.

Setup	Planning Tasks			
	Acrobot	Cart-Pole	Quadrotor	Car-like
w/o D	$8.14 \pm 10.37 (7.23 \pm 4.91)$	$5.43 \pm 8.59 (7.67 \pm 3.37)$	$0.81 \pm 4.11 (8.45 \pm 4.11)$	$13.29 \pm 13.36 (47.72 \pm 37.36)$
w/ D	$5.09 \pm 6.87 (5.18 \pm 3.82)$	$4.47 \pm 7.37 (6.25 \pm 3.13)$	$0.46 \pm 2.12 (6.49 \pm 2.71)$	$8.96 \pm 12.33 (47.68 \pm 20.44)$

Ablation Studies

To show the impact of the neural discriminator in the planning pipeline, we present an ablation study, comparing the planning time and path quality between (i) MPC-MPNetPath without neural discriminator, which predicts only one state at every iteration, and (ii) MPC-MPnetPath, which generates a batch of waypoints with the neural generator, and selects the best based on an estimated cost by the neural discriminator. All experiments are conducted in the same environmental setup as in the comparative studies.

From the results shown in Table 3.6, we can observe that the neural discriminator contributes to reducing the mean and standard deviation of planning time. It also helps with generating trajectories with a better cost quality. In MPC-MPNet, stochasticity is introduced by dropout

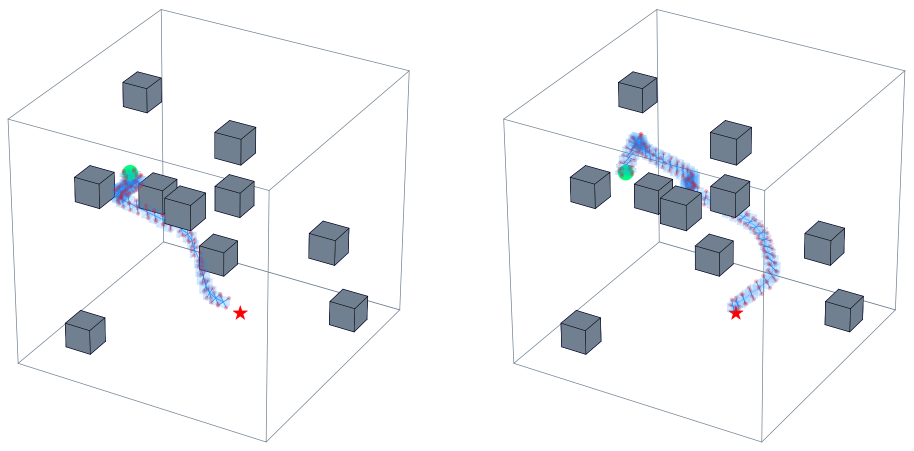

(a) Ours: $t = 1.5s$ $c = 7.6$ (b) SST: $t = 162.3s$ $c = 9.9$

Figure 3.22: Quadrotor environment: The problem requires finding a kinodynamically constrained motion of a 12 DOF quadrotor in challenging environments. In these scenarios, our methods were at least 50 times faster than SST.

layers in the neural generator. This operation generates a variety of samples, some of which might require replanning for the connections. Our process eliminates those cases using the neural discriminator by selecting a state with the minimum time to reach the given target, also resulting in fewer planning iterations and better path quality.

3.2.5 Discussion

In this section, we highlight the worst-case properties of our proposed algorithms named MPC-MPNetPath and MPC-MPNetTree. The former operates MPC without GPU processing and uses discriminator to remove unnecessary states to save computational resources. The latter builds on GPU programming and parallelly computes various nodes for simultaneous local extensions with MPC. Our methods plan without relying on bidirectional tree generation or any replanning as was required in the original MPNet and its variants. However, similar to MPNet, our neural generator does explore the sub-space of given state-space that potentially contains a path solution, thanks to Dropout, that implants randomness into our neural generator adapted from expert demonstrations. Since our generated state-spaces are confined to a subspace, the resulting informed tree also grows in that region due to directed extensions with MPC. To enable our methods to explore the state-space outside the generator's learned state-spaces, we propose a notion of stage-wise exploration.

Our stage-wise exploration strategy balances global exploitation-exploration in three phases. In the first phase, our proposed algorithms are operated for a fixed number of planning iterations N_1. In the second phase, i.e., after N_1 iterations and until N_2 iterations, our algorithm replaces the MPC-based local controller with a random shooting method. This phase allows exploration in the action-spaces while still keeping state-spaces informed as given by the neural generator. Finally, after N_2 steps and beyond in the third phase, our approach performs full exploration by randomly sampling both state and action spaces, like the SST algorithm. These three phases allow our trees to expand from an inner region, potentially containing a path solution, to an outer region in the worst-case for finding a solution if one exists. In our experimentation, we validated that, similar to MPNet, incorporating staged-wise exploration ensures 100% success rate while still retaining the computational benefits and performing better than classical planning approaches.

Since our methods perform stage-wise exploration of the state and action spaces and eventually explore them entirely over a large number of planning iterations, the resulting approach exhibits *probabilistic-completeness*. It implies that MPC-MPNet finds a path, if one exists, with the probability approaching to one as the number of planning iterations reaches infinity. The formal proofs can be derived in the same way as reported in [LLB16]. Furthermore, note that our MPC-MPNetTree method also uses the nearest neighbor search for extending trees. This method is adopted from SST, which selects the best neighbors in terms of their cost from the given start/root state and removes tree edges with relatively higher costs. Based on this nearest-neighbor selector and the random exploration, the SST method [LLB16] also guarantees *asymptotic-optimality*, i.e., over a large number of planning iterations, their planner will eventually find a minimum cost path

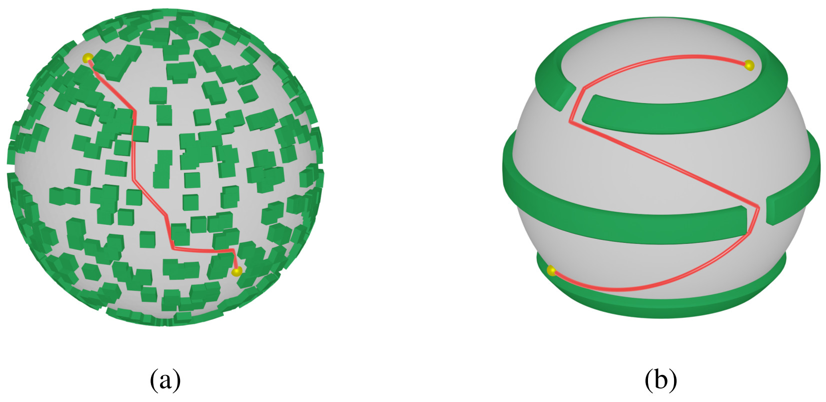


Figure 3.23: CoMPNetX generalized in sphere environment from (a) small cubical obstacles' geometry to (b) multiple longitudinal obstacle strips and planned near-optimal paths in sub-second computational times.

solution, if one exists. Since MPC-MPNetTree also does exploration, though in stages, and uses SST's like nearest node selector and pruner, it also exhibits *asymptotic-optimality* with proofs being similar to as presented in [LLB16].

3.3 Kinematic Manifold Constraints

Constrained Motion Planning (CMP) has a broad range of robotics applications for solving practical problems emerging in domains such as assistance at home, factory floors, disaster sites, and hospitals [CHL$^+$05]. In our daily life, most of our activities involve a large number of CMP tasks. For example, at our home, we interact with various objects to perform usual household chores such as cleaning and cooking, including opening doors, carrying a tray or a glass filled with water, and lifting boxes. Likewise, skilled workers manipulate their tools to solve a wide variety of tasks such as assembly at factory floors and advanced-level surgery in the hospitals.

In the last decade, Sampling-based Motion Planning (SMP) methods have surfaced as prominent motion planning tools in robotics [LaV06]. These algorithms randomly sample the robot joint-configurations to build a collision-free graph, which eventually connects the given start and goal configurations leading to a path solution [LaV06]. However, in CMP, the constraint equations implicitly define a configuration space comprising zero-volume constraint manifolds embedded in a higher-dimensional ambient space of the robot's joint variables [JP13b]. Therefore, the probability of generating random robot configurations on those manifolds is not just low but zero, which makes the state-of-the-art gold standard SMP methods [KL00, KF11, GSB14] [GSB15, QA15, JSCP15, QA16, TQAN18] fail in such problems [KMK18].

Recently, constraint-adherence methods that generate samples on the manifolds have been incorporated into existing SMP algorithms for CMP [KMK18]. These methods include projection and continuation-based approaches. The former uses Jacobian-based gradient descent to project a given configuration to the manifold. The latter takes a known constraint-adhering configuration to compute a tangent space using which new samples are generated closer to the manifold for projection. These advanced planning methods solve a wide range of tasks, but they often exhibit high computational time complexity with high variance, making them frequently impractical for real-world manipulation problems.

To overcome limitations of classical planners in CMP, we extended MPNet to solve CMP problems and proposed Constrained Motion Planning Networks (CoMPNet) [QDCY20]. CoMPNet is a deep neural network-based approach that takes the environment perception information, text-based task specification defining the constraints (e.g., *open the door*), and robot's start and goal configurations as an input and outputs a feasible path on the constraint manifolds. CoMPNet connects any two given configurations using a projection-based constraint-adherence operator, and like MPNet, it also performs a divide-and-conquer through bidirectional expansion. However, it avoids replanning, which is a computationally expensive process in CMP, and instead builds an informed tree of possible paths.

We further extend CoMPNet and presents a unified framework called Constrained Motion Planning Networks X (CoMPNetX)[3] [QDBY21], which extends CoMPNet and generates informed implicit manifold configurations to speed-up any SMP algorithm equipped with their

[3]The project videos and other supplementary material are available at https://sites.google.com/view/compnetx/home

constraint-adherence approach for solving CMP problems. CoMPNetX comprises the conditional neural generator, discriminator, a neural gradient-based projection operator, and sampling heuristics to propose samples for all kinds of SMP methods. Furthermore, compared to our previously proposed CoMPNet, this new approach, i.e., CoMPNetX, has the following novel features:

- CoMPNetX plans in implicit manifold configuration spaces, whereas CoMPNet only considers the robot configuration space. The implicit manifold configuration spaces are formed by the robot configuration and the constraint function. For instance, in the door opening task, the door, represented as a virtual-link manipulator using Task Space Regions (TSRs), and the robot arm forms an implicit manifold planning space for CoMPNetX.

- CoMPNet only considers the projection operator for constraint adherence. In contrast, in this work, we extend CoMPNet, naming it CoMPNetX, to operate with both projection- and continuation-based constraint adherence approaches for enhancing any SMP method, including batch and bidirectional techniques.

- In our previous work, the task sequences were defined by an expert as a text, e.g., open the cabinet and then move an object into the cabinet. CoMPNet sequentially takes the latent embeddings of those text-based task specifications to generate the motion sequences. However, text-based representations are agnostic of the given workspace and the overall planning objective. Therefore, this work introduces a strategy to combine CoMPNetX with the deep neural network-based task planning approaches that relieve an expert from providing task sequences during execution and provide context-aware neural task representations for CMP.

- Unlike CoMPNet, the proposed approach also comprises a discriminator function that predicts the distances of generated configurations from the constraint manifold and provides gradients to project them to the manifold if needed.

In summary, CoMPNetX can generate robot configurations for a wide range of SMP algorithms while retaining their worst-case theoretical guarantees. Our generator and discriminator are conditioned on the neural task representation and the environment observation encoding. The conditional generator takes the desired start and goal configurations to output intermediate implicit manifold configurations, and the conditional discriminator predicts their geodesic distances from the underlying manifold. We use the discriminator's predictions and their gradients as the operator to project the given configurations towards the constraint manifold if needed. CoMPNetX naturally forms a mutual symbiotic relationship with learning-based task programmers and exploits their inner states, representing tasks, to transverse multiple constrained manifolds for finding their path solutions. We show that these task representations from a learning-based task planner can lead to better performance in motion planning than human-defined text-based task representations (as in [QDCY20]). We test CoMPNetX with various SMP algorithms using both continuation and projection-based constraint-adherence methods on challenging problems and benchmark them against the state-of-the-art classical CMP algorithms. We also evaluate our models' generalization capacity to new planning problems and environment structures, such as in the sphere environment from being trained on settings with small obstacle blocks and generalizing to the environment with multiple obstacle strips forming various narrow passages (Fig. 3.23).

3.3.1 Preliminaries

In this section, we describe the problem of constrained motion planning with its basic terminologies. We also outline a brief overview of constrained-adherence operators employed by CMP methods for local planning under hard kinematic constraints.

Problem Definition

In the classical problem of motion planning, the robot system is defined by a configuration space (C-space) $Q \in \mathbb{R}^n$ with $n \in \mathbb{N}$ dimensions. The axis of C-space corresponds to the system's variables that govern their motion, such as robot joint-angles, and hence, the dimension n is equivalent to the robot's degree-of-freedoms (DOF).

The robot's surrounding environment is usually described as task-space $\mathcal{X} \in \mathbb{R}^m$ with $m \in \mathbb{N}$ dimensions, comprising obstacle $\mathcal{X}_{obs} \subset \mathcal{X}$ and obstacle-free $\mathcal{X}_{free} = \mathcal{X} \backslash \mathcal{X}_{obs}$ spaces. In the C-space terminology, the spaces $\mathcal{X}_{obs}$ and $\mathcal{X}_{free}$ are represented as $\mathcal{Q}_{obs}$ and $\mathcal{Q}_{free} = \mathcal{Q} \backslash \mathcal{Q}_{obs}$, respectively. In motion planning, a collision-checker InCollision$(\cdot)$ is assumed to be available that takes a robot configuration $q \in \mathcal{Q}$ and $\mathcal{X}_{obs}$, and outputs a boolean indicating if a given configuration lies in $\mathcal{Q}_{obs}$ or not.

We consider a setup where for a given current $x_t \in \mathcal{X}_{free}$ and target $x_T \in \mathcal{X}_{free}$ workspace observations, the high-level task planner, π_H, at time t, outputs an achievable sub-task representation Z_c for the low-level agent π_L. For each subtask, Z_c, we also assume there exist a constraint function $\mathbf{F}$. The agent, π_L, finds motion sequences in $\mathcal{Q}_{free}$ to achieve the given subtask, Z_c, under constraints $\mathbf{F}$, leading to a next observation x_{t+1}. This work considers deep neural networks-based state-of-the-art task planners as high-level agents, π_H, and proposes a novel low-level agent, π_L, i.e., CoMPNetX, that leverages $\{Z_c, \mathbf{F}\}$ for motion planning under task-specific constraints.

A fundamental unconstrained motion planning problem for a given start configuration $q_{init} \in \mathcal{Q}_{free}$, a goal region $\mathcal{Q}_{goal} \subset \mathcal{Q}_{free}$, environment obstacles $\mathcal{X}_{obs}$, and a collision-checker, is defined as:

Problem 1 (Unconstrained Motion Planning): Given a planning problem $\{q_{init}, \mathcal{Q}_{goal}, \mathcal{X}_{obs}\}$*, and a collision-checker, find a collision-free path solution* $\sigma : [0,1]$*, if one exists, such that* $\sigma_0 = q_{init}$*,* $\sigma_1 \in \mathcal{Q}_{goal}$*, and* $\sigma[0,1] \mapsto \mathcal{Q}_{free}$*.*

In the constrained motion planning, a planner also has to satisfy a set of hard constraints defined by a function $\mathbf{F}(q) : \mathcal{Q} \mapsto \mathbb{R}^k$, such that $\mathbf{F}(q) = 0$. The $k \in \mathbb{N}$ denotes the number of constraints imposed on the robot motion, which induces an $(n-k)$-dimensional space embedded in the robot's unconstrained ambient C-space, comprising one or more manifolds $\mathcal{M}$, i.e,

$$\mathcal{M} = \{q \in \mathcal{Q} \,|\, \mathbf{F}(q) = 0\}$$

In practice, a configuration q is assumed to be on the manifold if $\|\mathbf{F}(q)\|_2 < \varepsilon$, where $\varepsilon > 0$ is a tolerance threshold. Furthermore, the obstacle and obstacle-free spaces on the manifolds are denoted as $\mathcal{M}_{free} = \mathcal{M} \cap \mathcal{Q}_{free}$ and $\mathcal{M}_{obs} = \mathcal{M} \backslash \mathcal{M}_{free}$, respectively. A CMP problem for a given start q_{init} configuration, goal region $\mathcal{Q}_{goal} \subset \mathcal{Q}_{free}$, environment obstacles $\mathcal{X}_{obs}$, function $\mathbf{F}$, and a collision-checker, is defined as:

Problem 2 (Constrained Motion Planning): Given a planning problem $\{q_{init}, \mathcal{Q}_{goal}, \mathcal{X}_{obs}, \mathbf{F}\}$*, and a collision-checker, find a collision-free path solution* $\sigma : [0,1]$*, if one exists, such that* $\sigma_0 = q_{init}$*,* $\sigma_1 \in \mathcal{Q}_{goal}$*, and* $\sigma[0,1] \mapsto \mathcal{M}_{free}$*.*

In our work, we show that CoMPNetX solves both unconstrained (Problem 1) and constrained (Problem 2) planning problems. Furthermore, for the latter problem, we only consider kinematic constraints, i.e., the function $\mathbf{F}$ solely depends on robot configuration $q \in \mathcal{Q}$, not on other robot properties such as dynamics representing velocity or acceleration. Moreover, we define $\mathbf{F}(q)$ as distance to the constraint manifold with domain s, i.e.,

$$\mathbf{F}(q) = \text{Distance to the constraint manifold}$$

For instance, if the constraint is on the robot's end-effector to maintain a particular position, then $\mathbf{F}(q)$ can be defined as the distance of the robot's end-effector to that specific position with domain $s \in [0,1]$, spanning an entire or a fraction of a motion trajectory. Likewise, when the robot is moving, balancing constraints are usually imposed on the whole robot motion trajectory with $s = [0,1]$.

Algorithm 11: Projection Operator: Proj (q)

```
for $i \leftarrow 0$ to $N$ do
    $\Delta x \leftarrow \mathbf{F}(q)$
    if $\|\Delta x\|_2 < \varepsilon$ then
        return $q$
    else
        $q \leftarrow q - \mathbf{J}(q)^{+}\Delta x$
```

In the remaining section, we describe the two main types of classical constraint-adherence operators that ensure a given configuration or a motion between two configurations lies on the constraint manifold defined by $\mathbf{F}$.

Projection-based Constraint-Adherance Operator

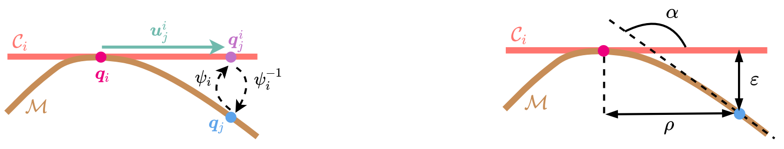
The projection operator (Proj) maps a given configuration $q \in \mathbb{R}^n$ to the manifold $\mathcal{M}$. It can be formulated as a constraint optimization problem [KMK19]

$$\min_{q'} \frac{1}{2}\|q - q'\|^2 \text{ subject to } \mathbf{F}(q') = 0,$$

with its dual as:

$$L(q', \lambda) = \frac{1}{2}\|q - q'\|^2 - \lambda \mathbf{F}(q'),$$

where λ corresponds to Lagrange multipliers. The above system is solved using gradient descent as summarized in Algorithm 11, where $\mathbf{J}^{+}(q)$ is the pseudoinverse of the Jacobian at configuration $q \in Q$. Algorithm 12 outlines the local planning procedure using a projection operator [KMK19, BSK11]. This procedure outputs all the intermediate configurations on the manifold in the given conditions and loop limit N, when transversing from a given start configuration (q_s) towards the end configuration (q_e) in small incremental steps $\gamma \in \mathbb{R}$. The projection-based steering stops if any of the following happens: (i) The loop limit is reached. (ii) The resulting configuration



(a) (b)

Figure 3.24: (a) A chart $\mathcal{C}_i$ operators comprising exponential ψ_i and logrithmic ψ_i^{-1} functions for mapping between the tangent space at q_i and the manifold. (b) The parameters defining the chart validity region.

q_{i+1} is in a collision. (iii) The stepping distance is diverging rather than converging to prevent overshooting the target configuration, i.e., either $d_2 > d_1$ or $d > \lambda_1\gamma$. (v) The progress in manifold space D becomes greater than a scalar λ_2 times the progress in the ambient space $d_w = \|q_e - q_s\|$.

```
Algorithm 12: Projection Integrator ($q_s, q_e$)
    $i \leftarrow 0$; $D \leftarrow 0$
    $d_w \leftarrow \|q_e - q_s\|$; $q_i \leftarrow q_s$
    while $i < N$ do
        $q_{i+1} \leftarrow \text{Proj}(q_i + \gamma(q_e - q_i))$
        $d \leftarrow \|q_{i+1} - q_i\|_2$
        $D \leftarrow D + d$
        $d_1 \leftarrow \|q_i - q_e\|_2$; $d_2 \leftarrow \|q_{i+1} - q_e\|_2$
        if InCollision($q_{i+1}$) or $d_2 > d_1$ or $d > \lambda_1\gamma$ or $D > \lambda_2 d_w$ then
            break
    $i \leftarrow i + 1$
    return $\{q_j\}_{j=0}^{i}$
```

Continuation-based Constraint-Adherence Operator

The continuation-based approaches [KMK19, JP17, KUSP16] represent the manifold through a set of local parameterizations, known as charts $\mathcal{C}$, forming an *atlas* $\mathcal{A}$.

A chart $\mathcal{C}_i = (q_i, \Phi_i(q_i))$, with an index $i \in \mathbb{N}$, locally parameterizes a manifold through a tangent space and its orthonormal basis Φ_i at a known constraint-adhering configuration $q_i \in \mathcal{M}$. The orthonormal basis $\Phi_i \in \mathbb{R}^{(n-k)\times n}$ is used to define an exponential map $\psi_i : \mathbb{R}^k \mapsto \mathbb{R}^n$ and its inverse, i.e., a logarithmic map $\psi_i^{-1} : \mathbb{R}^n \mapsto \mathbb{R}^k$, between the parameter u_j^i on the tangent space

Algorithm 13: Atlas Integrator $(q_s, q_e, \mathcal{A}_{\mathcal{M}})$

```
$i \leftarrow 0; D \leftarrow 0$
$d_w \leftarrow \|q_e - q_s\|$
$q_i \leftarrow q_s$
$\mathcal{C}_i \leftarrow$ GetChart$(q_i, \mathcal{A}_{\mathcal{M}})$
$u_i \leftarrow \psi_i^{-1}(q_i)$
$u_e \leftarrow \psi_i^{-1}(q_e)$
**while** $\|u_i - u_e\|_2 > \gamma$ **do**
    $u_{i+1} \leftarrow u_i + \gamma(u_e - u_i)/\|u_e - u_i\|_2$
    $q_{i+1} \leftarrow \psi_i(u_{i+1})$
    $d \leftarrow \|q_{i+1} - q_i\|_2$
    $D \leftarrow D + d$
    $d_1 \leftarrow \|q_i - q_e\|_2; d_2 \leftarrow \|q_{i+1} - q_e\|_2$
    **if** InCollision$(q_{i+1})$ or $d_2 > d_1$ or $d > \lambda_1\gamma$ or $d < \varepsilon$ or $D > \lambda_2 d_w$ or $i > N$ **then**
        break
    $i \leftarrow i + 1$
    **if** not RegionValidity$(u_i, q_i)$ or $u_i \notin \mathcal{P}_{i-1}$ **then**
        $\mathcal{C}_i \leftarrow$ GetChart$(q_i, \mathcal{A}_{\mathcal{M}})$
        $u_i \leftarrow \psi_i^{-1}(q_i)$
        $u_e \leftarrow \psi_i^{-1}(q_e)$
**return** $\{q_j\}_{j=0}^{i}$
```

and the manifold around configuration q_i (Fig. 3.24 (a)). The basis $\Phi_i \in \mathbb{R}^{n\times k}$ is computed by solving a following system of equations:

$$\begin{pmatrix} \mathbf{J}(q_i) \\ \Phi_i^\top \end{pmatrix} \Phi_i^\top = \begin{pmatrix} \mathbf{0} \\ \mathbf{I} \end{pmatrix}, \tag{3.20}$$

where $\mathbf{J}(q_i) \in \mathbb{R}^{k\times n}$ is the Jacobian of $\mathbf{F}$ at the configuration q_i, $\mathbf{0} \in \mathbb{R}^{k\times k}$, and $\mathbf{I} \in \mathbb{R}^{k\times k}$ is the identity matrix.

The exponential mapping ψ_i is a two step process. The first step determines a configuration q_j^i in the ambient space using the mapping ϕ_i, i.e.,

$$q_j^i = \phi_i(u_j^i) = q_i + \Phi_i u_j^i \tag{3.21}$$

The second step takes the q_j^i and orthogonally projects it to the manifold resulting in q_j, by solving the following system:

$$\left.\begin{aligned} \mathbf{F}(q_j) = \mathbf{0} \\ \Phi_i^\top (q_j - q_j^i) = \mathbf{0} \end{aligned}\right\} \tag{3.22}$$

The above equations are usually solved iteratively by a Newton method until the error $\|(q_j - q_j^i)\|_2 < \varepsilon$ is tolerable or the maximum iteration limit is reached.

The inverse logarithmic mapping ψ_i^{-1} from the manifold to the tangent space is straightforward to compute, i.e.,

$$u_j^i = \psi_i^{-1}(q_j) = \Phi_i^\top (q_j - q_i) \tag{3.23}$$

Note that each chart $\mathcal{C}_i$ has a validity region $\mathcal{V}_i$ in which it properly parameterizes the manifold and exceeding that region could lead to divergence when orthogonaly projecting configurations to the manifold during the exponential mapping process. This validity region is governed by the following conditions:

$$\|q_j^i - q_j\| \leq \varepsilon; \; \frac{\|u_j^i\|_2}{\|q_i - q_j\|} < \cos(\alpha); \; \|u_j^i\| \leq \rho \tag{3.24}$$

where ε and α indicate the maximum allowable distance and curvature, respectively, between the chart $\mathcal{C}_i$ and the underlying manifold $\mathcal{M}$, and ρ defines the radius of sphere around q_i (Fig. 3.24 (b)). Furthermore, the validity region $\mathcal{V}_i$ can have a complex shape and is usually approximated by a convex polytope $\mathcal{P}_i \subset \mathcal{V}_i$, represented as a set of linear inequalities defined in a tangent space of chart $\mathcal{C}_i$.

To realize the local planning using continuation operator, there exist two types of methods naming *atlas* integrator (Algorithm 13) and *tangent bundle* integrator (Algorithm 14). The latter, in contrast to the former, is less strict about the intermediate configurations being on the manifold and performs projections only when needed and does not separate the tangent spaces into half-spaces to prevent overlaps. In our implementations, these integrators assume both start (q_s) and end (q_e) configurations to be on the manifold. The procedure RegionValididty in the atlas integrator returns False if any of the above-mentioned region validity conditions are violated.

Algorithm 14: Tangent Bundle Integrator ($q_s, q_e, \mathcal{A}_\mathcal{M}$)

```
$i \leftarrow 0; D \leftarrow 0$
$d_w \leftarrow \|q_e - q_s\|$
$q_i \leftarrow q_s$
$\mathcal{C}_i \leftarrow$ GetChart$(q_i, \mathcal{A}_\mathcal{M})$
$u_i \leftarrow \psi_i^{-1}(q_i)$
$u_e \leftarrow \psi_i^{-1}(q_e)$
while $\|u_i - u_e\|_2 > \gamma$ do
    $u_{i+1} \leftarrow u_i + \gamma(u_e - u_i)/\|u_e - u_i\|_2$
    $q_{i+1} \leftarrow \phi_i(u_{i+1})$
    $d \leftarrow \|q_{i+1} - q_i\|_2$
    $D \leftarrow D + d$
    $d_1 \leftarrow \|q_i - q_e\|_2; d_2 \leftarrow \|q_{i+1} - q_e\|_2$
    if InCollision$(q_{i+1})$ or $d_2 > d_1$ or $d > \lambda_1\gamma$ or $d < \varepsilon$ or $D > \lambda_2 d_w$ or $i > N$ then
        break
    $i \leftarrow i + 1$
    if $\|\phi_{i-1}(u_i) - q_i\|_2 > \varepsilon$ or $u_i \notin \mathcal{P}_{i-1}$ then
        $q_i \leftarrow \psi_{i-1}(u_i)$
        $\mathcal{C}_i \leftarrow$ GetChart$(q_i, \mathcal{A}_\mathcal{M})$
        $u_i \leftarrow \psi_i^{-1}(q_i)$
        $u_e \leftarrow \psi_i^{-1}(q_e)$
return $\{q_j\}_{j=0}^{i}$
```

3.3.2 Related Work

In this section, we present the existing methods that address the problem of CMP, ranging from relaxation-based methods for trajectory optimization and control to strict approaches such as projection and continuation for sampling-based planning algorithms.

The relaxation-based methods represent the hard-constraints as soft-constraints by incorporating them as a penalty into the cost function. The cost function is optimized to get the desired robot behavior. For instance, the IK-based reactive control method [ABB$^+$15, JSB$^+$15] used at the DARPA Robotics Challenge operates in the workspace and finds constrained robot motion through convex optimization of the given cost function. However, these approaches often provide incomplete solutions as they are susceptible to local minima. The trajectory optimization methods [RZBS09, SDH$^+$14] also optimize the given cost function over the entire trajectory to find a feasible motion plan. However, due to the relaxation, they weakly satisfy the given constraints and are typically only effective on short-horizon problems. Recently, Bonalli et al. [BCB$^+$19] proposed a trajectory optimization method for implicitly-defined constraint manifolds, but their approach is yet to be explored and analyzed in practical CMP robotics problems.

To satisfy hard-constraints without relaxation on the robot motion, the SMP algorithms [LaV06], such as multi-query Probabilistic Road Maps (PRMs) [KL98], and single-query Rapidly-exploring Random Trees (RRTs) [LaV98] with its bidirectional variant [KL00], have been augmented with constraint-adherence methods, such as projection and continuation, to solve a wide range of CMP problems.

The projection-based method was first utilized with a variant of PRMs for parallel manipulators under specialized loop-closure constraints [Han00]. The parallel manipulators were treated as active/passive links and were composed into a constraint-adhering configuration using projection. Yakey et. el [YLK01] introduced the Randomized Gradient Descent (RGD) method for closed-chain kinematics constraints that generates C-space samples and projects them to the constraint manifold. However, their approach required a significant parameter tuning and was later extended to a generalized framework using RRTs and a Jacobian pseudo-inverse based projection method [Sti07]. In a similar vein, Berenson. et al. [BSK11] proposed the Constrained Bidirectional RRT (CBiRRT) with an intuitive constraint representation approach called Task Space Regions (TSRs). TSRs represent general end-effector pose constraints and allow a quick computation of geodesic distances from the constraint manifolds. Another class of sampling-based methods that use projection operators and plan in the task-space include [KKKL94, YKH04, YG05]. These methods find a task-space motion plan and find their corresponding configurations in the C-space, which limits their exploration and thus does not yield completeness guarantees.

The continuation-based methods compute tangent-spaces at a known constraint-adhering configuration to generate new nearby samples for quick projections to the constraint manifold. Yakey et. el [YLK01] used continuation to generate new configuration samples within tangent space, which were projected to the manifold using RGD for closed-chain kinematic constraints. The continuation methods have also been used for general end-effector constraints [WFS07, Sti10]. Inspired by the definition of differentiable manifolds [Spi99], recent approaches do not discard tangent spaces. Instead, they compose them using data-structures into an *atlas* for a piece-wise linear approximation of the constraint manifold [Hen02]. These methods include Atlas-RRT [JP17] and TangentBundle(TB)-RRT [KUSP16] with an underlying single-query bidirectional RRTs algorithm [KL00]. Atlas-RRT ensures all samples to be on the manifold and separates tangent spaces into tangent polytypes using half-spaces for uniform coverage. In contrast, TB-RRT lazily projects the configurations for constraint-adherence, i.e., only when switching the tangent spaces, and has overlapping tangent spaces, which sometimes lead to invalid states. There also exist variants of Atlas-RRT that allow asymptotic optimality [JP13a, JP13b] and kinodynamic planning [BRP18] under constraints.

Recently, Kingston et. el [KMK19] introduced Implicit MAnifold Configuration Spaces (IMACS) to decouple the choice of constraint-adherence methods from the underlying selection of SMP planners. IMACS highlights that any SMP method equipped with the following two components can solve CMP problems. First, a uniform sampling technique to generate samples on the manifold. Second, a constraint integrator function to connect two configurations on the manifold. IMACS incorporates the constraint function into C-Space, presenting an implicit manifold space to an underlying SMP method. These SMP methods, augmented with a constrained integrator, are shown to solve various CMP problems. Despite these advancements, existing

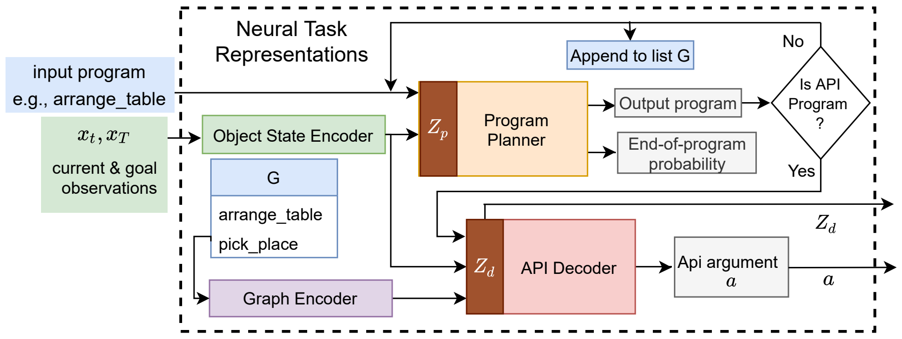


Figure 3.25: The Neural Task Representations for CoMPNetX are obtained by exploiting a learning-based task programmer's internal state Z_d and program arguments a.

SMP methods are computationally inefficient and take up to several minutes for solving practical problems not just in CMP but also in unconstrained planning problems.

CoMPNetX extends IMACS and our previously proposed CoMPNet [QDCY20] and also introduces neural-gradient-based projections to generate informed implicit manifold configurations for underlying SMP methods equipped with any constrained integrator. Our approach can also be interpreted as Neural Informed Implicit MAnifold Configuration Spaces (NIIMACS), which replaces the abstraction layer of IMACS with neural-learned sampling distributions to prioritize sampling in the subsets of a contraint manifold that potentially contains a path solution for a given problem.

3.3.3 Neural Task Representations

This section describes the process to obtain the neural task representations, utilized by CoMPNetX to define task-specific constraints in a scalable and generalizable way. These representations come from the internal state of a learning-based task planner. Although various learning-based task planners can be utilized for acquiring these representations, we adapt a variant [TKH19] of the Neural Task Programming (NTP) [XNZ$^+$18] in our framework.

This variant, which we name NTP2, extends original NTP by relieving the need for task demonstration at the test time. NTP2 uses the goal x_T and current x_t observations of the environment to decompose a given high-level task into a feasible sequence of intermediate sub-tasks. We use NTP2 to obtain the neural task representations and the sub-task sequences for CoMPNetX. It comprises the following modules:

Program Planner: It is a deep neural network-based iterative program predictor that takes a high-level symbolic task p_t, the environment's current x_t and goal x_T observations as an input and outputs a next sub-program p_{t+1} and the end-of-program probability r, indicating the accomplishment of a given task.

API Decoder: A program is defined as an API program if it requires arguments for the execution. Given an api program p predicted by the program planner, the neural networks based API Decoder predicts their required arguments a. The inputs to the API decoder are the current x_t and goal x_T observations, the API program p, and a fixed size graph encoding representing the program hierarchy.

The overall flow of the algorithm is shown in the Fig. 3.25. The current and goal observations are encoded into latent embeddings using their encoders. The program planner, conditioned on observation encodings, iteratively decomposes the given program (e.g., arrange_table) into subprograms by generating a probability distribution over a set of predefined program instances (e.g., pick and place). The program with maximum probability is selected, which becomes the input to the program planner in the next iteration. This process is repeated until an API-program is selected. For instance, the given program, arrange_table, can lead to the selection of a pick_place program which subsequently results in the selection of either pick or place programs. The pick and place are defined as API programs requiring arguments from the API decoder. This API decoder, conditioned on observation encodings and graph embeddings, predicts the API program's arguments indicating the object that needed to be grasped (pick) and moved (place). The graph

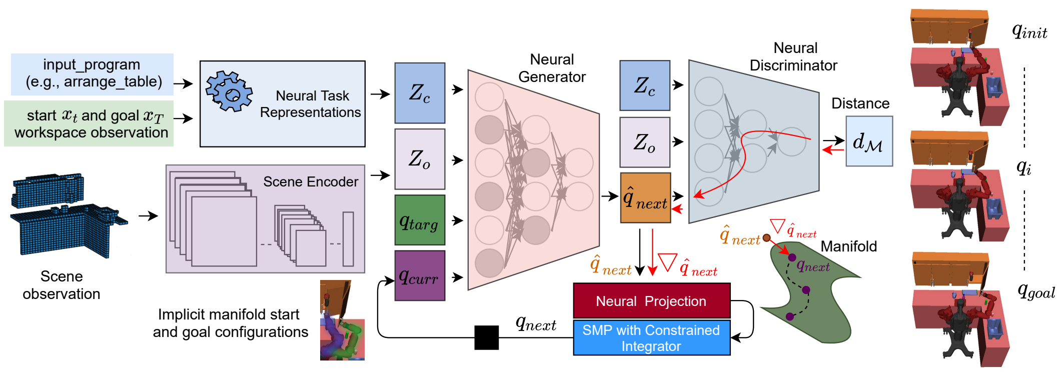


Figure 3.26: CoMPNetX execution traces for the constrained door opening subtask. Our method comprises a conditional neural generator and discriminator and a planning algorithm.

embeddings are given by the graph encoder that takes a list of non-API programs (Fig. 3.25) and encodes them into a fixed-size latent representation. In our implementation of NTP2, the current observation contains the current poses of the given objects in the environment and the robot end-effector pose. The goal observation includes the final poses of all objects at the end of the task. Furthermore, the program planner and the API decoder were trained using the cross-entropy loss for the given expert demonstration. For more details on the implementations, refer to [TKH19].

To generate a neural task representation for the CoMPNetX, we take the latent inner embedding Z_d of API Decoder and their corresponding arguments a (Fig. 3.25). The internal state Z_d comprises current and goal encodings, graph embedding representing the given task hierarchy, and an API program embedding. Note that the latent state Z_d and arguments a contain sufficient information, i.e., a given high-level task, their sub-task hierarchy, and workspace representation, for the CoMPNetX to effectively plan the feasible robot motion path respecting the task constraints at any instant. This is in contrast to the original CoMPNet framework [QDCY20] that relied on hand-engineered task plans, and sub-tasks were represented as text-descriptions, making them oblivious of given high-level tasks, their hierarchical structure, and overall workspace setup.

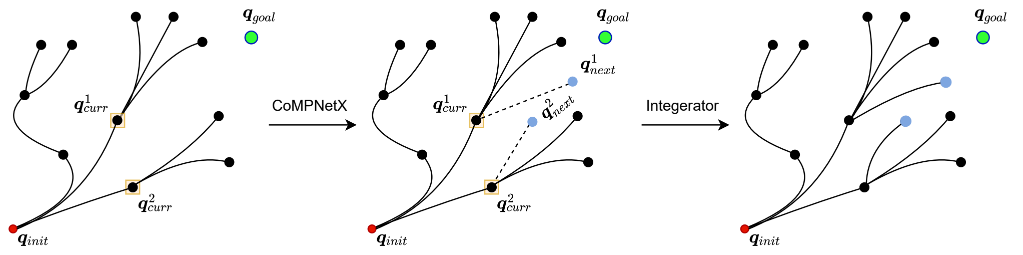


Figure 3.27: *K*-Batch CoMPNetX: The process shows COMPNetX exploiting neural networks parallelization to generate $K = 2$ informed manifold configurations from randomly selected nodes in the tree towards the goal configuration(s).

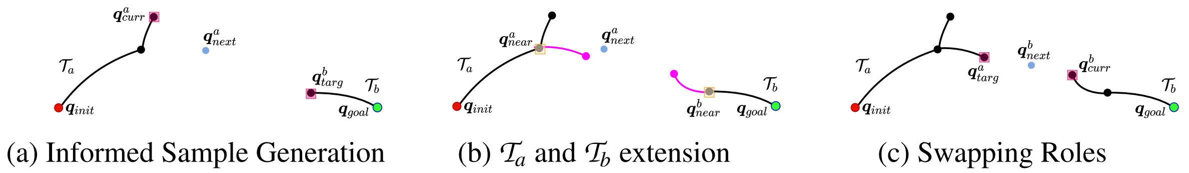


Figure 3.28: Bidirectional CoMPNetX: (a)-(c) show the CoMPNetX bidirectional sample generation, soliciting neural informed-trees from start and goal to quickly march towards each.

3.3.4 Constrained Motion Planning Networks

This section formally presents CoMPNetX (Fig. 3.26), comprising a conditional generator, discriminator, neural projection operator, and neural samplers. The neural generator and discriminator are conditioned on the task and scene observation encodings to generalize across different environments and planning problems. Our method with a constrained integrator and an underlying SMP algorithm generates feasible motion plans on the constraint manifolds for the given CMP problems.

Task Encoder

The task-encoder processes the neural task representations given as $Z_s = [Z_d, a]$. As mentioned earlier, the Z_d is a fixed-sized vector comprising the workspace current and goal observation encodings, the API program embeddings, and the graph encoding (representing the program hierarchy). Our task encoder takes Z_s, comprising Z_d and a, as an input and composes them into a fixed-size latent embedding $Z_c \in \mathbb{R}^{d_1}$ of size d_1 using a neural network.

Scene Encoder

The scene encoder takes the raw environment perception as a 3D depth point-cloud processed into voxels and transforms them to an embedding $Z_o \in \mathbb{R}^{d_2}$ of dimension d_2. The 3D voxel grids of dimensions L $\times$ W $\times$ H $\times$ C are converted into 2D voxel patches as L $\times$ W $\times$ (HC), where L, W, H, and C correspond to length, width, height, and the number of channels, respectively. The voxel patches are encoded into Z_o using a 2D convolutional neural network (CNN). We process 3D voxels into 2D voxel patches as 3D maps require 3D-CNNs, which are known to be computationally intensive and their representations often contain empty volumes [ZLU18]. The scene embedding is passed as a fixed-size feature vector describing the environmental obstacles to a subsequent generator and discriminator. Although neural task representations Z_c contain poses of manipulatable objects in their embeddings, scene observation Z_o also includes information about static non-movable objects acting as obstacles in the environment.

Conditional Neural Generator

CoMPNetX's generator G_ϕ, with parameters ϕ, is a stochastic neural model that outputs a variety of implicit manifold configurations leading to a constrained path solution (Fig. 3.26). Because the generator is trained on both unconstrained and constrained path demonstration data, the output distribution of the neural model tend to fall on or near the constraint manifolds when conditioned on task-specific constraints. Our generator derives its stochastic behavior from using Dropout [SHK+14] during inference, which instantly slices G_ϕ in a probabilistic manner, inculcating variations in the generated samples. Although other techniques such as input Gaussian noise can be used to foster stochasticity, they require a reparametrization trick and are often hard to train end-to-end [CWD+18]. In contrast, Dropout helps capture stochastic behavior from demonstration data, which we observed to be consistently better than hand-crafted input noise distributions in our planning problems.

The generator's input is the task-observation encodings (Z_c and Z_o) that encode the given neural task representations and scene observation, respectively, and the current q_{curr} and target q_{targ} manifold configurations. The output is the next configuration $\hat{q}_{next}$ on/near the constraint manifold that will take the system closer to the given target, i.e.,

$$\hat{q}_{next} \leftarrow G_{\phi}(Z_c, Z_o, q_{curr}, q_{targ}) \tag{3.25}$$

Given the demonstration trajectories $\sigma^* = \{q^*_0, \cdots, q^*_T\}$ from an oracle planner, we train the generator together with the task and observation encoders in an end-to-end manner using the mean-square loss function, i.e.,

$$\frac{1}{N_B}\sum_{i=0}^{N}\sum_{j=0}^{T_i-1} ||q_{i,j+1} - q^*_{i,j+1}||^2, \tag{3.26}$$

where i and j iterates over the number of given paths and the number of nodes in each path, respectively, and N_B is the averaging term.

Conditional Neural Discriminator

CoMPNetX's discriminator D_θ, with parameters θ, is a deterministic neural model that predicts the distance $d_{\mathcal{M}} \in \mathbb{R}$ of a given configuration $\hat{q}$ from an implicit constraint manifold $\mathcal{M}$ conditioned on the task Z_c and observation Z_o encodings, i.e.,

$$d_{\mathcal{M}} \leftarrow D_\theta(\hat{q}, Z_c, Z_o) \tag{3.27}$$

CoMPNetX uses the discriminator predictions and their gradients as the operator, named NProj, to project the given configurations to the constraint manifold if their predicted distances are greater than a threshold ν, thus discriminating samples based on their distances from the manifold and fixing them accordingly as,

$$q \leftarrow \hat{q} - \gamma \nabla_{\hat{q}} D_\theta(\hat{q}, Z_c, Z_o), \tag{3.28}$$

where $\gamma \in \mathbb{R}^+$ is a hyperparameter denoting a step size.

To train the discriminator network D_θ, we minimize the mean-square loss between its predictions and the true labels. The true labels are the geodesic distances of demonstration trajectories from the constraint manifolds. Furthermore, we introduce a trick to create negative training samples with relatively larger distances from the manifold. The negative training samples comprise the robot configuration from the unconstrained tasks (e.g., reach a given object) and the virtual-link configuration from positive training samples and their corresponding distances are computed by querying **F**.

```
Algorithm 15: COMPNetX ($Z_s, v, q_{curr}, q_{targ}$)
    $Z_c \leftarrow$ GetTaskEncoding($Z_s$)
    $Z_o \leftarrow$ GetObsEncoding($v$)
    $\hat{q}_{next} \leftarrow G_\phi(Z_c, Z_o, q_{curr}, q_{targ})$
    $d_{\mathcal{M}} \leftarrow D_\theta(\hat{q}_{next}, Z_c, Z_o)$
    if $d_{\mathcal{M}} > \nu$ then
        $\hat{q}_{next} \leftarrow \hat{q}_{next} - \gamma \nabla_{\hat{q}_{next}} D_\theta(\hat{q}_{next}, Z_c, Z_o)$
    return $\hat{q}_{next}$
```

Neural Samplers

Once trained, CoMPNetX can be used in a number of ways to generate informed neural samples for the underlying SMP algorithms equipped with a constrained adherence method. Fig. 3.26 and Algorithm. 15 present an overall flow of information between different neural modules of CoMPNetX. For a given current q_{curr} and target q_{targ} configuration(s), COMPNetX, conditioned on encodings Z_c and Z_o, generates the next configuration(s) $\hat{q}_{next}$ and projects them towards the constraint manifold using neural gradients if needed. Thanks to CoMPNetX's informed but stochastic sampling and built-in parallelization capacity of neural networks, our method can be adapted to most of underlying SMP methods. For case studies, we present two sampling strategies named *K*-Batch CoMPNetX and Bidirectional CoMPNetX, which together cover a wide range of SMP methods.

Algorithm 16: *K*-Batch COMPNetX

```
$\mathcal{T} \leftarrow$ InitializeSMP($q_{init}, q_{goal}$)
$K_{q_{targ}} \leftarrow K$Replicas($q_{goal}$)
for $i \leftarrow 0$ to $N_{max}$ do
    if $i < N_{ismp}$ then
        $K_{q_{curr}} \leftarrow$ SelectNodes($\mathcal{T}, K$)
        $K_{q_{next}} \leftarrow$ CoMPNetX($K_{Z_s}, K_v, K_{q_{curr}}, K_{q_{targ}}$)
    else
        $K_{q_{next}} \leftarrow$ TraditionalSMP()
    goal_reached $\leftarrow$ SMP($K_{q_{next}}, \mathcal{T}$)
    if goal_reached then
        $\sigma \leftarrow$ ExtractPath($\mathcal{T}$)
if $\sigma$ is not empty then
    ExecutePlan($\sigma$)
else
    return Failure or AskExpert
return $\varnothing$
```

***K*-Batch CoMPNetX:** Our approach exploits the neural networks' innate parallelization capacity to generate a batch of samples with size $K \in \mathbb{N}_{\geq 1}$ using CoMPNetX for the underlying unidirectional ($K = 1$) and batch ($K > 1$) SMP methods. In this setup, the input to CoMPNetX is in the form of batches of size K. The K target configurations q_{targ} are a set of samples from goal region $\mathcal{G}_{goal}$. The voxel map v and neural task representation Z_s are simply replicated K times. The K current configurations q_{curr} are obtained by randomly selecting K nodes in the graph leading to their corresponding next output configurations as follows:

$$K_{q_{next}} = \text{CoMPNetX}\left(K_{Z_s}, K_v, K_{q_{curr}}, K_{q_{targ}}\right),$$

$$\textbf{where } K_{q_{next}} = \begin{bmatrix} q^1_{next} \\ \vdots \\ q^K_{next} \end{bmatrix}, \cdots, K_{q_{targ}} = \begin{bmatrix} q^1_{targ} \\ \vdots \\ q^K_{targ} \end{bmatrix} \tag{3.29}$$

At the beginning of planning, the graph $\mathcal{T}$ might have only one sample, i.e., q_{init}. In that case, an initial set of $K_{q_{curr}}$ can be created by randomly sampling the manifold $\mathcal{M}_{free}$ or replicating q_{init} for K times. Fig. 3.27 shows a case with $K = 2$, and Algorithm. 16 presents K-Batch CoMPNetX algorithm with an underlying SMP. This approach is not just for batch sampling methods such as FMT* [JSCP15] and BIT* [GSB15] but can also be applied to any unidirectional SMP method like RRT [LaV98, KL00] and PRMs [KL98] by setting $K = 1$. Furthermore, our procedure shifts to traditional sampling techniques, introduced in IMACS [KMK19], after generating neural informed implicit manifold configurations using CoMPNetX for N_{smp} iterations. This allows our framework to explore the entire space in worst-case, leading to theoretical guarantees expected from a planning algorithm.

Bidirectional CoMPNetX: This approach incorporates Bidirectional SMP (BiSMP) methods into CoMPNetX that generate bidirectional trees $\mathcal{T}_a = (V, E)$ and $\mathcal{T}_b = (V, E)$ originating from the start q_{init} and goal q_{goal} configurations, respectively, with vertices V and edges E. Although the following approach can be formulated as K-Batch bidirectional CoMPNetX, we consider $K = 1$ and drop down the K notations introduced in the previous section for brevity. Furthermore, we also show that our approach can be combined with learning-based task planners such as NTP2 that generate neural task representations and intermediate subtasks for CoMPNetX, which in return accomplishes those subtasks, forming a mutually symbiotic relationship.

Algorithm 17: Bidirectional COMPNetX

```
$t \leftarrow 1$; $p_0 \leftarrow$ input_program
while not end_of_program do
    $x_t, v_t \leftarrow$ GetObservation()
    $p_t, Z_s,$ end_of_program $\leftarrow$ NTP2($x_t, x_T, p_{t-1}$)
    $q_{init}, q_{goal} \leftarrow$ GetConfigs($p_{\mathrm{t}}, Z_{\mathrm{s}}$)
    $\mathcal{T}_a, \mathcal{T}_b \leftarrow$ InitializeBiSMP($q_{init}, q_{goal}$)
    $q^a_{curr}, q^b_{targ} \leftarrow q_{init}, q_{goal}$
    for $i \leftarrow 0$ to $N_{max}$ do
        if $i < N_{ismp}$ then
            $q^a_{next} \leftarrow$ CoMPNetX($Z_s, v_t, q^a_{curr}, q^b_{targ}$)
        else
            $q^a_{next} \leftarrow$ TraditonalSMP()
        $q^a_{next}$, path_found $\leftarrow$ BiSMP($q^a_{next}, \mathcal{T}_a, \mathcal{T}_b$)
        if path_found then
            $\sigma_t \leftarrow$ ExtractPath($\mathcal{T}_a, \mathcal{T}_b$)
        $q^a_{curr} \leftarrow q^a_{next}$
        Swap($\mathcal{T}_a, \mathcal{T}_b$)
        Swap($q_{curr}, q_{targ}$)
    if $\sigma_t$ is not empty then
        ExecutePlan($\sigma_t$)
    else
        return Failure or AskExpert
    $t \leftarrow t + 1$
return $\varnothing$
```

In this procedure, CoMPNetX alternatively generates samples for both trees and greedily expands them towards each other by having current and target configurations in the opposite trees (Fig. 3.28), i.e.,

$$\textbf{Forward: } q^a_{next} \leftarrow \text{CoMPNetX}\left(Z_s, v, q^a_{curr}, q^b_{targ}\right)$$

$$\textbf{Backward: } q^b_{next} \leftarrow \text{CoMPNetX}\left(Z_s, v, q^b_{curr}, q^a_{targ}\right)$$

where configurations with superscript a and b corresponds to the tree $\mathcal{T}_a$ and $\mathcal{T}_b$, respectively.

Algorithm 17 presents an overall framework using NTP2 and CoMPNetX with an underlying bidirectional SMP algorithm, like RRTConnect [KL00], and a constrained-adherence method. NTP2 takes the current environment observation x_t, previous task program p_{t-1}, and the desired goal observation x_T and generates the next program p_t with their representation Z_s. The procedure GetConfigs takes the generated task information (p_t, Z_s) and obtains their corresponding start and goal configurations. These configurations and task-scene representations are given to CoMPNetX-BiSMP to accomplish the given subtask by generating a feasible motion plan.

Fig. 3.28 illustrates the internal process of a BiSMP, such as RRTConnect, using CoMPNetX generated samples. Let's assume tree $\mathcal{T}_a$ current configuration being used to generate the next sample (Fig. 3.28 (a)). The underlying BiSMP begins by extending $\mathcal{T}_a$ towards the next configuration q^a_{next} and updates q^a_{next} with the last state reached by constrained integrator towards the target q^b_{targ} (Fig. 3.28 (b)). The process then extends $\mathcal{T}_b$ towards the q^a_{next} and the extension process ends by returning updated q^a_{next} and a boolean path_found. The path_found is true when trees $\mathcal{T}_a$ and $\mathcal{T}_b$ are connected, depending on trees' connection strategy of an underlying BiSMP, and there exists a path between start and goal that satisfies all the desired constraints. To solicit bidirectional path generation using CoMPNetX, the roles of current and target configurations are also swapped along with planning trees' roles at the end of each planning iteration (Fig. 3.28 (c)). Our CoMPNetX-BiSMP quickly finds a path solution by exploiting the moving targets from its own distribution which improves the stability of the generator to find connectable paths as satisfying the two-point boundary value problem becomes easier when the two goal states are iteratively sampled from a distribution encoded by the generator, rather than one defined arbitrarily during the problem definition.

Note that the constraint function $\mathbf{F}$ is used only by an underlying SMP method. Furthermore like CoMPNet, CoMPNetX (batch and bidirectional) also extends the planning graph from the nearest node of the newly generated next node since all underlying SMP algorithms rely on the nearest neighbor for their graph extension towards the given configuration sample [LaV06]. It is also in contrast to the MPNet algorithm [QSBY19, QMSY20] that greedily finds a path by extending from q_{curr} to q_{next} in an overall planning method and repairs any non-connectable nodes via stochastic re-planning. Although the MPNet approach works extremely fast in unconstrained planning problems, re-planning becomes computationally expensive in CMP due to projections performed by the constrained integrator. Moreover, the constraint manifolds are non-euclidean in topography, and extension from nearest neighbors becomes convenient for geodesic interpolation. This is evident from the experimentation in our previous work [QDCY20], showing that leveraging MPNet's greedy path-finding approach, without replanning, often fails in finding a connectable path solution on the manifolds. However, in our extended analysis presented in this work, we show that CoMPNetX, in addition to CMP, can still be used with the MPNet planning algorithm for efficiently solving unconstrained planning problems with low computation times and high success rates in high-dimensional planning problems.

3.3.5 Implementation details

This section describes the data generation pipeline from setting up scenarios to obtaining expert demonstrations and observation data. We also describe training, and testing data splits for all scenarios considered in this work.

Scene setup

We setup the following cluttered environments imposing various hard kinematic constraints on the robot motion:

Sphere Environment: This environment requires the motion planning of a point-mass on the sphere with constraint $\mathbf{F}(q) = \|q\| - 1$, forming a two-dimensional manifold on a three-dimensional ambient space. In this setup, we create two scenarios:

- Scenario 1 - We generate 50 unique scenes by randomly placing 500 small obstacle blocks over the sphere (Fig. 3.23 (a)). For each scene, we randomly sample 2000 start and goal pairs on the obstacle-free space of the sphere.

- Scenario 2 - This setup requires transversing multiple narrow passages between the randomly selected start and goal configurations (Fig. 3.23 (b)). We randomly sample the unique 500 start and goal pairs from the obstacle-free space, each of which constitutes a CMP problem. This setup is only used to test our model's generalization capacity, trained on sphere - scenario 1, to an entirely different environment.

Bartender Environment: A dataset, named Bartender environment, containing three different scenarios was created to fully capture the complexities of the real-world task and constrained motion planning problems. The environment includes two tables placed perpendicular to each other. The table contains seven objects placed at random, and only five are movable under pre-specified motion constraints. The five movable/manipulatable items include a juice can (green), fuze bottle (purple), soda can (red), kettle, and red mug. The two stationary objects include a tray and a recycling bin that form the movable objects' goal locations. The juice can, soda can, and fuze bottle are to be placed into the recycling bin with only collision-avoidance constraints. The kettle and the red mug are to be placed on the tray with both stability and collision-avoidance constraints, i.e., no tilting is allowed during the robot motion. The three different scenarios are described as follow.

- Scenario 1 - In this scenario, the objects can be moved to their targets in any order. In other words, in most cases, all objects start, and goal configurations are reachable. We generate about 2000 unique scenes through the random placement of the movable and non-movable objects on the tables at the robot's right arm's reachable locations. Each scene contains a total of ten (five unconstrained and five constrained) planning problems.

- Scenario 2 - In this scenario, the goal location of either the red mug or the kettle contains an obstacle. The obstacles are formed by placing either juice bottle, fuze bottle, or soda can, at the goal location of the kettle or the red mug. For example, if the red mug's goal location contains the juice bottle, the task planner needs to account for this information during the planning process. That is, the juice bottle needs to be moved into the recycling bin before the red mug is attempted to be moved onto the tray. This enforces a constraint on the task planner to account for obstacles. We created 700 scenes in this setup, each with at least two constrained and two unconstrained planning problems.

- Scenario 3 - In this setup, the kettle and red mug are placed on the tray, and the task is to swap their start locations. In other words, the goal locations of both the kettle and the red mug are occupied by the red mug and the kettle, respectively. Therefore, there is a need for a sub-goal generation for one of the objects. For example, the tea kettle should be moved to a temporary location on the table. This is followed by the pick-place of the plastic mug to its goal location. Finally, the goal location of the tea kettle is now free for its pick-place operation. For this problem, we created 300 unique cases by random placement of the tray, and each case contained atleast six planning problems, i.e., three constrained and three unconstrained.

Kitchen Environment: In this scenario we have seven manipulatable objects: soda can, juice can, fuze bottle, cabinet door, black mug, red mug, and pitcher. The objective is to move the cans and bottle to the trash bin, open the cabinet door from any starting angle to a fixed final angle ($\pi/2.7$), transfer (without tilting) the black and red mugs from the cabinet to the tray, and move the pitcher from the table into the cabinet. We construct 2000 unique scenarios by the random placement of the trash bin, tray, and manipulatable objects (excluding door) on the table and by randomly selecting the cabinet's door starting angle between 0 to $\pi/4$. Each scenario contains a total of 14 planning problems, i.e., seven unconstrained (reach) and seven constrained (manipulation) problems.

Training & testing data splits

In the sphere (scenario 1), we use 40 environments for training and 10 for testing. The sphere (scenario 2) is used for testing only. In the bartender (scenario 1), and kitchen environments, we use 10% data for testing, and the remainder is used for training. All training paths were generated by an oracle planner, i.e., Atlas-RRTConnect. To train neural task programmer on all bartender (scenarios 1, 2 & 3) and kitchen environments, we use the same data split ratio, i.e., 10% is kept for testing. Note that the CoMPNetX is never trained on the sphere (scenario 2) and Bartender scenarios 2 and 3. We use them to evaluate CoMPNetX generalization capacity across different environment structures and planning problems.

Observation data

In the sphere environment, the observation data is a point-cloud converted into a voxel map of size $40 \times 40 \times 40$. However, for the other high-dimensional robot environments (Bartender and Kitchen), there exist workspace and entire scene observations at any time instant t. The workspace observation includes the current $x_t \in \mathcal{X}$ and the target $x_T \in \mathcal{X}$. The current workspace observation x_t at a given time is represented by each objects' poses and the robot end-effector pose. The target x_T is represented by the objects' target poses at the completion of the entire task. The scene observation is also a function of time represented as a voxel map v_t at instant t. We obtain raw point-cloud data from multiple Kinect sensors and process into voxel maps. The Kinect sensors are placed in the bartender and kitchen environments leading to voxel maps of dimensions $33 \times 33 \times 33$ and $32 \times 32 \times 32$, respectively.

NTP2: Programs and API Arguments Set

In our NTP2 setup, the list of initial programs includes arrange_table and swap_tray_objs. The Bartender (setup 1 and 2) and Kitchen tasks begin with the former, whereas the Bartender setup 3 begins with the latter program. The initial program can call either pick_place, subgoal_gen,

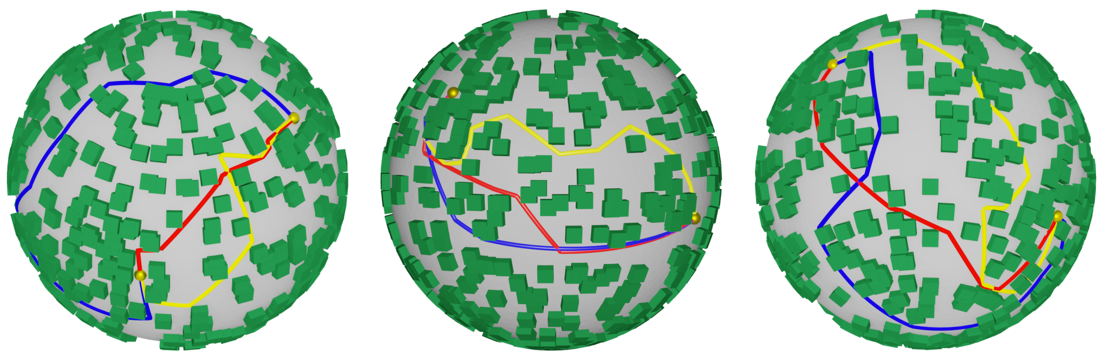
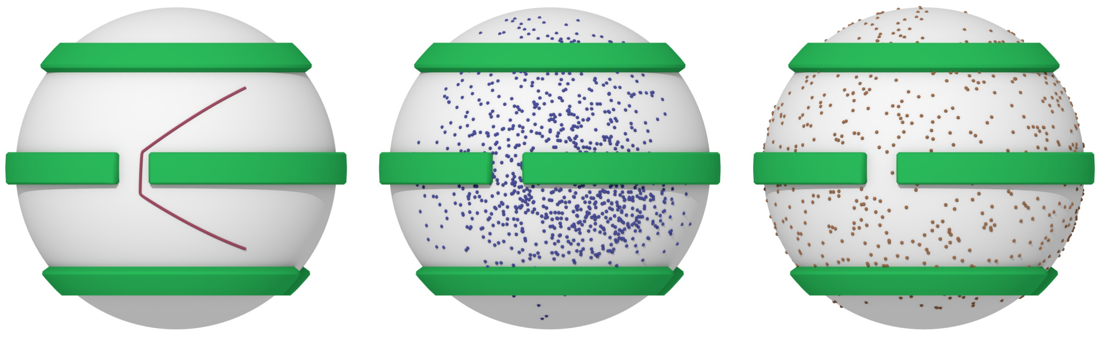


Figure 3.29: Sphere Environment (Scenario 1): The paths found by CoMPNetX-FMT* (red), FMT* (yellow) and RRTConnect (blue) with atlas operator in three example scenes.

return_arm, or no_op programs followed by their underlying API-programs named pick and place. The API-programs pick and place represent an unconstrained planning problem, requiring a robot to reach a given target/object, and a constrained planning problem, demanding manipulation under manifold constraints, respectively. An API-program also gets an argument, predicted by the API-decoder, which in our cases, is one of the objects (e.g., juice can, fuze bottle, soda can, etc.) to be picked or placed in the given scenario. Furthermore, the program return_arm requires a robot to return to its initial default configuration from any starting state, and the program no_op means no operation needed. Finally, the subgoal_gen is executed to move objects acting as obstacles out of the way through pick-place procedures to achieve the desired sub-task.

3.3.6 Results

In this section, we present the results and analysis of the following evaluation studies: (i) A comparison study evaluating CoMPNetX and state-of-the-art classical SMP planning methods with an underlying constraint-adherence approach (projection, atlas, or tangent bundle) on unseen challenging problems in environments named Sphere, Bartender, and Kitchen. (ii) An ablation study comparing CoMPNetX with its ablated models and our previous method



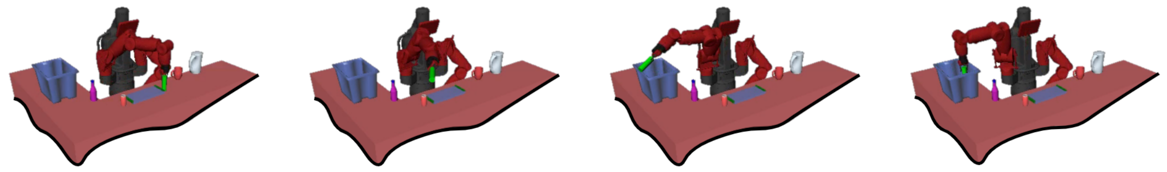
(a) Planning Problem (b) CoMPNetX Sampling (c) Uniform Sampling

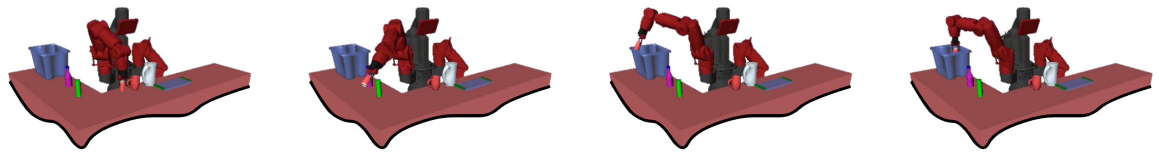
Figure 3.30: Sphere Environment: CoMPNetX stochastically generates samples in the subset that potentially contains a path solution. It contrasts with traditional approaches that randomly explore the entire space.

CoMPNet [QDCY20]. (iii) An extended evaluation to highlight the mutualistic relationship of learning-based task programmers and CoMPNetX and their capacity to generalize across different planning domains.

Comparative analysis

This section compares SMP methods augmented with batch and bidirectional CoMPNetX against their classical setups in solving CMP problems. In the batch method, we select FMT* [JSCP15], a state-of-the-art classical SMP algorithm, and it is proven to perform better than standard approaches like RRT* and PRM* [KF11]. FMT* begins with an initial batch of N_{init} uniform samples, including a goal configuration. In case, the initial set of samples does not yield a path solution, FMT* continues to expand the tree by generating a new random sample in every planning iteration. We choose FMT* to highlight the flexibility offered by CoMPNetX in generating sample batches with different $K \geq 1$ according to the given SMP method. In CoMPNetX-FMT*, we generate an initial batch of N_{init} samples with $K << N_{init}$. The initial K configurations are randomly sampled from an obstacle-free space, to form $K_{q_{curr}}$, which are passed to CoMPNetX to obtain the next set of configurations $K_{q_{next}}$. In the next step, the $K_{q_{next}}$


(a) Move juice can to the trash bin.


(b) Move soda can to the trash bin.

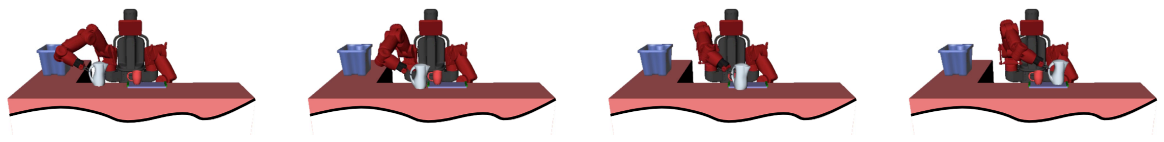
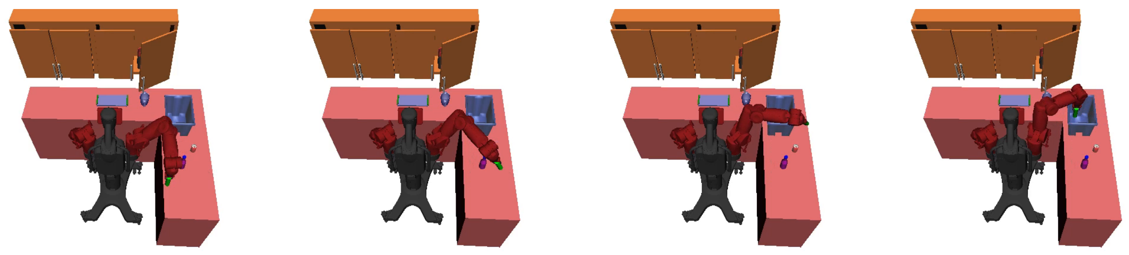
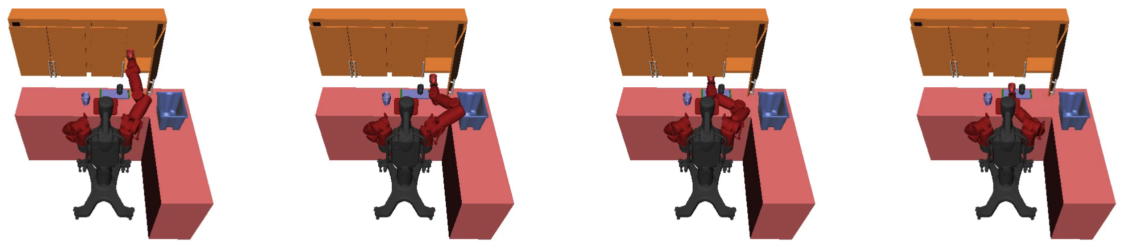

(c) Carefully place the kettle, i.e., without tilting, onto the tray.

Figure 3.31: Bartender setup (SC1): Figs. (a-c) show CoMPNetX motion sequences of moving juice can, soda can, and kettle to their targets in three different test cases.

becomes $K_{q_{curr}}$, and the process of sample generation with CoMPNetX is repeated until it gathers an initial batch of size N_{init}, which also includes a goal state. Furthermore, in case, the initial batch does not yield a path solution, in every subsequent planning step, we randomly select a node in the FMT* tree as an input to CoMPNetX to generate a new informed sample towards the given target (see Algorithm 16). In the bidirectional CoMPNetX, we merge CoMPNetX into RRTConnect, as reported in Algorithm 17.

In the sphere environment, we evaluate both batch and bidirectional CoMPNetX. In the first scenario (Fig. 3.29), CoMPNetX-FMT* with atlas, tangent-bundle, and projection exhibit similar performances with over 99% success rates, computation times of about 0.89 ± 0.13 seconds, and the path lengths of about 2.18 ± 0.014 units. In contrast, FMT* with atlas, tangent-bundle, and projection appeared to have 1.22 ± 0.034, 1.34 ± 0.061, and 1.14 ± 0.027 seconds computation times, around 95%, 89% and 98% success rates, respectively, and somewhat similar


(a) Move the juice can to the trash.


(b) Carefully place, without tilting, the red mug from cabinet onto the tray.

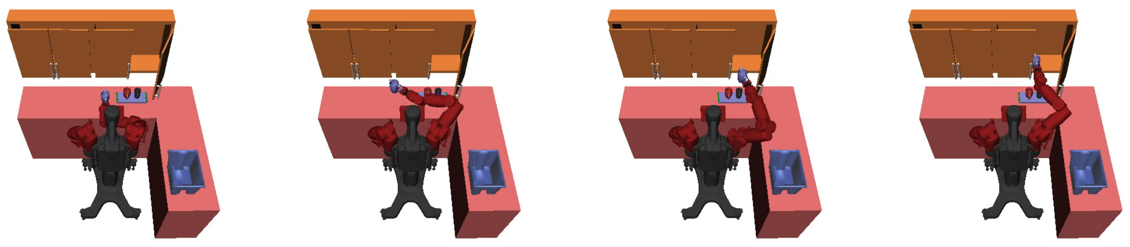

(c) Carefully move pitcher into the cabinet.

Figure 3.32: Kitchen setup: Figs. (a-c) show instances of CoMPNetX planned motions for the juice can, red mug, and pitcher under constraints in three different test scenarios.

path lengths as CoMPNetX. Overall, COMPNetX-FMT* finds near-optimal paths with lower computation times and higher success rates than classical FMT*. On the other hand, CoMPNetX-RRTConnect and RRTConnect exhibits similar computation times of about 0.01-0.02 seconds. However, the former's path lengths appeared to be significantly better than the latter approach, as shown in Fig. 3.29.

The second sphere scenario, is entirely an unseen environment for CoMPNetX as it was only trained on scenario 1. In this case, CoMPNetX-FMT* with all types of constraint-adherence operators demonstrate performances with around 96% success rates, computation times of about

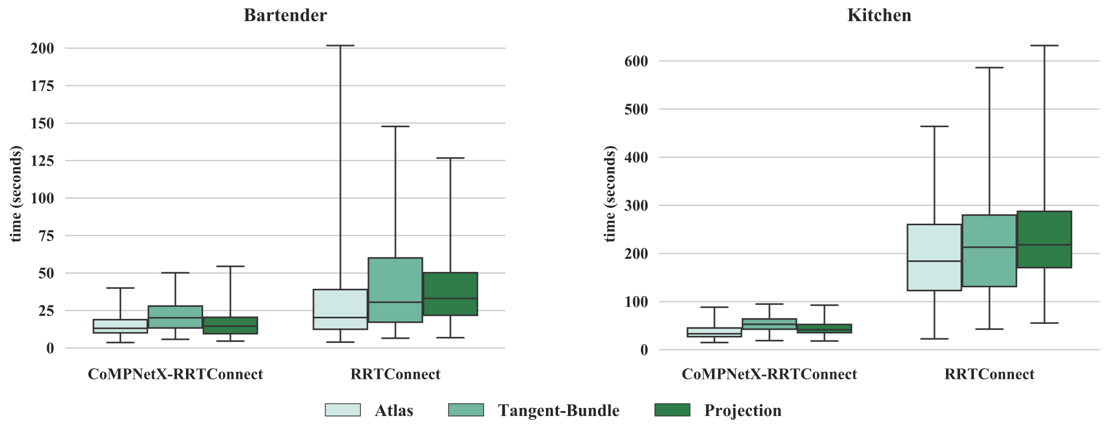


Figure 3.33: The boxplots show the total computation times of CoMPNetX-RRTConnect and RRTConnect with atlas, tangent-bundle, and projection-based constraint operators in the bartender and kitchen environments.

0.55 ± 0.11 seconds, and the path lengths of about 3.13 ± 1.8 units. The classical FMT* with atlas, tangent-bundle, and projection takes 1.93 ± 1.34, 2.05 ± 1.44, and 0.95 ± 0.34 seconds computation times with success rate of around 89%, 87% and 100%, respectively. Generally, COMPNetX-FMT* found better quality paths, in terms of path lengths, with lower computation times and generalized to this new environment with high success rates comparable to classical FMT*. CoMPNetX-RRTConnect and RRTConnect performed similarly in terms of computation times but the latter provides poor quality path solutions. Fig. 3.30 depicts the exploration by CoMPNetX and uniform sampling for a given problem. It can be seen that our method explores the space that potentially contains a path solution, thus leading to better performance, and this becomes even more significant in high-dimensional problems, as presented in the remainder of this section.

In the Bartender (scenario 1) and Kitchen environments, to solve constrained manipulation tasks, we focus on bidirectional SMP methods only since they have become a standard tool for solving high-dimensional CMP problems, and other methods such as unidirectional SMP algorithms struggle in such cases and exhibit high computational times with low success rates [JP17, KUSP16, BSK11]. We evaluate CoMPNetX-RRTConnect and traditional RRTConnect with Atlas, Tangent Bundle, and Projection-based constrained-adherence methods, resulting in Atlas-RRT [JP17], TB-RRT [KUSP16], and CBiRRT [BSK11] algorithms. Figs. 3.31 (a-c) show instances of CoMPNetX-RRTConnect generating motions in the Bartender (Scenario 1) for manipulating juice can (Fig. (a)), soda can (Fig. (b)), and kettle (Fig. (c)) in three different test scenarios. Likewise, Figs. 3.32 (a-c) shows CoMPNetX-RRTConnect motion sequences for moving juice can, red mug, and pitcher in three different kitchen scenarios. Furthermore, Fig. 3.26 displayed waypoints generated by our approach for the cabinet's door opening task.

In these high-dimensional CMP scenarios, Fig. 3.33 provides the box plots of the total computational time to solve all the manipulation tasks. Table 3.7 presents the mean success rates with their standard deviations of all methods. Furthermore, Table 3.8 compares the mean computation times with standard deviations for the individual objects, grouped by their constraint types, in each of the scenarios.

It can be seen that our method exhibits significantly lower inter-quartile computational time ranges with a narrow spread than other methods while retaining similar success rates. Moreover, the results also show that with the increasing complexity of the planning problems from sphere to kitchen environment, the computation times of traditional methods increase significantly with large standard deviations compared to our approach. For instance, between bartender and kitchen task (Table 3.8), the planning times of manipulation under collision-avoidance and stability constraints increase from ~ 1 to ~ 7 seconds for CoMPNetX-RRTConnect and from ~ 1 to ~ 40 seconds for traditional RRTConnect. blue Furthermore, in these experiments, we randomize the positioning of objects in the environments, so the models are shown to generalize to new, unseen objects' positioning. However, these models can also generalize to new objects if trained accordingly with a variety of different items.

Table 3.7: The total mean success rates with standard deviations, over five trials, of CoMPNetX-RRTConnect and traditional RRTConnect for solving all manipulation problems in the bartender and kitchen environments.

Algorithms	Type of Constraint-Adherence	Bartender (%)	Kitchen (%)
RRTConnect	Projection	98.3 ± 1.1	87.7 ± 4.8
	Atlas	99.2 ± 0.6	95.4 ± 3.8
	Tangent-bundle	97.8 ± 2.4	90.1 ± 5.7
CoMPNetX-RRTConnect	Projection	98.3 ± 0.6	88.4 ± 1.8
	Atlas	99.8 ± 0.1	95.3 ± 1.3
	Tangent-bundle	97.8 ± 2.3	90.4 ± 2.7

Table 3.8: The computation time comparison of CoMPNetX-RRTConnect and RRTConnect in solving manipulation problems for individual objects. The objects are denoted by their first letter and grouped by their constraints.

Algorithms	Type of	Bartender		Kitchen		
	Constraint-Adherence	J/F/S	R/K	J/F/S	C	R/B/P
Traditional	Projection	12.64 ± 8.21	1.06 ± 0.87	32.64 ± 22.40	0.05 ± 0.04	49.79 ± 22.96
	Atlas	10.86 ± 11.08	0.93 ± 0.82	24.87 ± 19.81	0.04 ± 0.03	41.28 ± 24.02
	Tangent-bundle	16.23 ± 14.78	1.68 ± 0.82	27.54 ± 21.47	0.05 ± 0.03	46.61 ± 26.06
CoMPNetX	Projection	4.59 ± 2.75	1.17 ± 1.21	8.02 ± 3.34	0.04 ± 0.01	7.70 ± 3.63
	Atlas	4.51 ± 2.24	0.77 ± 0.38	6.26 ± 3.44	0.02 ± 0.01	5.84 ± 2.78
	Tangent-bundle	6.32 ± 3.14	1.21 ± 1.09	8.68 ± 3.34	0.04 ± 0.02	9.62 ± 3.37

Ablative analysis

In this analysis, we ablate various components of CoMPNetX to highlight their significance in solving complex CMP problems. Table 3.9 summarizes the results with mean total computation time and their standard deviation, and mean success rates for solving all manipulation problems in the Bartender (scenario 1) and Kitchen environments.

The first alteration is to remove the NProj, i.e., the neural discriminator's gradient-based

Table 3.9: The total mean computation times with standard deviations and mean success rates are presented for various sampling approaches with an underlying RRTConnect and atlas-based integrator.

Tasks	Algorithms with Atlas Integrator and an underlying RRTConnect			
	CoMPNetX	CoMPNetX (w/o NProj)	CoMPNet	Continuation-based Sampling
Bartender	15.05 ± 06.86 (99.8%)	17.31 ± 09.10 (98.1%)	19.77 ± 10.84 (94.3%)	33.34 ± 33.16 (99.2%)
Kitchen	36.47 ± 14.16 (95.4%)	42.57 ± 14.94 (93.0%)	46.93 ± 15.79 (90.8%)	194.17 ± 96.39 (95.4%)

projections (Eqn. 3.28), from CoMPNetX. Note that in Algorithm 15, we use NProj only when the distance of generated configurations from the manifold is greater than ν. The value of ν is selected to be a positive scalar multiple of the tolerance ε so that to fix those configurations that are an order of magnitude distance away from the manifold than the allowed tolerance. It can be seen that there are only a fraction of cases (2-3 %) where generated configurations by the Neural Generator were not close to the manifold in constrained planning problems and fixing them with NProj led to performance gains.

The second ablation is to evaluate the impact of neural task representations on CoMPNetX. In our previous work [QDCY20], we show that task-representations are crucial for CoMPNet. In that setting, text-based task representations led to significant improvements in performance than CoMPNet without any task-representations. Moreover, the results also highlighted that text-based representations become better than simple one-hot encoding when the number of tasks increases, e.g., from bartender to kitchen environments. It is because one-hot representations become very limited in practice with a growing set of multi-task and multimodal constraints. In this study, we now compare the neural task and text-based task representations for constrained neural motion planning. Table 3.9 presents the comparison of CoMPNetX (with neural task representations and without NProj) and CoMPNet [QDCY20] (with text-based task representation and without NProj) in solving all the manipulation problems in the bartender and kitchen environments. The results indicate that the former, i.e., neural task representations, leads to better performance than the latter in computation times and success rates. Moreover, the statistical paired testing of these two methods resulted in p-values of 1.13×10^{-06} and 4.84×10^{-07} in the bartender and kitchen environments, which validates that CoMPNetX outperforms CoMPNet [QDCY20] by a significant margin. The reason is that the neural task representations consider the workspace observation and the overall program hierarchy, whereas the text-representations are agnostic of underlying task semantics. Furthermore, the learning-based task programmer not just provide task representations but also generate a task plan that saves lot of effort in hand-engineering sub-task sequences. Nevertheless, despite all ablations of CoMPNetX, it can be seen that our method performs significantly better than classical sampling techniques.

Extended Analysis: Mutual Symbiotic Relationship

In our comparative analysis, we show that CoMPNetX generalizes to new locations of the objects (i.e., not seen during training) and solves those practical problems in few seconds where gold standard SMP methods take up to several minutes to obtain comparable success rates. In this extended analysis, we show the joint operation of learning-based task programmer and CoMPNetX and evaluate our models, trained on bartender (scenario 1), for further generalization to new problems, such as in bartender scenarios 2 and 3, to simultaneously solve both unconstrained and constrained planning problems. Note that our trained model on bartender scenario 1 never had cases where either start, goal, or both states of the given sub-task object were occupied by other objects, acting as obstacles. Therefore, the planner needs to move them out of the way before accomplishing the desired sub-task.

Table 3.10 presents the total mean computation times with mean success rates of CoMPNetX for solving all unconstrained (pick) and constrained (place) tasks in the bartender scenarios 2 and 3. In these scenarios, the NTP2 success rate was about 95% and 89%, and from those successful cases, CoMPNetX achieves around 90% and 80% success rate, respectively, in solving given motion planning problems. In unconstrained planning problems, CoMPNetX calls an underlying MPNet algorithm [QMSY20], solving problems in 2-3 seconds computation time, i.e., the computational gains over gold-standard SMPs are retained for unconstrained problems as well. In constrained planning problems, CoMPNetX uses RRTConnect, and their computation times for individual tasks were similar to reported for Bartender scenario 1.

Fig. 3.34 and Fig.3.35 show the joint execution of NTP2 and CoMPNetX in one of the cases in bartender scenarios 2 and 3, respectively. In this particular case of scenario 2, the task is to move the soda can out of the red mug's target location and then move the red mug to its

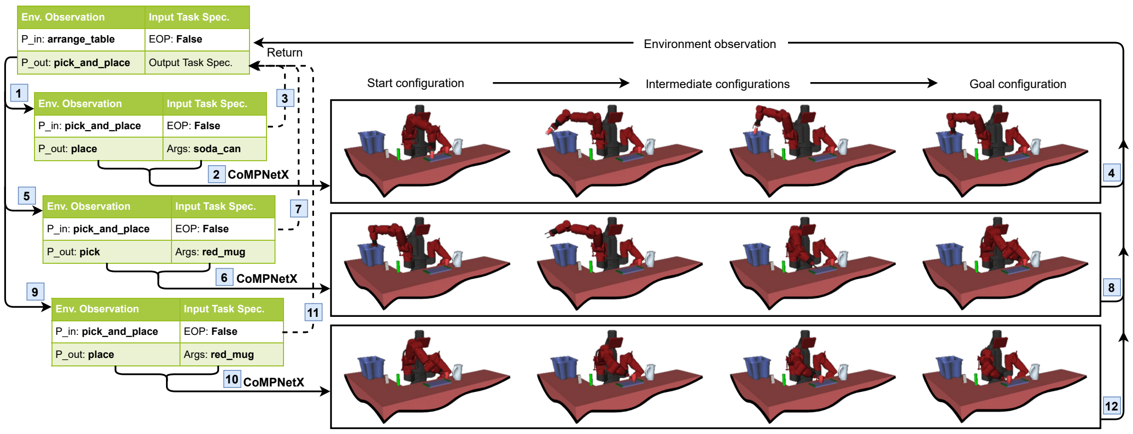


Figure 3.34: A mutual symbiotic operation of a learning-based task programmer and CoMPNetX in the Bartender scenario 2. The numbers in small boxes indicate the order in which the procedures are executed.

Table 3.10: The total computation times and mean success rates for CoMPNetX, solving both unconstrained and constrained problems with underlying MPNet and RRTConnect algorithms, respectively, in the Bartender scenarios 2 and 3.

Datasets	Planning times and success rates	
	Pick (Unconstrained)	Place (Constrained)
Bartender-SC2	8.57 ± 5.18 (88.7%)	17.47 ± 10.07 (93.1%)
Bartender-SC3	6.42 ± 3.21 (79.3%)	10.23 ± 3.17 (81.2%)

desired place. Likewise, in this scenario 3 example, the robot has to swap both red mug and kettle locations, which represents a situation where both start and goal locations of the objects are occupied. It can be seen that cross-fertilization of neural task programmers and neural motion planners are crucial for solving challenging practical problems, and CoMPNetX with neural task representation exhibits generalization to problems outside the domain of its training set.

3.3.7 Discussion

In this section, we briefly discuss the CoMPNetX stochastic behavior and its benefits in learning-based motion planning and the way our approach retains the completeness and optimality guarantees of an underlying SMP planner.

CoMPNetX, like its predecessors, uses Dropout [SHK+14], with fifty percent probability,

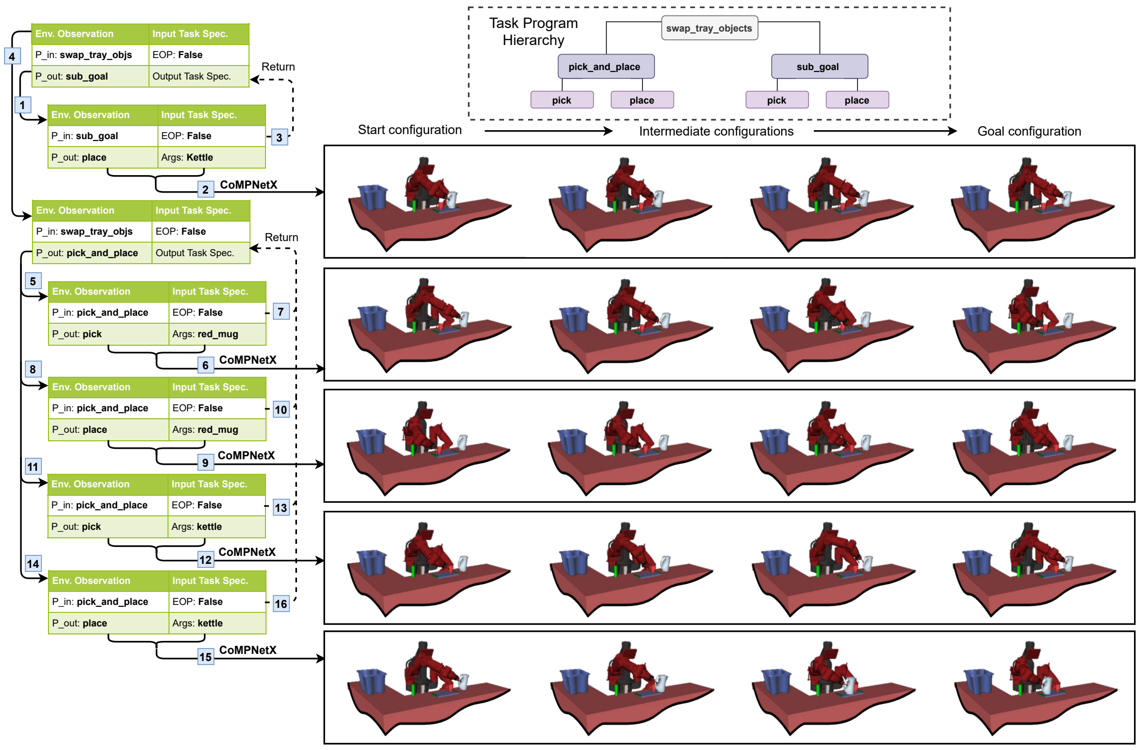


Figure 3.35: CoMPNetX generating motion sequences to swap red mug and kettle in the Bartender scenario 3. The task programmer follows the indicated program hierarchy.

to skip some neurons in the generator network G_ϕ during a forward pass for planning. This process induces a stochastic behavior as in each forward pass, the underlying SMP planner gets a randomly sliced version of CoMPNetX's neural generator. In [GG16], it is shown that Dropout-based slicing of neural networks can model uncertainty in the estimation of their parameters. And our studies show that Dropout can help generate a variety of samples in the subspace of a given configuration space that potentially contains a path solution (Fig. 3.30). Thus, our framework exploits CoMPNetX stochastic behavior to generate informed trees for any underlying SMP planner in the region that potentially contains a path solution for a given problem through our bidirectional and batch planning methods.

An SMP algorithm AL is *probabilistically complete* if the probability of finding a path solution, if one exists, approaches to 1 as the number of samples n in their graph $\mathcal{T}_n$ approaches to ∞. The primary reason for SMP methods to exhibit such completeness is based on their configuration-space sampling strategies that explore the entire space as the number samples n in their graph grows to a large value. There exist a wide range of SMP algorithms that exhibit *probabilistic completeness* and can be merged with projection and continuation-based constraint-adherence approaches for CMP. In conjecture 3.1, we propose that any SMP method that has *probabilistic completeness* guarantees with their traditional sampling techniques will still retain them with our CoMPNetX sampling strategies, i.e.,

Conjecture 3.1 (Probabilistic Completeness) *Given a planning problem* $\{q_{init}, Q_{goal}, \mathcal{X}_{obs}, \mathbf{F}\}$*, and a collision-checker, CoMPNetX will generate samples for an underlying SMP method AL, leading to a tree* $\mathcal{T}_n^{AL}$ *originating at* q_{init} *with number of nodes n, such that the probability of finding a path solution* $\sigma : [0,1] \mapsto \mathcal{M}_{free}$*, if one exists, approaches to one as* $n \to \infty$*, i.e.,* $\mathbb{P}_{n\to\infty}(\mathcal{T}^{AL} \cap Q_{goal} \neq \varnothing) = 1$.

In addition to *probabilistic completeness*, some SMP algorithms also demonstrate *asymptotic optimality*, i.e., as the number of nodes n in the tree $\mathcal{T}_n$ approaches a large value/infinity, the algorithm will find an optimal path with respect to a given cost function $J(\cdot)$, if one exists, with a probability of 1. In the following proposition 2, we claim that with CoMPNetX, the underlying SMP planner will continue to have their *asymptotic optimality* guarantees, i.e.,

Conjecture 3.2 (Asymptotic Optimality) *Given a planning problem* $\{q_{init}, Q_{goal}, \mathcal{X}_{obs}, \mathbf{F}, J\}$*, and a collision-checker, CoMPNetX adaptively generates configuration samples for an asymptotic optimal SMP algorithm such that the solution, if one exists, asymptotically converges to an optimal path solution,* $\sigma^* : [0,1] \mapsto \mathcal{M}_{free}$*, w.r.t* $J(\cdot)$*, as the number of generated samples n approaches to infinity.*

Justifications for Conjectures 3.1 & 3.2 In algorithms 16 and 17, our procedures to merge

CoMPNetX into any SMP planner consist of exploitation and exploration stages. In the former stage, the procedure uses CoMPNetX to adaptively sample the subspace of an implicit manifold configuration space that potentially contains a path solution. And in the latter, it leverages classical sampling techniques that guarantee uniform coverage of the underlying manifolds. These two stages are balanced through a hyperparameter N_{ismp}, which defines the number iterations for which our process relies on exploitation before switching to the exploration stage. Therefore, CoMPNetX with underlying SMP methods do explore the entire configuration spaces as $n \rightarrow \infty$, i.e., $n >> N_{ismp}$. Since CoMPNetX does not alter the internal mechanism of an underlying SMP algorithm and explore the entire state-space over time, it inherits the characteristics of that SMP method, including their *probabilistic completeness* and *asymptotic optimality* and the proofs for our claims remain almost the same as derived in [KMK19].

3.4 Acknowledgements

Chapter 3.1, in part, is a reprint of the following papers. The dissertation author is the primary author of these papers.

- **A.H.Qureshi**, Y.Miao, A.Simeonov, and M.C.Yip,"Motion Planning Networks: Bridging the Gap Between Learning-based and Classical Motion Planners", *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48-66, Feb. 2021, doi: 10.1109/TRO.2020.3006716.

- **A.H.Qureshi**, A.Simeonov, M.J.Bency, and M.C.Yip, "Motion Planning Networks", *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2118-2124, Montreal, Canada, 2019.

- **A.H.Qureshi** and Michael.C.Yip, "Deeply Informed Neural Sampling For Robot Motion Planning", *IEEE International Conference on Intelligent Robot and Systems (IROS)*, pp. 6582-6588, 2018.

Chapter 3.2, in part, is a reprint of L. Li, Y. Miao, **A. H. Qureshi** and M. C. Yip,“MPC-MPNet: Model-Predictive Motion Planning Networks for Fast, Near-Optimal Planning Under Kinodynamic Constraints”, *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4496-4503, July 2021, doi: 10.1109/LRA.2021.3067847. The dissertation author is the co-author of this paper.

Chapter 3.3, in part, is a reprint of the following papers. The dissertation author is the primary author of these papers.

- **A.H.Qureshi**, J.Dong, A.Baig, and M.C.Yip,“Constrained Motion Planning Networks X”, *IEEE Transactions on Robotics*, 2021.
- **A.H.Qureshi**, J.Dong, A.Choe, and M.C.Yip, “Neural Manipulation Planning on the Constraint Manifolds”, *IEEE Robotics and Automation Letters*, 2020.

Chapter 4

Policy Ensemble Composition

In this work, we propose a novel policy ensemble composition method[1] [QJQ$^+$20] that takes the basic, task-agnostic robot policies, transfers them to new complex problems, and efficiently learns a composite model through standard- or hierarchical-RL [SLA$^+$15, SWD$^+$17, HZAL18, VOS$^+$17, FDA17, NGLL18]. This composition model has an encoder-decoder architecture. The encoder is a bidirectional recurrent neural network that embeds the given skill set into latent states. The decoder is a feed-forward neural network that takes the given task information and latent encodings of the skills to output the mixture weights for skill set composition. This dissertation shows that the proposed composition framework can combine the given skills both concurrently (*and* -operation) and sequentially (*or* -operation) as per the need of the given task. The proposed model is evaluated in challenging scenarios including problems with sparse rewards and benchmark it against the state-of-the-art standard- and hierarchical- RL methods. The results show that the proposed composition framework is able to solve extremely hard RL-problems where standard- and hierarchical-RL methods are sample inefficient and either fail or yield unsatisfactory results.

[1]Supplementary material and videos are available at https://sites.google.com/view/compositional-rl

4.1 Related Work

In the past, robotics research has been primarily focused on acquiring new skills such as Dynamic Movement Primitives (DMPs) [SPNI05] or standard reinforcement learning policies. A lot of research in DMPs revolves around learning compact, parameterized, and modular representations of robot skills [SPNI05, INH$^+$13, PDPN13, MHM11]. However, there have been quite a few approaches that address the challenge of composing DMPs in an efficient, scalable manner. To date, DMPs are usually combined through human-defined heuristics, imitation learning or planning [KKGB12, MKP10, AAST12, VV08]. Likewise, RL [SB18] research is also centralized around learning new policies [LHP$^+$15, SLA$^+$15, SWD$^+$17, HZAL18] for complex decision-making tasks by maximizing human-defined rewards or intrinsic motivations [SHM$^+$16, QNYI17, QNYI18, LFDA16].

To the best of the authors' knowledge, there hardly exists approaches that simultaneously combine and transfer past skills into new skills for solving new complicated problems. For instance, [Tod09], [HPZ$^+$18] and [SKTI17] require humans to decompose high-level tasks into intermediate objectives for which either Q-functions or policies are obtained via learning. The high-level task is then solved by merely maximizing the average intermediate Q-functions or combining intermediate policies through temporal-logic. Note that these approaches do not combine task-agnostic skills thus lack generalizability and the ability to transfer skills to the new domains. A recent and similar work to the proposed method is a multiplicative composition policies (MCP) framework [PCZ$^+$19]. MCP comprises i) a set of trainable Gaussian primitive policies that take the given state and proposes the corresponding set of action distributions and ii) a gating function that takes the extra goal information together with the state and outputs the mixture weights for composition. The primitive policies and a gating function trained concurrently using reinforcement learning. In their transfer learning tasks [PCZ$^+$19], the primitive polices parameters are kept fixed, and the gating function is trained to output the mixture weights according to the new goal information. In the ablation studies, we show that training an MCP like-gating function that directly outputs the mixture weights without conditioning on the latent encoding of primitive actions gives inferior performance compared to the proposed method. The proposed method utilizes all information (states, goals, and primitive skills) in a structured way through attention framework, and therefore, leads to better performance.

Recent advancements lead to Hierarchical RL (HRL) that automatically decomposes the complex tasks into subtasks and sequentially solves them by optimizing the given objective function [VOS$^+$17, NGLL18]. In a similar vein, the options framework [SPS99, Pre00] is proposed that solves the given task through temporal abstraction. Recent methods such as option-critic algorithm [BHP17] simultaneously learns a set of sub-level policies (options), their termination functions, and a high-level policy over options to solve the given problem. Despite being an exciting step, the option-critic algorithm is hard to train and requires regularization [VMO$^+$16, HBKP18], or else it ends up discovering options for every time step or a single option for the entire task. In practice, the sub-level options or objectives obtained via HRL are inherently task-specific and therefore cannot be transferred to new domains.

4.2 Background

We consider a standard RL formulation based on Markov Decision Process (MDP) defined by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$, where $\mathcal{S}$ and $\mathcal{A}$ represent the state and action space, $\mathcal{P}$ is the set of transition probabilities, and $\mathcal{R}$ denotes the reward function. At time $t \geq 0$, the agent observes a state $s_t \in \mathcal{S}$ and performs an action $a_t \in \mathcal{A}$. The agent's action a_t transitions the environment state from $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ with respect to the transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$ and leads to a reward $r_t \in \mathcal{R}$.

For compositionality, we extend the standard RL framework by assuming that the agent has access to the finite set of primitive policies $\Pi = \{\pi_i\}_{i=0}^{N}$ that could correspond to agent's skills, controller, or motor-primitives. The presented composition model is agnostic to the structure of primitive policy functions, but for the sake of this work, we assume that each of the sub-policies $\{\pi_i\}_{i=0}^{N}$ solves the MDP defined by a tuple $\{\hat{\mathcal{S}}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}^i\}$. Therefore, $\hat{\mathcal{S}}$, $\hat{\mathcal{P}}$ and $\hat{\mathcal{R}}^i$ are the state-space, transition probabilities and rewards of the primitive policy π_i, respectively. Each of the primitive policies $\pi_i : \hat{S} \times \mathcal{A} \to [0,1], \forall i \in [0,1,\cdots,N]$, takes a state $\hat{s} \in \hat{\mathcal{S}}$ and outputs a distribution over the agent's action space $\mathcal{A}$. The presented composition model is defined as a composite policy $\pi_\theta^c : \mathcal{S} \times \mathcal{A}^{N+1} \times \mathcal{A} \to [0,1]$, parameterize by θ, that outputs a distribution over the action space conditioned on the environment's current state $s \in \mathcal{S}$ and the primitive policies $\{\hat{a}_i \in \mathcal{A}\}_{i=0}^{N} \sim \Pi$. The state space of the composite model is $\mathcal{S} = [\hat{\mathcal{S}}, \mathcal{G}]$. The space $\mathcal{G}$ could include any task specific information such as target locations. Hence, the state inputs to the primitive policies Π and composite policy π_θ^c need not to be the same.

In remainder of this section, the experiments show that the composition model solves an MDP problem. To avoid clutter, it is assumed that both primitive policy ensemble and composite policy have the same state space $\mathcal{S}$, i.e., $\mathcal{G} = \emptyset$. The composition model samples an action from a distribution parameterized by the actions of sub-level policies and the state $s \in \mathcal{S}$ of the environment. Therefore, we can augment the given state space $\mathcal{S}$ as $\mathcal{S}^c : \mathcal{A}^{N+1} \times \mathcal{S}$, where $\mathcal{A}^{N+1} : \{\mathcal{A}^i\}_{i=0}^{N}$ are the outputs of sub-level policies. Hence, compositional MDP is defined as $\{\mathcal{S}^c, \mathcal{A}, \mathcal{P}^c, \mathcal{R}\}$ where $\mathcal{S}^c = \mathcal{A}^N \times \mathcal{S}$ is the new composite state-space, $\mathcal{A}$ is the action-space, $\mathcal{P}^c : \mathcal{S}^c \times \mathcal{S}^c \times \mathcal{A} \to [0,1]$ is the transition probability function, and $\mathcal{R}$ is the reward function for the given task.

4.3 Policy Ensemble Composition

This section presents the policy ensemble composition framework, shown in Fig. 4.1. The composition model consists of i) the encoder network that takes the outputs of primitive policies and embeds them into latent spaces; ii) the decoder network that takes current state s_t of the

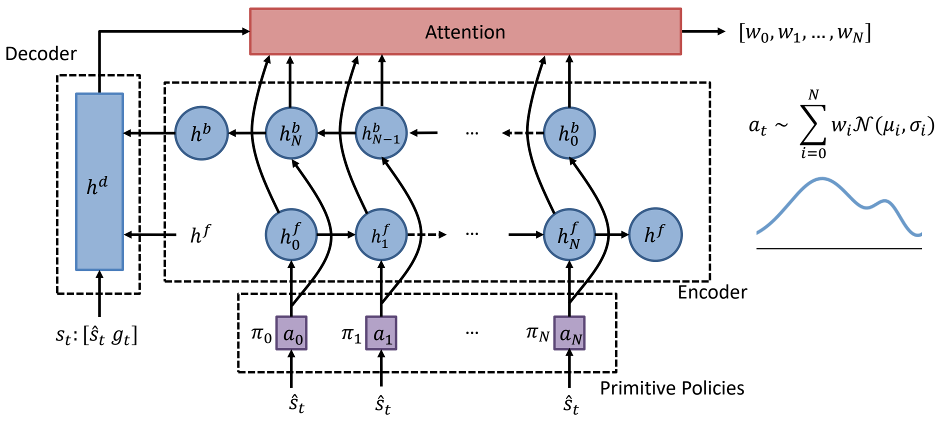


Figure 4.1: Policy ensemble composition model that takes the state information s_t and a set of primitive policies' output $\{\hat{a}_i\}_{i=0}^N$ to compute a composite action a_t.

environment and the latent embeddings from the encoder network to parameterize the attention network; iii) the attention network that outputs the probability distribution over the primitive low-level policies representing their mixture weights. The remainder of the section explains the individual models of the composition framework and the overall training procedure.

Encoder Network

The encoder is a bidirectional recurrent neural network (BRNN) that consists of Long Short-Term Memory units [HS97]. The encoder takes the outputs of the policy ensemble $\{\hat{a}_i\}_{i=0}^N$ and transform them into latent states of forward and backward RNN, denoted as $\{h_i^f\}_{i=0}^{N+1}$ and $\{h_i^b\}_{i=0}^{N+1}$, respectively, where $h_i^f, h_i^b \in \mathbb{R}^d; \forall i \in [0, 1, \cdots, N+1]$. The $N+1$ states of forward and backward RNN corresponds to their last hidden states denoted as h^f and h^b, respectively, in Fig. 4.1.

Decoder Network

The decoder is a simple feed-forward neural network that takes the last hidden states of the forward and backward encoder network, i.e., $\{h^f, h^b\}$, and the current state of the environment s to map them into a latent space $h \in \mathbb{R}^d$. The state input to the decoder network is defined as

$s : [\hat{s}, g]$, where $\hat{s} \in \hat{\mathcal{S}}$ is the state input to the low-level policy ensemble and $g \in \mathcal{G}$ could be any additional information related to the given task, e.g., goal position of the target to be reached by the agent.

Attention Network

The composition weights (see Fig. 4.1) $\{w_i \in [0,1]\}_{i=0}^{N}$ are determined by the attention network as follows:

$$q_i = W^T \cdot \tanh(W_f \cdot h_i^f + W_b \cdot h_i^b + W_d \cdot h); \forall i \in [0, N] \tag{4.1}$$

where $W_f, W_b, W_d \in \mathbb{R}^{d \times d}$ and $W \in \mathbb{R}^d$. The weights $\{w_i\}_{i=0}^{N}$ for the composite policy are computed using gumbel-softmax denoted as softmax(q/T), where T is the temperature term [JGP16].

Composite policy

Given the primitive policy ensemble $\Pi = \{\pi_i\}_{i=0}^{N}$, the composite action is the weighted sum of all primitive policies outputs, i.e., $\pi_\theta^c = \sum_i^N w_i \pi_i$. Since, we consider the primitive policies to be Gaussian distributions, the output of each primitive policy is parameterized by mean μ and variance σ, i.e., $\{\hat{a}_i \sim \mathcal{N}(\mu_i, \sigma_i)\}_{i=0}^{N} \leftarrow \{\pi_i\}_{i=0}^{N}$. Hence, the composite policy can be represented as $\pi_\theta^c = \sum_i^N w_i \mathcal{N}(\mu_i, \sigma_i)$, where $\mathcal{N}(\cdot)$ denotes Gaussian distribution, and $\sum_i w_i = 1$. Given the mixture weights, other types of primitive policies, such as DMPs [SPNI05], can also be composed together by the weighted combination of their normalized outputs.

Composite model training objective

The general objective of RL methods is to maximize the cumulative expected reward, i.e., $J(\pi_\theta^c) = \mathbb{E}_{\pi_\theta^c}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma : (0,1]$ is a discount factor. The policy gradient methods is used to update the parameters θ of the composite model, i.e., $\theta \leftarrow \theta + \eta \bigtriangledown_\theta J(\pi^c_\theta)$, where η is the learning rate. The composite policy can be trained through standard RL and HRL methods, described as follow.

Standard Reinforcement learning

In standard RL, the policy gradients are determined by either on-policy or off-policy updates [LHP+15, SLA+15, SWD+17, HZAL18] and any of them could be used to train composite model. However, in this work, the off-policy soft-actor critic (SAC) method [HZAL18] is used for the training of the policy function. SAC maximizes the expected entropy $\mathcal{H}(\cdot)$ in addition to the expected reward, i.e.,

$$J(\pi^c_\theta) = \sum_{t=0}^{T} \mathbb{E}_{\pi^c_\theta}[r(s_t, a_t) + \lambda \mathcal{H}(\pi^c_\theta(\cdot|s_t))] \tag{4.2}$$

where λ is a hyperparameter. We use SAC as it motivates exploration and has been shown to capture the underlying multiple modes of an optimal behavior. Since there is no direct method to estimate a low-variance gradient of Eqn (4.2), we use off-policy value function-based optimization algorithm. Since, the composite policy is a tractable function $\pi^c_\theta(a_t|s_t, \{\pi_i\}_{i=0}^N)$ parameterized by θ. The composite policy update through SAC requires the approximation of Q- and value-functions. The parametrized value- and Q-function are denoted as $V_\phi(s_t)$ with parameters ϕ, and $Q_\xi(s_t, a_t)$ with parameters ξ, respectively. Since, SAC algorithm build on the soft-policy iteration, the soft value-function $V_\phi(s_t)$ and soft Q-function $Q_\xi(s_t, a_t)$ are learned by minimizing the squared residual error $J_V(\phi)$ and squared Bellman error $J_Q(\xi)$, respectively, i.e.,

$$J_V(\phi) = \mathbb{E}_{s_t \sim \mathcal{M}}[\frac{1}{2}(V_\phi(s_t) - \hat{V}(s_t))^2] \tag{4.3}$$

$$J_Q(\xi) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{M}}[\frac{1}{2}(Q_\xi(s_t,a_t) - \hat{Q}(s_t,a_t))^2] \tag{4.4}$$

where $\mathcal{M}$ is a replay buffer, $\hat{V}(s_t) = \mathbb{E}_{a_t\sim\pi^c_\theta}[Q_\xi(s_t,a_t) - \log\pi^c_\theta(a_t|s_t)]$ and $\hat{Q}$ is the Bellman target computed as follows:

$$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}[V_{\phi'}(s_t+1)] \tag{4.5}$$

The function $V_{\phi'}(s_t)$ is the target value function with parameters ϕ'. The parameters ϕ' are the moving average of the parameters ϕ computed as $\tau\phi + (1-\tau)\phi'$, where τ is the smoothing coefficient. Finally the policy parameters are updated by minimizing the following expected KL-divergence.

$$J_{\pi^c}(\theta) = \mathbb{E}_{s_t\sim\mathcal{M}}\left[D_{KL}\left(\pi^c_\theta(\cdot|s_t)\middle|\middle|\frac{\exp(Q_\xi(s_t,\cdot))}{Z_\xi(s_t)}\right)\right] \tag{4.6}$$

where Z_ξ is a partition function that normalizes the distribution. Since, just-like SAC, the Q-function is differentiable, the above cost function can be determined through a simple reparametization trick, see [HZAL18] for details. Like SAC, we also maintain two Q-functions that are trained independently, and we use the minimum of two Q-functions to compute Eqn. 4.3 and Eqn. 4.6. This way of using two Q-function has been shown to alleviate the positive biasness problem in the policy improvement step. The overall training procedure is summarized in Algorithm 1.

Hierarchical Reinforcement Learning

In HRL, there are currently two streams - task decomposition through sub-goals [NGLL18] and option framework [BHP17] that learns temporal abstractions. In the options framework, the options can be composite policies that are acquired with their termination functions. In task decomposition methods that generate sub-goal through high-level policy, the low-level policy can be replaced with composite policy. In this work, the latter approach [NGLL18], known as HIRO

Algorithm 18: Composition model training using SAC

```
Initialize parameter vectors $\phi$, $\phi'$, $\theta$, $\xi$
**Input:** Primitive policies $\Pi = \{\pi_i\}_{i=0}^{N}$
for each iteration do
    for each environment step do
        Compute primitive policies state $\hat{s}_t \leftarrow s_t \backslash g_t$
        Sample primitive actions $\{\hat{a}_{i,t}\}_{i=0}^{N} \sim \Pi(\hat{s}_t)$
        Sample composite action $a_t \sim \pi_\theta^c(a_t|s_t, \{\hat{a}_{i,t}\}_{i=0}^{N})$
        Sample next state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(s_t, a_t, \{\hat{a}_{i,t}\}_{i=0}^{N}, r_t, s_{t+1})\}$
    for each gradient step do
        Update value function $\phi \leftarrow \phi - \eta \bigtriangledown_\phi J_V(\phi)$
        Update Q-function $\xi \leftarrow \xi - \eta \bigtriangledown_\xi J_Q(\xi)$
        Update policy $\theta \leftarrow \theta - \eta \bigtriangledown_\theta J_{\pi^c}(\theta)$
        $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
```

algorithm, is used to train our policy function.

Like, standard HIRO, we use two level policy structure. At each time step t, the high-level policy $\pi_{\theta'}^{hi}$, with parameters θ', observes a state s_t and takes an action by generating a goal $g_t \in \mathcal{S}$ in the state-space $\mathcal{S}$ for the composite low-level policy $\pi_\theta^{c:low}$ to achieve. The $\pi_\theta^{c:low}$ takes the state s_t, the goal g_t, and the primitive actions $\{\hat{a}_i\}_0^N$ to predict a composite action a_t through which an agent interacts with the environment. The high-level policy is trained to maximize the expected task rewards given by the environment whereas the composite low-level policy is trained to maximize the expected intrinsic reward defined as the negative of distance between current and goal states, i.e., $||s_t + g_t - s_{t+1}||^2$. To conform with HIRO settings, we perform off-policy correction of the high-level policy experiences and we train both high- and low-level policies via TD3 algorithm [FvHM18]. The algorithm to train composite policy through HIRO that employs the two level policy structure is outlined as follows.

The high-level policy generates the sub-goals for the low-level composite policy to achieve. The low-level composite policy also have access to the primitive policy actions. Like HIRO, we use TD3 algorithm [FvHM18] to train both high-level and low-level policies with their corresponding Q-functions, Q^{hi} and Q^{lo}, respectively. The low-level policy $\pi_\theta^{c:low}$, with parameters θ, is trained to maximize the Q-values from the low-level Q-function Q^{lo} for the given state-goal pairs. The Q-function (Q^{lo}) parameters ξ are optimized by minimizing temporal-difference error for the given transitions, i.e.,

$$J_Q^{lo}(\xi) = \big(r^{lo}(s_t, g_t, s_{t+1}) + \gamma Q_\xi^{lo}(s_{t+1}, g_{t+1}, \pi_\theta^{c:lo}(s_{t+1}, g_{t+1}, \{\hat{a}\}_0^N)) - Q_\xi^{lo}(s_t, g_t, a_t)\big)^2 \tag{4.7}$$

where $r^{lo}(s_t, g_t, s_{t+1}) = -\|s_t + g_t - s_{t+1}\|$ and $g_{t+1} \sim \pi_{\theta'}^{hi}(s_{t+1})$.

The high-level policy $\pi_{\theta'}^{hi}$, with parameters θ', is trained to maximize the values of Q^{hi}. The Q-function (Q^{hi}) parameters ξ' are trained through minimizing the following loss for the given transitions.

$$J_Q^{hi}(\xi') = \big(\sum_{t=0}^{c-1} R_t(s_t, a_t, s_{t+1}) + \gamma Q_{\xi'}^{hi}(s_{t+c}, \pi_{\theta'}^{hi}(s_{t+c})) - Q_\xi^{hi}(s_t, \hat{g}_t)\big)^2 \tag{4.8}$$

During training, the continuous adaptation of low-level policy poses a non-stationery problem for the high-level policy. To mitigate the changing behavior of low-level policy, [NGLL18] introduced off-policy correction of the high-level actions. During correction, the high-level policy action g is usually re-labeled with $\hat{g}$ that would induce the same low-level policy behavior as was previously induced by the original high-level action g (for details, refer to [NGLL18]). Algorithm 2 presents the procedure to train the composite policy with HIRO.

4.4 Results

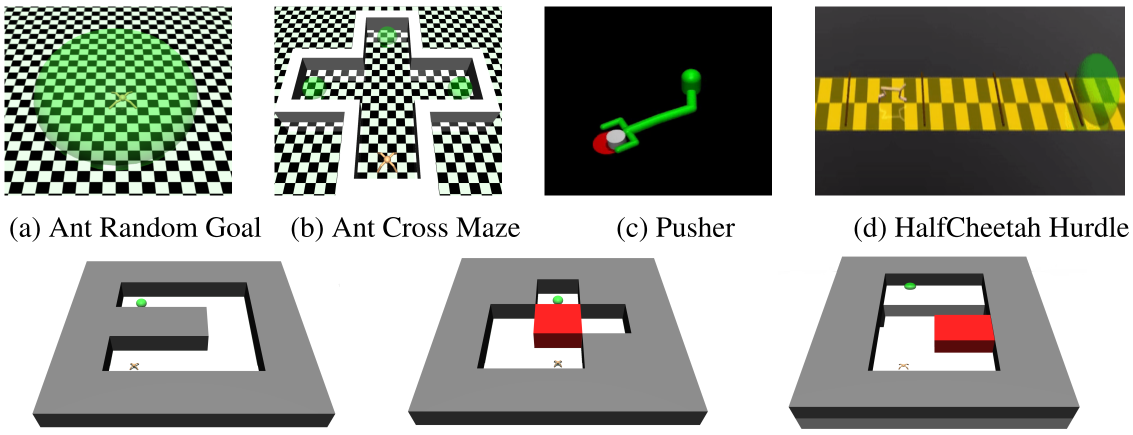
The proposed method is evaluated and compared against standard RL, and HRL approaches in challenging environments (shown in Fig. 4.2) that requires complex task planning and motion control. The implementation details of all presented methods and environment settings are provided in Appendix C of supplementary material. We also do an ablative study in which we take

Algorithm 19: Composition model training using HIRO

```
Initialize parameter vectors $\phi$, $\phi'$, $\theta$, $\xi$
Input: Primitive policies $\Pi = \{\pi_i\}_{i=0}^{N}$
for each iteration do
    for each environment step do
        Compute primitive policies state $\hat{s}_t \leftarrow s_t \backslash g_t$
        Sample primitive actions $\{\hat{a}_{i,t}\}_{i=0}^{N} \sim \Pi(\hat{s}_t)$
        Sample high-level action $g_t \sim \pi^{hi}(\hat{s}_t)$
        Sample composite action $a_t \sim \pi^{c}_{\theta}(a_t|s_t, g_t, \{\hat{a}_{i,t}\}_{i=0}^{N})$
        Sample next state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(s_t, g_t, a_t, \{\hat{a}_{i,t}\}_{i=0}^{N}, r_t, s_{t+1})\}$
    for each gradient step do
        Sample mini-batch with c-step transitions
          $\{(g^k, s^k_{t:t+c}, a^k_{t:t+c-1}, \{\hat{a}^k_{j,t:t+c-1}\}_{j=0}^{N}, r^k_{t:t+c-1})\}_{k=1}^{B} \sim \mathcal{M}$
        Compute rewards for low-level policy $\{r_i^{lo}\}_{i=0}^{B} \leftarrow \{r^{lo}(s_i, g_i, s_{i+1})\}_{i=0}^{B}$
        Update $\pi^{c:lo}$ w.r.t $Q^{lo}$ using $\{(s_k, g_k, a_k, \{\hat{a}_{i,k}\}_{i=0}^{N}, r_k^{lo}, s_{k+1})\}_{k=0}^{B-1}$ [NGLL18]
        Update $\pi^{hi}$ w.r.t $Q^{hi}$ using $\{(s_k, \hat{g}_k, \sum_{i=k}^{c-1} r_k, s_{k+c})\}_{k=0}^{B-1}$ [NGLL18]
```

away different components of our composite model to highlight their importance. Furthermore, we depict attention weights of our model in a navigation task to highlight its ability of concurrent and sequential composition.

We consider the following seven environments for our analysis: (1) **Pusher:** A simple manipulator has to push an object to a given target location. (2) **Ant Random Goal:** In this environment, a quadruped-Ant is trained to reach the randomly sampled goal location in the confined circular region. (3) **Ant Cross Maze:** The cross-maze contains three target locations. The task for a quadruped Ant is to reach any of the three given targets by navigating through a 3D maze without collision. (4) **HalfCheetah Hurdle:** In this problem, the task for a halfcheetah is to run and jump over the three barriers to reach the given target location. (5) **Ant Maze:** A ⊃-shaped maze poses a challenging navigation task for a quadruped-Ant. In this task, the agent is given random targets all along the maze to reach while training. However, during the evaluation, we test the agent for reaching the farthest end of the maze. (6) **Ant Push:** A challenging environment


(a) Ant Random Goal (b) Ant Cross Maze (c) Pusher (d) HalfCheetah Hurdle

(e) Ant Maze (f) Ant Push (g) Ant Fall

Figure 4.2: Benchmark control and manipulation tasks requiring an agent to reach or move the object to the given targets (shown in red for pusher and green for rest).

that requires both task and motion planning. The environment contains a movable block, and the goal region is located behind that block. The task for an agent is to reach the target by first moving to the left of the maze so that it can move up and right to push the block out of the way for reaching the target. (7) **Ant Fall:** A navigation task where the target is located across the rift in front of the agent's initial position. There also happen to be a moveable block, so the agent has to move to the right, push the block forward, fill the gap, walk across, and move to the left to reach the target location. (8) **Multi-goal Point Mass:** In this scenario, the task is to navigate a point-mass to one of the four goals located diagonally to agent initial position.

In all tasks, the primitive skills of the agent are obtained for the composite policy. For Ant, the four basic policies for moving left, right, up, and down. The pusher uses two primitive policies that are to push an object to the left and down. In HalfCheetah hurdle environment, the low-level policies include jumping and running forward. Finally for the point-mass robot, the composition model takes four policies for moving in the up, down, left and right directions. Furthermore, in all environments, except pusher, the primitive policies were agnostic of high-level tasks (such as target locations) that were therefore provided separately to the composite model via decoder network. This highlights the ability of this model to transfer basic robot skills to novel problems.

Table 4.1: Performance comparison of presented composition model against SAC [HZAL18], TRPO [SLA+15], and PPO [SWD+17] on benchmark control tasks in terms of normalized distance (lower the better) of an agent from the given target.

Methods	Environments			
	Ant Random Goal	Ant Cross Maze	Pusher	HalfCheetah Hurdle
SAC	0.21 ± 0.08	0.78 ± 0.06	0.17 ± 0.02	0.79 ± 0.01
TRPO	1.09 ± 0.15	0.85 ± 0.15	0.64 ± 0.09	0.87 ± 0.05
PPO	1.06 ± 0.11	0.95 ± 0.07	0.71 ± 0.06	0.88 ± 0.04
Our Method	0.11 ± 0.05	0.11 ± 0.02	0.14 ± 0.02	0.27 ± 0.22

Comparative study

In the comparative studies, the test environments are divided into two groups. The first group includes Pusher, Random Goal Ant, Ant Cross Maze, and HalfCheetah-Hurdle environments, whereas the second group comprises the remaining environments that require task and motion planning under weak reward signals.

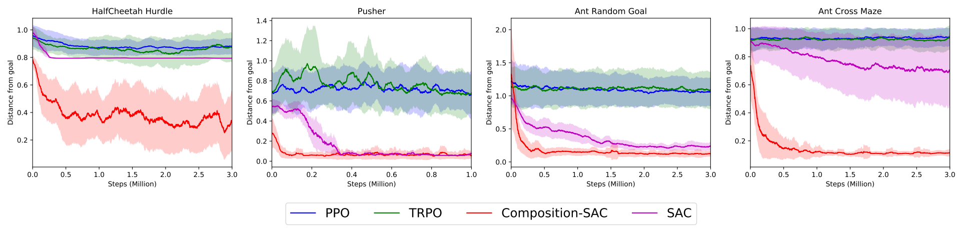


Figure 4.3: Comparison results of our composition method against standard RL methods averaged over ten trials. The axes represents the distance of the agent/object from the target and environment steps in millions.

In the first group of settings, the composite model trained with SAC [HZAL18] against the standard Gaussian policies obtained using SAC [HZAL18], PPO [SWD+17], and TRPO [SLA+15]. The HRL methods are excluded in these cases as the environment rewards sufficiently represent the underlying task, whereas HRL approaches are applicable in cases that have a weak reward signal or require task and motion planning. Table 4.1 presents the mean and standard deviation of the agent's final distance from the given targets after the end of an evaluation rollout over the ten trials. Fig. 4.3 shows the mean learning performance over all trials during the three million training steps. In these set of problems, TRPO and PPO entirely fail to reach the goal, and

SAC performs reasonably well but only in simple Ant Random Goal and Pusher environments as it fails in other cases. The composite policy obtained using SAC successfully solves all tasks and exhibit high data-efficiency by learning in merely a few thousand training steps.

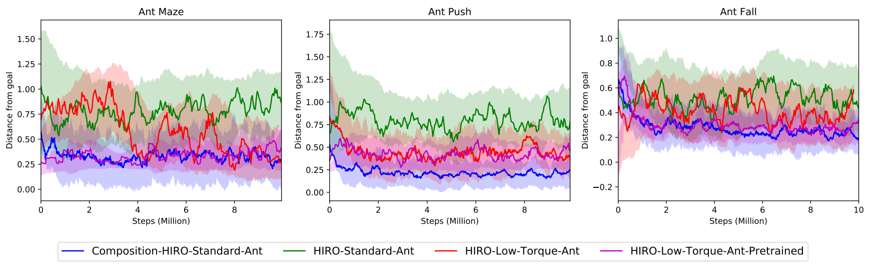


Figure 4.4: Comparison of composition model trained with HIRO and three variants of standard HIRO. The pretrained HIRO undergoes extra 4 million training steps. The standard- and low-torque-Ant has 150 and 30 units torque limit.

In the second group of environments, the distance-based rewards are used that are weak signals as greedily following them does not lead to solving the problem. Furthermore, in these environments, policies trained with standard RL, including the composite policy, failed to solve the problem even after 20 million training steps. Therefore, the composite policy trained with HIRO [NGLL18] and compared its performance against standard HIRO formulation [NGLL18]. We also tried to include option-critic framework [BHP17], but we were unable to get any considerable results with their online implementation despite several attempts with the parameter tuning. One of the reasons option-critic fails is because it relies purely on task rewards to learn, which makes them inapplicable for cases with weak reward signals [NGLL18].

[NGLL18] use a modified Ant in their paper that has 30 units joint torque limit (low-torque-Ant). For our composition method, we use Mujoco standard Ant that has a torque limit of 150 units which makes the learning even harder as the Ant is now more prone to instability than a low-torque-Ant. For standard HIRO formulation, we trained three variants i) HIRO trained on standard Mujoco Ant, ii) HIRO trained on low-torque Ant (as in [NGLL18]); iii) HIRO pretrained for the equal number of training steps as used to train the primitive policies of our composition method on a low-torque-Ant (For more details, refer to Appendix C.1).

Fig. 4.4 shows the learning performance, averaged over ten trials, during 10 million steps.

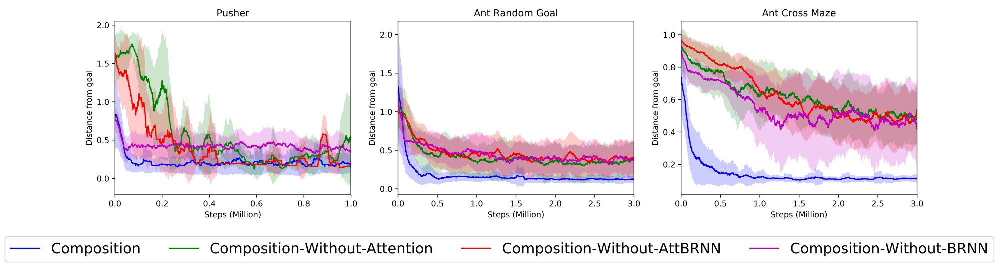


Figure 4.5: Ablative Study: Performance comparison, averaged over ten trials, of our composite model and its ablated variations that lack attention model, bidirectional-RNN (BRNN) or both attention and BRNN (AttBRNN).

Figure 4.6: The attention weights: The blue and yellow colors show low and high values, respectively

It can be seen that the composite policy with HIRO outperforms standard HIRO [NGLL18] by a significant margin. It also certifies the utility of solving RL tasks using composition by leveraging basic pre-acquired skills. Furthermore, HIRO performs poorly with standard Ant, even pretraining did not improve the performance, as it imposes a harder control problem.

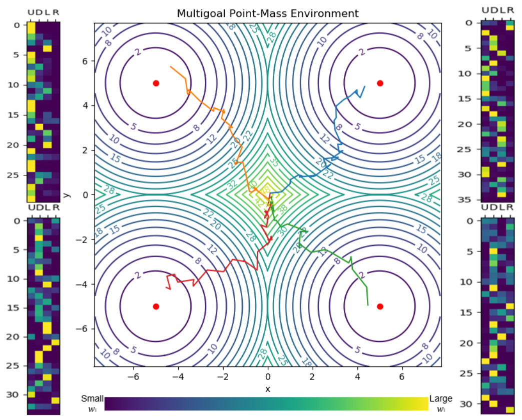


Figure 4.7: Each path corresponds to its adjacent attention weight mapping. The weighting “strength” of each primitive policy is depicted for each step (i.e. up (U), down (D), left (L), and right (R)).

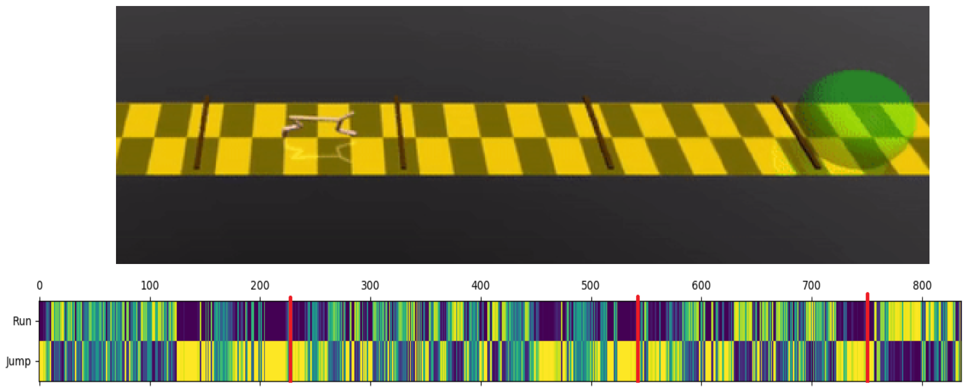


Figure 4.8: The depiction of attention weights for the halfcheetah-hurdle environment. The weighting “strength” of each primitive policy (i.e., run and jump) for each step is shown.

Ablative study

We remove bidirectional RNN (BRNN) (similar to [PCZ$^+$19]), attention-network, and both attention-network and BRNN (AttBRNN) from our composition model to highlight their importance in the proposed architecture in solving complex problems. We train all models with SAC [HZAL18]. The first model is our composite policy without attention in which the decoder network takes the state information and last hidden states of the encoder (BRNN) to directly output actions rather than mixture weights. The second model is without attention network and BRNN, it is a feed-forward neural network that takes the state information and the primitive actions and predicts the action to interact with the environment. The third model is without BRNN, it is a feed-forward neural network that takes the state and goal information, and output the mixture weights. The mixture weights are then used to combine the actions from primitive policies. This setting is same as [PCZ$^+$19] for their transfer learning problems. Fig. 4.5 shows the mean performance comparison, over ten trials, of our composite model against its ablated versions on a Ant Random Goal, Cross Maze Ant, and Pusher environment. We exclude remaining test environments in this study as ablated models completely failed to perform or show any progress.

Note that the better performance of our method compared to ablated versions highlight the merits of our architecture design. Intuitively, BRNN allows the dynamic encoding of a skill set that could be of variable lengths[2]. And our decoder network uses the encoded skills together with given state and goal information (states, goals, and primitive skills) in a structured way using attention framework and provides significantly better performance than other ablated models, including [PCZ+19].Furthermore, another merit of using the attention network is that it bypasses the complex transformation of action embeddings (composition-without-attention) or actions and state-information (composition-without-AttBRNN) directly to action space. Hence, the proposed architecture is crucial for the composition of task-agnostic sub-level policies to solve new problems.

Depiction of attention weights

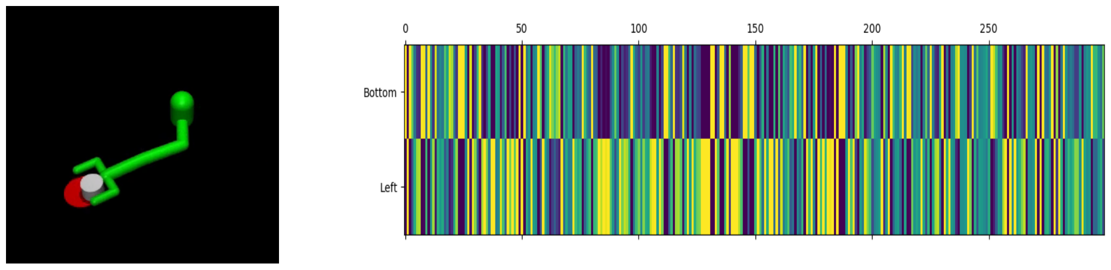


Figure 4.9: The depiction of attention weights for the pusher environment. The weighting "strength" of each primitive policy (i.e., Bottom and Left) for each step is shown.

In order to further assess the merit of utilizing an attention network, we apply our model to a simple 2D multi-goal point-mass environment as shown in Fig. 4.7. The point-mass is initialized around the origin (with variance $\sigma^2 = 0.1$) and must randomly choose one of four goals to reach. For this experiment we use dense rewards with both a positional and actuation cost. Primitive policies of *up* $(+y)$, *down* $(-y)$, *left* $(-x)$, and *right* $(+x)$ were trained and composed to reach goals, represented here as red dots, in the "diagonal" directions where a combination of two or more primitive policies are required to reach each goal.

The four mappings in the figure give us insight into how the attention network is utilizing the given primitives to achieve the desired task. At each step in a given path, the weights $\{w_i\}_{i=0}^{N}$

[2]However, in this work, we only consider a primitive skill set of fixed size

for each low-level policy are assigned and composed together to move the point-mass in the desired direction. We see here that even with some noise and short-term error, the attention weights are strongest for primitive policies that move the point-mass to its chosen goal. We also see that multiple policies are activated at once to achieve more direct movements toward the goal, as opposed to "stair-stepping" where only one primitive is activated at a time. Both of these observations point to the concurrent and sequential nature of this composition model.

We further illustrate the attention weights of our composite policy in halfcheetah-hurdle (Fig. 4.8), pusher (Fig. 4.9), and cross-maze-ant (Fig. 4.10) environments. Our results highlight the ability of our model to compose a given set of primitive polices, both sequentially and concurrently, to solve the given problem. Fig. 4.6 shows the color-coding for the scale of attention weight value given by the composite policy.

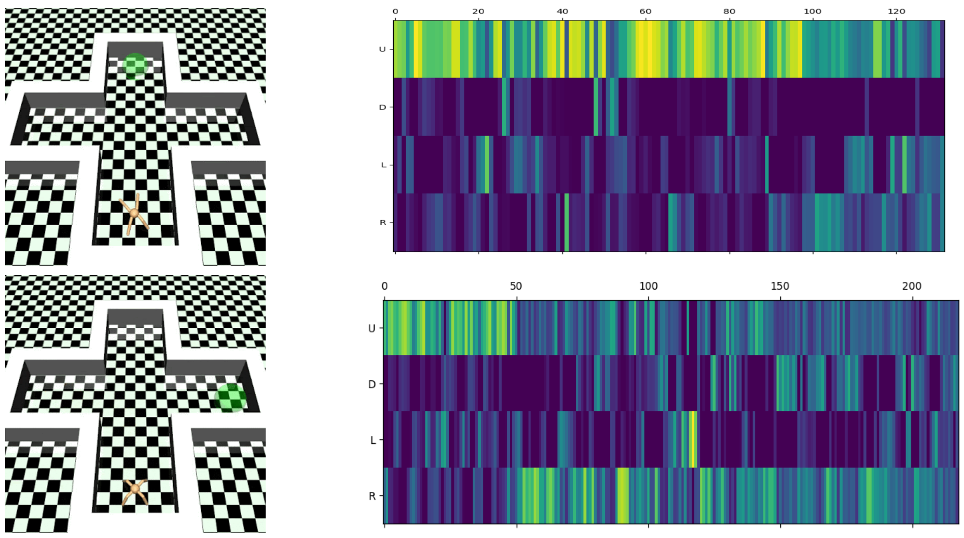


Figure 4.10: The depiction of attention weights for the cross-maze environment. The weighting "strength" of each primitive policy (i.e., up (U), down (D), left (L), and right (R)) for each step is shown.

Apart from compositionality, another research problem in the way of building intelligent machines is the autonomous acquisition of new skills that were lacking in the system and therefore, hindered it from the solving the given task [LUTG17]. In this section, we demonstrate that our composition model holds the potential for building such a system that can simultaneously learn

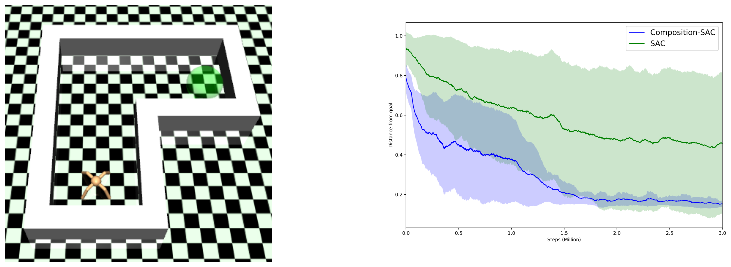


Figure 4.11: Missing skill. The composite model with a primitive policy for moving right and a trainable policy function. The composition framework trained the new function to move in the upward direction to reach the given goal.

new missing skills and compose them together with the existing skill set to solve the given problem.

Fig. 4.11 shows a simple ant-maze environment in which the 3D quadruped-Ant has to reach the target, indicated as a green region. In this problem, we equip our composition model with a single primitive policy for walking in the right direction and a trainable, parametric policy function. The trainable policy function takes the state without goal information and outputs the parameters for Gaussian distribution to model the robot's action space. Note that the composite model requires a skill of walking upward in addition to moving in the right direction to reach the given goal. To determine if our composition model can teach the new parametric policy function to move upward, we trained our composition model along with the new policy using the shared Q- and value-functions. Once trained, we observed that our composition model learned the parametric policy function for walking in an upward direction with slight right-turning behavior.

The plot in Fig. 4.11 shows the performance of our composition model and standard RL policy obtained with SAC in this environment. The vertical axis indicates the distance from the target location averaged over five randomly seeded trials. The horizontal axis shows the number of environment steps in millions. It can be observed that our method converged faster than standard RL policy. It shows that our approach is also utilizing the existing skills and therefore learns faster than a standard policy function that solves the entire task from scratch. Furthermore, in current settings, we do not impose any constraint on the new policy function that would make it learn only the missing skills rather than replicating existing skills or solving the given task entirely by itself. We leave the formulation of such constraint for autonomous acquisition of new, missing skills to our future works.

4.5 Implementation details

Pretraining Benchmark Method

In our results, we also consider the pretraining of benchmark methods for an equivalent amount of time as used to train the primitive policies for our composition model for a better analysis. The total number of steps needed to train primitive skills of Ant (up, down, left, and right), halfcheetah (run and jump), and pusher (down and left) were 1, 0.5, and 0.1 million, respectively. In Fig. 4.4, HIRO was pretrained for both standard (30 units) and low-torque-ant (150 units). The results of HIRO standard-ant are excluded in Fig.4.4 to avoid clutter as they led to no improvement in the performance. For the non-hierarchical tasks presented in Fig. 4.3, the benchmark methods are not pretrained. However, to account for pretraining, the performance of other methods (TRPO, PPO, SAC) can be accessed after 1, 0.5, and 0.1 million for the ant, halfcheetah, and pusher environments, respectively. Also, notice that in Fig. 4.3, the pretraining will have no significant effect on the performance of TRPO and PPO in all environments and SAC in half-cheetah hurdle and cross-maze-ant environments. Furthermore, since pusher and random-goal-ant are relatively simple environments (due to no obstacles or maze around), pretrained SAC can perform similar to our composition method.

Environment Details

In this section, we present the environment details including reward functions, primitive policies, and state space information. The reward functions are presented in the Table 4.3 together with the overall reward scaling values. For Ant and halfcheetah-hurdle environments, the primitive policies were trained in a free-space that had no mazes, hurdle or fixed targets in the surrounding.

Ant environments: In these environments, we use 8 DOF four-legged Ant with 150 units torque limit. The primitive policies of moving left, right, down and up were shared across all these tasks. These sub-level-policies were obtained using SAC and each policy was trained for about 0.25 million steps. In these environments, the information g in the state $s : [\hat{s}, g]$ corresponds to the target location. Let us introduce the notation to defined reward function. Let r_{xy}, g_{xy}, u, and f_c denote xy-position of the robot's torso, xy-position of the goal, joint torques, and contact-cost, respectively. The scaling factors are defined as λ. The reward function for the following environments is defined as with reward scaling of 5 units:

$$-\lambda_g ||r_{xy} - g_{xy}||^2 + \lambda_v v_{xy} + \lambda_s I(\text{IsAlive}) - \lambda_{ct} ||u||^2 - \lambda_c f_c \tag{4.9}$$

Ant Random Goal: In this environment, the ant has to navigate to any randomly sampled target within the confined circular region of radius 5 units. The goal radius is defined to be 0.25 units. The reward function coefficients λ_g, λ_v, λ_s, λ_{ct}, and λ_c are 0.3, 0.0, 0.05, 0.01, and 0.001, respectively.

Ant Cross Maze: In this environment, the ant has to navigate through the 3D maze to reach any of the target sampled from the three targets. The goal radius is defined to be 1.0 units. The reward function parameters are same as for the random-goal ant environment.

For the remaining environment (Ant Maze, Ant Push and Ant Fall), we use the following reward function with no reward scaling:

$$\lambda_g ||r_{xyz} - g_{xyz}||^2 - \lambda_{ct} ||u||^2 - \lambda_c f_c \tag{4.10}$$

where coefficients λ_g, λ_{ct}, and λ_c are set to be 1.0, 0.05, and 0.5×10^{-4}.

Ant Maze: In this environment, we place the Ant in a $\supset$-shaped maze for a navigation task between given start and goal configurations. The goal radius is defined to be 5 units. During training, the goal is uniformly sampled from $[-4,20] \times [-4,20]$ space, and the Ant initial location is always fixed at $(0,0)$. During testing, the agent is evaluated to reach the farthest end of the maze located at $(0,19)$ within L2 distance of 5.

Ant Push: In this environment, the Ant is initially located at $(0,0)$ coordinate, the moveable block is at $(0,8)$, and the goal is at $(0,19)$. The agent is trained to reach randomly sampled targets whereas during testing, we evaluate the agent to reach the goal at $(0,19)$ within L2 distance of 5.

Ant Fall: In this environment, the Ant has to navigate in a 3D maze. The initial agent location is $(0,0)$, and a movable block is at $(8,8)$ at the same elevation as Ant. Their is a rift in the region $[-4,12] \times [12,20]$. To reach the target on the other side of the rift, the Ant must push the block down into the rift, and then step on it to get to the goal position.

Table 4.2: Hyperparameters

Parameters	SAC	HIRO	TRPO	PPO
Learning rate (η)	3×10^{-4}	1×10^{-4}	-	-
Discount factor (γ)	0.99	0.99	0.99	0.99
Nonlinearity in feedforward networks	ReLU	ReLU	ReLU	ReLU
Minibatch samples size	256	128	-	-
Replay buffer size	10^6	2×10^5	-	-
Batch-size	-	-	1000	1000
Target parameters smoothing coefficient (τ)	0.005	0.005	-	-
Target parameters update interval	1	2	-	-
Gradient steps	1	1	0.01	0.01
Gumbel-softmax temperature (T)	0.5	0.5	-	-

Pusher: In pusher environment, a simple manipulator has to move an object to the target location. The primitive policies were to push the object to the bottom and left. These low-level policies were obtained via SAC after 0.1 million training steps. In this environment, the state information for both primitive policies and the composite policy include the goal location. Therefore, $\mathcal{G}$, in this case, is null. The reward function is given as:

$$-\lambda_g||o_{xy} - g_{xy}||^2 - \lambda_o||r_{xy} - o_{xy}||^2 - \lambda_{ct}||u||^2 \tag{4.11}$$

where o_{xy}, g_{xy}, r_{xy}, and u are xy-position of object, xy-position of goal, xy-position of arm, and

Table 4.3: Network Architectures.The right most column shows the hidden units per layer.

	Model Architectures	Hidden units
Composition-HIRO	High-level Policy: Three layer feed forward network	300
	Encoder Network: Bidirectional RNN with LSTMs	128
	Decoder Network (Single layer feed forward network)	128
	Attention Network: $W_f, W_b, W_d \in \mathbb{R}^{d\times d}; W \in \mathbb{R}^d$	128
Composition-SAC	Encoder Network: Bidirectional RNN with LSTMs	128
	Decoder Network (Single layer feed forward network)	128
	Attention Network: $W_f, W_b, W_d \in \mathbb{R}^{d\times d}; W \in \mathbb{R}^d$	128
HIRO	High-level Policy: Three layer feed forward network	300
	Low-level Policy: Three layer feed forward network	300
Standard RL policy	Two layer feed forward network	256

joint-torques. The coefficients λ_g, λ_o, and λ_{ct} are 1.0, 0.1, and 0.1, respectively.

Halfcheetah-hurdle: In halfcheetah-hurdle environment, a 2D cheetah has to jump over the three hurdles to reach the target. The primitive policies include run forward and jump where each poilcy was trained with SAC for 0.25 million steps. Furthermore, in this environment, the information g in the state $s : [\hat{s}, g]$ corresponds to the x-position of the next nearest hurdle in front of the agent as well as the distance from that hurdle. The reward function is defined as:

$$-\lambda_g ||r_{xy} - g_{xy}||^2 - \lambda_{hc} hc(\cdot) + \lambda_{rg} I(\text{goal}) + \lambda_z |v_z| + \lambda_v v_x - \lambda_c cc(\cdot) \tag{4.12}$$

where r_{xy}, g_{xy}, v_z, and v_x are xy-position of robot torso, xy-position of goal, velocity along z-axis, and velocity along x-axis, respectively. The function $hc(\cdot)$ returns a count indicating the number of hurdles in front of the robot. The indicator function $I(\text{goal})$ returns 1 if the agent has reached the target otherwise 0. The function $cc(\cdot)$ is a collision checker which returns 1 if the agent collides with the hurdle otherwise 0. The reward function coefficients λ_g, λ_{hc}, λ_{rg}, λ_z, λ_v, and λ_c are 0.1, 1.0, 1000, 0.3, 1.0 and 2, respectively.

Hyperparameters and Network Architectures

Table 4.2 summarizes the hyperparameters used to train policies with SAC [HZAL18], TRPO [SLA$^+$15], PPO [SWD$^+$17], and HIRO [NGLL18]. Table 4.3 summarizes the network architectures.

4.6 Acknowledgements

Chapter 4, in part, is a reprint of **A.H.Qureshi**, J. J. Johnson, Y. Qin, T. West, B. Boots, and M.C.Yip. "Composing Task-Agnostic Policies via Deep Reinforcement Learning", *International Conference on Representation Learning (ICLR)*, 2020. The dissertation author is the primary author of this paper.

Chapter 5

Task Planning

Many solutions exist that can solve rearrangement planning problems, particularly throughout the sub-field of task-and-motion planning (TAMP) [GCH$^+$20], these often come with a wide range of significant limitations that restrict their utility in the real world. In particular, these approaches often rely on having known object models.

However, recent advances in deep learning have provided ways for us to weaken these assumptions. Work on deep learning for grasping gives us the ability to grasp unknown objects [MML$^+$17, MEF19], including in cluttered scenes [MME$^+$20, SMTF21]. A growing body of work also looks at using learned representations for robotic task or motion planning [SJA$^+$18, HLF$^+$19, PBK$^+$19, ISL20]. Others have learned samplers for integration into traditional motion planners, e.g. [QMSY20, QDCY20]. We can also more accurately segment unknown objects from the world, giving us ways to identify and pick up objects that we have never seen before [XXMF20a, XXMF20b]. Recently, graph neural networks have also been used to represent 3D scenes [BFM$^+$20].

One question that remains, though, is how we can plan to execute sequences of actions including grasps and placements, in order to perform long-horizon rearrangement tasks in unknown environments. Multiple objects might block one another or prevent things from being moved into new positions. In particular, we need to make decisions about which objects to grasp or move when there are multiple options that are viable at any given time.

In this work, we describe **Ne**ural **R**earrangement **P**lanning (NeRP) [AQF21], an approach for rearranging unknown objects from perceptual data in the real world, as shown in Fig. 5.2. NeRP represents a scene as a graph over segmented objects. Given a current and the goal scene and object segmentation, it finds an alignment between the two sets of unknown objects in the goal and current image using pre-trained ResNet50 [HZRS15] features, and use them to output multi-step rearrangement actions. NeRP is trained using the synthetically generated data in simulation for random object rearrangement.

It uses learned neural networks for choosing the object that needs to be picked and the distribution of possible placements in terms of relative transforms from the current position of the object in the point-cloud space. These operations can be sequenced over time via a sampling-based planning algorithm, which allows us to choose sequences of manipulations in order to perform rearrangement tasks with unknown objects. At execution time, we use model-free grasping [SMTF21] to pick and use MPPI controller [DMEF21] to place objects and thereby achieve the specified goal. After each placement action, NeRP observes the scene and re-plans the best course of actions to account for inaccuracies in execution. Fig 5.1 shows execution of object rearrangement on unseen objects with real robot using NeRP.

To our knowledge, this is the first system for end-to-end rearrangement planning for unknown objects. Specifically our contributions are:

- a graph neural network approach for computing which objects to move and estimating where they can be placed to complete a rearrangement task,
- an algorithm for planning and execution that will reactively select where to place these unknown objects in order to complete the rearrangement task.

In addition, we show results both on a simulated object rearrangement task and via real-world

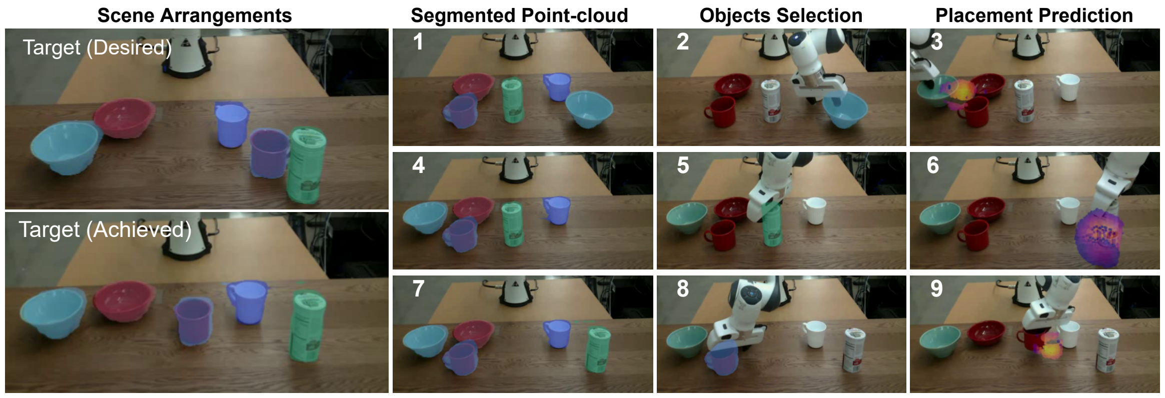


Figure 5.1: NeRP finds a sequence of pick and place operations to rearrange unknown objects to their target arrangement (shown in the top left), choosing both which object to manipulate and where to place it.

robot execution.

5.1 Related Works

Perhaps the most relevant area of work to rearrangement planning is the field of task and motion planning (TAMP). TAMP is a broad area of study that looks at integrating discrete high-level planning (which objects to grab, which actions to execute) with continuous low level planning (like which positions in which to place objects) [GCH+20]. This area generally relies on model-based methods, using known objects and fully-observable domains, although work has been done to weaken these assumptions, particularly in recent years. For example, recent work has looked at task and motion planning with partial observability [GPLP+20], enabling re-arrangement even if certain objects are not fully visible, and others have used learning to guide task and motion planning [KWKLP19].

Rearrangement planning is a common subset of task and motion planning [CHES11, KRS17, NLC+19, LZK+20, GCH+20]. It has recently been identified as an interesting challenge area for robotics research [BCC+20]. It is interesting in part because it involves long-term planning; recent methods often use Monte Carlo Tree Search or similar methods to explore multiple future possibilities, e.g. [KRS17, LZK+20]. Partial observability is a serious issue, as in many cases objects are partly or wholly occluded [NLC+19], which necessitates special considerations [GPLP+20].

One possible route to building on these is via Motion Planning Networks (MPNet) [QMSY20], which predict a set of waypoints that you can use for classical planning based on sensor data. Neural Task Graphs [HNX+19] propose a way to perform multi-step planning with known objects, but require a demonstration of the correct action sequence. Another approach is via Deep Affordance Foresight [XMMM+20], which learns predictive affordance models to allow completion of longer horizon tasks.

Visual Robot Task Planning [PBK+19] learns an autoencoder style representation for multi-step task planning, but does not generalize to different goals. Similarly, PlaNET works via deep visual predictions, showing what the possible consequences are for near-horizon actions [HLF+19]. Universal Planning Networks use a learned latent space for motion planning together with a simple gradient descent planner [SJA+18]. Other work like Q-MDP net for planning under partial observability [KHL17], focusing on navigation instead of placement and rearrangement.

Broadly Exploring Local Policy Trees break up task planning into many different learned high level tasks [ISL20], and use an RRT-like approach to navigate between these via a latent space. However, this approach is not applied to the real world in the same way as our approach is, and as it uses a learned latent space it is most likely less general. Simeonov et al. [SDK+20] propose a method that might be most similar to ours, which looks at a whole sequence of manipulation skills to do task and motion planing with unseen rigid objects. However, they assume a high level task skeleton is given.

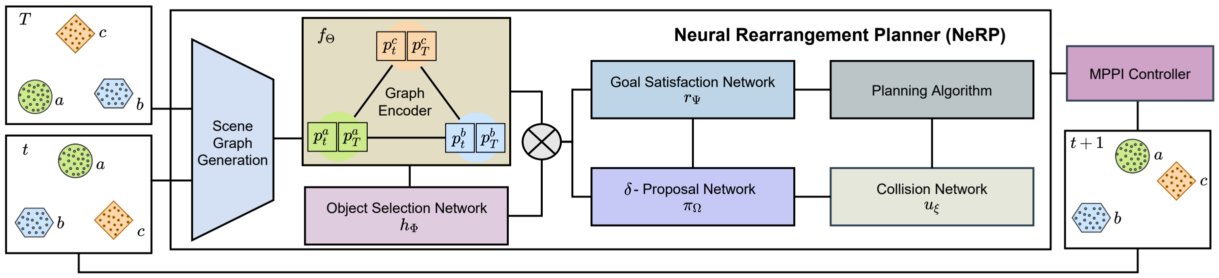
5.2 Problem Definition

Let $X = \{x_1, x_2, \cdots, x_n\} \in \mathcal{X}$ denote an unordered set of n points each of dimension d. Let $\mathcal{M} \in \mathbb{R}^n$ be a point-wise selection operator, i.e., $\mathcal{M} \times \mathcal{X} \mapsto \mathcal{X}'$, over a given set, where $\mathcal{X}' \subset \mathcal{X}$, and $\Delta \in \mathbb{R}^d$ be a set action operator such that $\mathcal{X} \times \Delta \mapsto \mathcal{X}$. Therefore, a set-based planner can be defined as a new class of function $\Pi : \mathcal{X} \mapsto \mathcal{M} \times \Delta$ that determines a sequence of set operators in $\mathcal{M} \times \Delta$ space to point-wise transform a given set to another desired set. In this work, we propose NeRP, a set-based planner that outputs a sequence of actions $\{(M_0, \delta_0), (M_1, \delta_1), \cdots, (M_H, \delta_H)\} \in \mathcal{M} \times \Delta$ for the initial $X(t)$ and target $X(T)$ unordered sets such that $X(t) \times M \times \delta \mapsto X(T)$, where $M \in \mathcal{M}$ and $\delta \in \Delta$. We demonstrate the application of our method to rearrangement planning problems for robotics.

Let $\mathcal{Q}$ and $\mathcal{A}$ be the robot configuration and action spaces, respectively. A low-level agent can be defined as a function $\pi^L : \mathcal{Q} \mapsto \mathcal{A}$ that achieves the given robot configurations $q \in \mathcal{Q}$ by executing a sequence of actions $a_{\{m\}} = \{a_1, a_2, \cdots, a_m\} \in \mathcal{A}$. In our case, we consider the general robot interaction setting, where for a given current $X(t)$ and target $X(T)$ sets, the high-level agent, $\pi^H \in \Pi$, at time t, outputs a set action $(M(t), \delta(t)) \in \mathcal{M} \times \Delta$, leading to an achievable sub-goal set $q_\delta \subset \mathcal{Q}$ for the low-level agent. The low-level agent π^L executes a sequence of actions $a \in \mathcal{A}$ to achieve (M_0, δ_0) in the environment and return the control to task planner where it gets new set of observation $X(t+1)$ and replans accordingly.

5.3 Neural Rearrangement Planner (NeRP)

This section formally presents our approach for Neural Rearrangement Planning (NeRP), which comprises four main components: the object alignment network, the object selection and objects placement prediction networks, and the collision detection network. Fig. 5.2 shows an overview of this model architecture. We use these networks to perform multi-step planning in order to perform object rearrangement in unknown scenes.



T
c
a
b
t
a
b
c
Scene Graph Generation
f_Θ
p_t^c p_T^c
Graph Encoder
p_t^a p_T^a
p_t^b p_T^b
Object Selection Network h_Φ
Neural Rearrangement Planner (NeRP)
Goal Satisfaction Network r_Ψ
Planning Algorithm
δ- Proposal Network π_Ω
Collision Network u_ξ
MPPI Controller
$t+1$
a
c
b

Figure 5.2: NeRP's model architecture overview. It comprises scene graph generation and encoding. The encoding is used to select objects and compute their relative translations for rearrangement planning tasks.

Neural Networks

Objects Alignment and Graph Generation: The objects alignment module determines the individual objects' correspondence in the current $X(t)$ and target $X(T)$ scene point-clouds for the graph generation. We use Unknown Objects Instance Segmentation (UCN) [XXMF20a] to extract specific object, i, RGB-information from the given scene and it's corresponding point clouds $X^i \subset X$. The RGB information for each individual objects are passed through pre-trained Resnet model [HZRS15] to extract their latent features $w \in W$ for computing L2 norm based objects similarity scores s, an $N \times N$ matrix with $N \in \mathbb{N}$ denoting the number of instance segmentations in the current and target scenes.

The graph generator takes the current and target scene point clouds with their segmentation labels and the similarity matrix s as an input and generates an undirected graph $G = (V, E) \in \mathcal{G}$. Each vertex $V^i \in V$ of the graph is computed as follows. Objects in the current observation and target observation are assigned to each other by minimizing the assignment cost using Hungarian method. For instance, in Fig 5.2, object a_t from the current scenario (at time t) is paired with object a_T from the target scenario (at time T) using their feature-based similarity scores $s(a_t, a_T)$. Once object assignment is computed, the center of observed object point cloud $p_t \in \mathbb{R}^3$ and $p_T \in \mathbb{R}^3$ are concatenated to represent $V^i \in \mathbb{R}^6$, the features for graph vertex.

Graph Encoder Network: Our graph encoder network is based on a Higher-order

Algorithm 20: NeRP $\big(X(t), X(T)\big)$

```
$K \leftarrow \text{InstanceSegmentation}(X(t), X(T))$
if at $t = 0$ then
    $h \leftarrow 2 \times |K|$                                   //set planning horizon
else if $h == 0$ then
    report out-of-planning-budget
$s \leftarrow \text{ObjsAlignment}(X(t), X(T), K)$
$G = (V, E) \leftarrow \text{GraphGen}(X(t), X(T), K, s)$
$\overline{X}(t), \overline{G} \leftarrow X(t), G$                           //make a backup copy
$rollouts \leftarrow []$
for 1 to n_rollouts do
    $rollout \leftarrow []$
    $X(t) \leftarrow \overline{X}(t)$
    $G \leftarrow \overline{G}$
    while $h > 0$ do
        $z_{\{N\}} \leftarrow f_{\Theta}(G)$                              //graph encodings
        $\rho_{\{N\}} \leftarrow h_{\Phi}(z_{\{N\}})$
        $i \sim \text{Mult}(\rho_{\{N\}})$                           //graph node sampling
        $\delta_{\{B\}} \leftarrow \pi_{\Omega}(z^i_{\{B\}})$                        //set action generation
        $\hat{X}^i_{\{B\}}(t) = X^i_{\{B\}}(t) + \delta_{\{B\}}$
        $N' = N \backslash i$                                  //unselected node indices
        $\hat{X}^{N'}_{\{B\}}(t) = X^{N'}_{\{B\}}(t)$
        $x_{\{N' \times B\}} \sim \{(\hat{X}^i_{\{B\}}, 0) \cup (\hat{X}^j_{\{B\}}, 1)\}\ \forall j \in N'$
        $ids \leftarrow u_{\xi}(x_{\{N' \times B\}}) > \varepsilon$
        $\delta_{\{B'\}} = \delta_{\{B\}}[ids]$                          //collision-free actions
        $v_{\{B'\}} \leftarrow r_{\Psi}(z^i_{\{B'\}}, \delta_{\{B'\}})$                      //set-action scores
        $j \leftarrow \text{argmax}\ v_{\{B'\}}$                             //best action
        $V^i(t) \leftarrow V^i(t) + \delta^j$                        //update graph vertex
        $X^i(t) \leftarrow X^i(t) + \delta^j$                        //update point-cloud
        $e \leftarrow \text{ComputeError}(V)$
        $rollout \cup (i, \delta_{\{B'\}}, j, e)$
    $rollouts \cup rollout$
$(i, \delta_{\{B'\}}, j) \leftarrow \text{BestRollout}(rollouts)$
$h \leftarrow h - 1$
$X^i_{best} \leftarrow X^i(t) + \delta_j$                            //best placement
$p_{best} \leftarrow \text{ComputeMean}(X^i_{best})$
$X^i_{all} \leftarrow X^i_{\{B'\}}(t) + \delta_{\{B'\}}$                        //all placement points
$X^i_{\mathcal{B}} \leftarrow k\text{-Neighbors}(p_{best}, X^i_{all})$
$c_{map} \leftarrow \text{ComputeCost}\big(p_{best}, X^i_{\mathcal{B}}\big)$
return $X^i_{\mathcal{B}}, c_{map}$
```

Graph Neural Networks [MRF$^+$19], known as k-GNNs, that takes the graph $G = (V, E)$ and hierarchically learns the objects' latent embeddings, $\{z^i;\ \forall i \in [0, N]\}$. We use local k-GNNs, more precisely 1-2-3-GNNs with max-pooling aggregation operator, where each graph layer performs message-passing between individual vertices and local subgraph structures, denoted as $g \subset G$, along the hierarchy from $k = 1$ to $k = 3$, i.e.,

$$f_k^t(g) = \sigma\left(f_k^{t-1}(g) \cdot \Theta_1^t + \max_{g \in \mathcal{N}_L(s)} \left(f_k^{t-1}(g) \cdot \Theta_2^t \right) \right) \tag{5.1}$$

where N_L denotes the local-neighborhood of subgraph g (for more details, refer to [MRF+19]), Θ_1^t and Θ_2^t are the learnable weight parameters, and σ is the component-wise non-linear function ReLU. In our problem setting, k-GNNs with max-pooling operators outperformed vanilla GNNs [WPC+20], as the former captures the fine to coarse level structure of the given current-target aligned scene graph which is crucial to have scene-aware node embeddings. Note that our object alignment network captures across the scene correspondence between objects, whereas our graph encoder network captures the overall current and target placement structure (Fig. 5.2).

Object Selection Network: Our Object Selection Network, $h_\Phi : \mathcal{G} \mapsto \mathcal{M}$, is an object-centric neural model, with parameters Φ, that takes the individual graph node embeddings $z_{\{N\}}$, representing current-target scene graph, and predicts their selection scores $\rho_{\{N\}}$, where $\rho^i \in [0, 1]$, i.e.,

$$\rho_{\{N\}} \leftarrow h(z_{\{N\}}; \Phi) \tag{5.2}$$

We convert all selection scores $\rho_{\{N\}}$ into probabilities to parameterize a multinomial distribution. A graph node index i is sampled from this distribution during planning which denotes a particular object-pairs embedding in the scene graph. Note that the index i also maps to the instance segmentation label, thus resulting into a subset selection operator $M_i \in \mathcal{M}$. The selected object pair's graph node embedding z^i is then used to predict the next relative transformation for the subset $M_i \times X(t) \mapsto X^i(t)$. Furthermore, we train this module using the binary-cross entropy loss as:

$$l_{\Phi,\Theta} = -\frac{1}{N}\sum_i y^i \cdot \log(h_\Phi(z^i) + (1-y^i) \cdot \log(1-h_\Phi(z^i)) \tag{5.3}$$

where y^i is the binary label from demonstrations indicating a graph node to be selected.

δ-Proposal Network: The δ-Proposal Network $\pi_\Omega : \mathcal{X} \mapsto \Delta$, with parameters Ω, is a stochastic neural model that takes the selected node embeddings $z^i_{\{B\}}$ and outputs a variety of candidate translations $\delta^i_{\{B\}} \in \Delta$ that would move the object to a potential new placement region, i.e.,

$$\delta^i_{\{B\}} \leftarrow \pi(z^i_{\{B\}};\Omega) \tag{5.4}$$

where $\delta^i_{\{B\}}$ contains a variety of δ actions of size B and $z^i_{\{B\}}$ contains B replicas of z^i. These $\delta^i_{\{B\}}$ actions are applied to B replicas of selected object's current point-cloud $X^i_{\{B\}}(t)$ which leads to its next placement in the point-cloud space, i.e.,

$$X^i_{\{B\}}(t+1) = \begin{bmatrix} X^i_1(t) \\ \vdots \\ X^i_B(t) \end{bmatrix} + \begin{bmatrix} \delta^i_1 \\ \vdots \\ \delta^i_B \end{bmatrix} \tag{5.5}$$

Furthermore, our generator model obtains its stochasticity from Dropout layers [SHK$^+$14] applied to the networks' linear layers with probability $p \in [0,1]$ during execution. For the given Graph $G = (V,E)$ and the true next step delta $\bar{\delta}$ from demonstrations, this module is trained by minimizing the following:

$$l_{\Omega,\Theta}(M, f_\Theta(G), \bar{\delta}) = \left\| \rho_\Omega\big(M \circ f_\Theta(G)\big) - \bar{\delta} \right\|_2 \tag{5.6}$$

where M is a mask operator which selects the graph node embedding that corresponds to the given true delta $\bar{\delta}$ label only.

Goal Satisfaction Network: Our goal satisfaction network is a value function that scores the given actions for their ability to accomplish the given target arrangements. It is a neural function $r_\Psi : \mathcal{G} \times \Delta \mapsto [0,1]$, with parameters Ψ, that takes a selected graph node embedding and the given action δ as an input and outputs goal satisfaction scores, i.e.,

$$\hat{y}_r \leftarrow \sigma\big(r_\Psi(M \circ f_\Theta(G), \delta)\big) \tag{5.7}$$

where $\hat{y}_r \in [0,1]$, M is a mask operator that selects a graph node embedding corresponding to the given action δ, and σ is a sigmoid function to squash predicted scores to $[0,1]$. The $\hat{y}_r = 1$ indicates that the given action δ can take the selected object to its target location, and $\hat{y}_r = 0$ indicates otherwise. Furthermore, this function is optimized through binary-cross entropy loss as follows:

$$l_{\Psi,\Theta}(M, f_\Theta(G), y_r) = -y_r \cdot \log(\hat{y}_r) - (1 - y_r) \cdot \log(1 - \hat{y}) \tag{5.8}$$

where y_r denotes the true label.

Collision Network: Our collision detection network $u_\xi : \mathcal{X} \mapsto [0,1]$, with parameters ξ, uses a number of PointNet++ set-abstraction layers [QYSG17] to detect the intersection between any two given sets $X_0, X_1 \subset X$. In our setting, X_0 and X_1 can be any arbitrarily selected object point-clouds with their feature labeled as $Y_0 = 0$ and $Y_1 = 1$, respectively. Our collision network's input is a subset sampled from a joint-set of given point-clouds and their feature masks, i.e., $x \subset (X_0, Y_0) \cup (X_1, Y_1)$. The feature mask indicates element-wise point-cloud correspondence to the given sets. The network predicts scores $\rho_c \in [0,1]$ indicating degeree of sets' intersections. We train our collision-checker independently from other NeRP models using the BCE loss with training samples containing both intersecting and disjoint point-cloud sets.

NeRP Training

We train our core models, i.e., graph encoder $f_\Theta(\cdot)$, node selector $h_\Phi(\cdot)$, δ-proposal network $\pi_\Omega(\cdot)$, and goal satisfaction evaluator $r_\Psi(\cdot)$, jointly in an end-to-end manner by optimizing

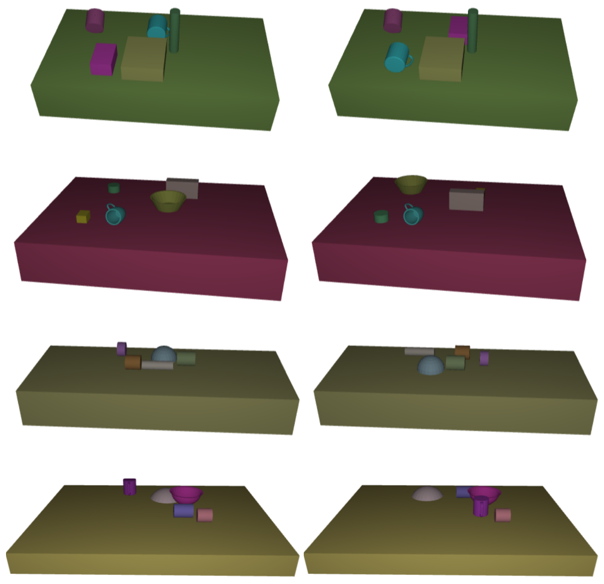


Figure 5.3: Examples of generated data. Objects are randomly placed on the table, and we chose different random motions as well.

the following objective:

$$\frac{1}{N_{\mathcal{M}}} \sum_{\tau \sim \mathcal{M}} l_{\Phi,\Theta}(z_{\{N\}}, y_{\{N\}}) + l_{\Omega,\Theta}(z^i, \overline{\delta}^i) + l_{\Psi,\Theta}(z^i, \overline{\delta}^i, y_r), \tag{5.9}$$

where $\tau = (G, y_{\{N\}}, \overline{\delta}^i, M, y_r)$ denotes a training sample from demonstration data $\mathcal{M}$, comprising a current-target paired scene graph G with its node embeddings $z_{\{N\}} = f_\Theta(G)$. It also includes the node selection labels as $y_{\{N\}}$, and a differentiable mask M operator to select a desired node embedding $z^i = M \circ z_{\{N\}}$. Furthermore, we also provide a desired next action $\overline{\delta}^i$ and its goal satisfaction value y_r for the selected node. The graph encoder is learned end-to-end with other core models to capture scene embedding useful for the overall rearrangement planning process whereas the collision network is trained independently.

NeRP Planning Algorithm

Algorithm 20 describe our multi-step planning algorithm for efficiently solving tabletop rearrangement tasks. For the given observations, $X(t)$ and $X(T)$, our method begins by computing their instance segmentation K using UCN [XXMF20a]. A current-target aligned scene graph G is then instantiated using those segmented observations and the objects similarity scores s (Lines 6-7).

Given an observation graph $G = (V, E)$, our multi-step predictive planning approach imagines n action sequences out to horizon h, stores them into a buffer $rollouts$, and executes the first action through a low-level controller of the best-selected planning sequence. In our approach, we use a Model Predictive Path Integral (MPPI) controller based on learned collision avoidance models [DMEF21] to perform the given placement actions leading to the next observation graph for the planning. After each pick-and-place action is executed, the horizon h is decremented and we replan. If controller fails to execute an action (e.g. it fails to grasp or drops the object early) the plan horizon is incremented again to give another opportunity to the planner for re-planning.

In a planning sequence, the graph encoder, f_Θ, outputs the graph nodes' embeddings $z_{\{N\}}$. The function h_Φ takes graph embeddings and predicts $\rho_{\{N\}}$ which parametrize a Multinomial distribution for sampling an index $i \in [0, N]$. For a selected graph node's embedding z^i, multiple replicas are fed to our stochastic δ-proposal network, π_Ω, to determine the various next step actions $\delta_{\{B\}}$ for the placements of the object i (Line 15-18).

Each of the candidate placements of the selected object in $\hat{X}^i_{\{B\}}(t)$ forms a new scenario. To determine the best next-step scene arrangement, we translate the object point cloud according to the sampled δ and use our collision and goal-satisfaction networks. The collision-network, u_ξ, takes all possible next step objects point-clouds and returns action indices, ids, leading to all collision-free next-step arrangements (Line 22-24). The goal satisfaction network takes these collision-free placement actions, $\delta_{\{B'\}}$, and provides the scores, $v_{\{B'\}}$, to select the best move for simulating the next-step using the graph G and current point-cloud $X(t+1)$ (Line 25-28).

Furthermore, for each planning step, we compute a L2 norms based error e between the updated graph vertices $V(t)$ and $V(T)$, indicating difference between current simulated scene and the final arrangement. All this planning step information, including error e, is stored in the rollout buffer. Once all sequences are unrolled, the first step of the best-unrolled sequence, based on minimum error e, is selected to determine best object i and the placement points, X^i_{all}, and a placement cost map c_{map} for the object i. To select X^i_{all} and compute their cost-map c_{map}, we calculate the centroid, p_{best}, of the next best placement point-cloud X^i_{best}, and select all points in X^i_{all} that are within a ball of radius of p_{best}.

The cost values are calculated based on the distance between p_{best} and all placement locations $X^i_{\mathcal{B}}$ (Line 38). These placement locations with cost map are given to the MPPI low-level controller for robot execution. The controller generates grasps for object i using [SMTF21] and executes the motions using [DMEF21]. Once the object is lifted, it chooses the placement with minimum c_{map} for which there is a collision-free, kinematically feasible path.

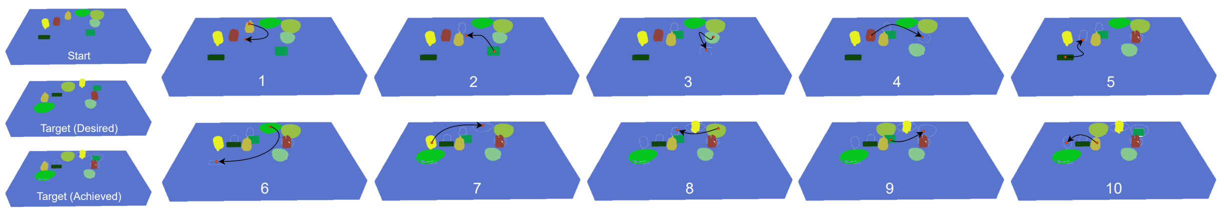


Figure 5.4: An example plan rollout showing how NeRP chose to move objects around in order to get between two goal states with very different arrangements of obstacles. In this case, it took 10 steps to get to the goal state.

Data Generation

To train the NeRP models, we generate random synthetic scene data rendered with variable camera poses. All scenes contained randomly selected five objects, initially scattered all over the tabletop of changing dimensions (as shown in Fig. 5.3). Each scenario's target arrangement was determined by randomly swapping the objects' placements in the initial scene, so that each object's goal location is blocked by a random other object. Hence, transitioning from an initial to target scene requires some items to be moved to another empty location, which we call *storage*, to vacate the occupied position. We generate the intermediate placement action using a model-based expert rearrangement planner (as described in Section 5.4). The expert planner generates intermediate scene sequences from which we determine the step-wise relative scene transformations $(M, \overline{\delta}) \in \mathcal{M} \times \Delta$ for training our model-free, set-based rearrangement planner. Following this procedure, we gathered a dataset comprising seven thousand re-arrangement task problems with their intermediate planning sequences.

5.4 Results

We performed four sets of experiments. First, we tested our method on unseen synthetic data against various classical baselines. Second, we show generalization of our method on object rearrangement for unseen number of objects. Third, we do ablation study to evaluate effect of each component of NeRP and finally we demonstrate our method's sim-to-real generalization performance on real-world object rearrangement tasks with different sets of unseen objects and for unseen rearrangement tasks. We use the following metrics for quantitative comparison of different methods:

- **Success Rate** indicates the percentage of successfully solved unseen arrangement problems; an object arrangement is considered successful if the maximum displacement for each object does not exceed 5*mm*.
- **Planning Steps** measures the number of steps required to rearrange the objects from source configuration to target configuration.
- **Final Error** measures the average L2 distance between the desired target arrangement and the actual arrangement achieved by a given planner.

Note that success rate is computed over all the rearrangement scenarios whereas planning steps and final error are computed only on successful rearrangements.

Table 5.1: Comparison between NeRP and several classical baselines. NeRP produces shorter, more accurate plans than baseline methods.

Algorithms	Performance Metrics		
	Success rate (%) $\uparrow$	Planning steps $\downarrow$	Final error $\downarrow$
Expert	90.67 ± 0.60	8.41 ± 2.61	$\mathbf{0.0 \pm 0.0}$
NeRP (Ours)	$\mathbf{94.56 \pm 0.73}$	$\mathbf{7.01 \pm 2.10}$	0.019 ± 0.013
Classical (greedy, parallelized)	68.20 ± 0.79	13.60 ± 5.99	0.023 ± 0.017
Classical (random, parallelized)	59.23 ± 1.32	42.80 ± 60.63	0.019 ± 0.011

Algorithm Comparison

To validate our approach, we first tested on 500 simulated unseen object rearrangement problems that were generated with 5 objects following the procedure in Section 5.3 (Data Generation). Table 5.1 shows how our method compares to multiple baseline methods. These baselines include:

Expert: The expert planner is a model-based planning approach that uses objects' unique ids, transformations, table mesh, object meshes, and FCL [PCM12] collision-checker for planning. For this model, we set the same step length limit as NeRP, i.e., $2 \times |K|$. It randomly selects an object to move that is not at its goal position, and then checks to see if its target location is empty or occupied. If the target location is occupied, it will move the occupying object to free space, chosen by randomly sampling a feasible *storage* location on the table. If the target location is free, it will instead move the object to the goal. This process is repeated until all the objects are at their goal positions.

Classical (greedy, parallelized): This is a model-free version of the "Expert" planner. It uses instance segmentation and our object alignment model to match objects between the current and target scenes. Once objects pair are computed, it randomly selects an object, checks if its target location is occupied or empty, and moves the occupying object to storage by sampling multiple placements in parallel and rejecting colliding placement positions using our collision-net u_ξ instead of using FCL. Once the target location is free, it moves the selected object to its target, and proceeds by randomly selecting another object and repeating the process.

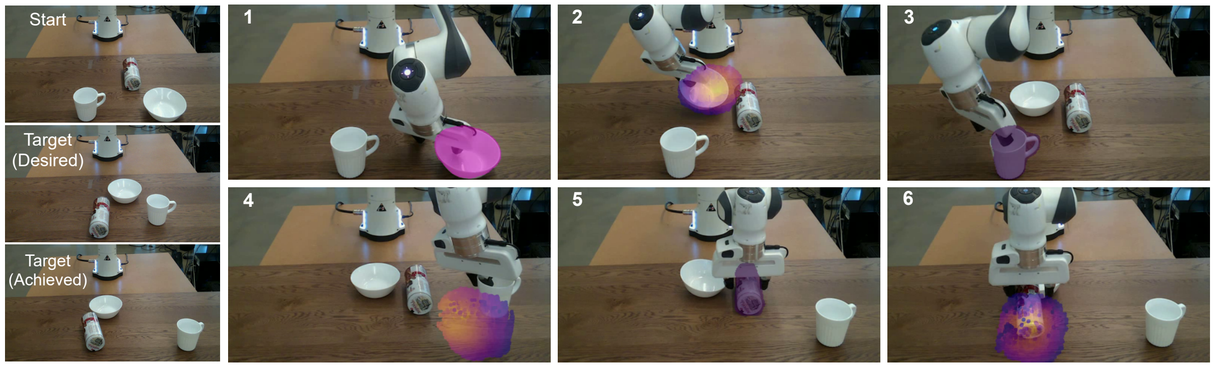


Figure 5.5: Example of a planning sequence. The robot repeatedly selects which object to move and either moves it to the appropriate goal position or to a *storage* position to enable future execution.

Classical (random, parallelized): The random baseline plans along multiple sequences, and executes the first action of the best selected sequence, following the algorithm given by Alg. 20. The best sequence is selected based on minimum error, e, from the target arrangement. However, unlike, NeRP, this baseline randomly chooses an object and performs a sequence execution like a classical (greedy, parallelized) approach mentioned above.

Comparison to classical methods: Table 5.1 shows how NeRP out-performs all baseline methods, including the model-based expert that uses all environment state information. This is for several reasons. First, the random placements can often be in-collision, especially in cases with a small tabletop area. Second, model-free baselines diverge in cases with noisy feature-based objects' pairing, which is inevitable with learning-based feature matching approaches. Third, in the multi-step, multi-sequence classical planning cases, executing the first action and replanning the sequences again instead of greedily utilizing each action step takes a longer time to converge.

Generalization to different numbers of objects: Table 5.2 shows NeRP performance over 500 scenarios with a number of objects ranging from two to eight, whereas NeRP training

Table 5.2: Generalization of NeRP to different number of objects. Note that NeRP is trained on random rearrangements of 5 objects.

Number of Objects	Performance Metrics		
	Success rate $(\%) \uparrow$	Planning steps $\downarrow$	Final error $\downarrow$
3 Objects	98.25 ± 0.57	4.58 ± 0.82	0.039 ± 0.036
4 Objects	97.60 ± 1.20	5.70 ± 1.38	0.027 ± 0.025
6 Objects	98.09 ± 0.40	8.69 ± 2.15	0.013 ± 0.013
7 Objects	90.62 ± 1.03	9.47 ± 2.23	0.011 ± 0.008
8 Objects	87.50 ± 2.50	10.72 ± 2.08	0.010 ± 0.008

Table 5.3: Analysis of the effects of ablation of various components of the network. Removing stochasticity, the object selection network or the goal selection network has a significant negative effect on performance.

Algorithms	Performance Metrics		
	Success rate $(\%) \uparrow$	Planning steps $\downarrow$	Final error $\downarrow$
NeRP	94.56 ± 0.73	7.01 ± 2.10	0.019 ± 0.013
w/o Dropout	36.87 ± 0.24	8.23 ± 3.07	0.019 ± 0.012
w/o OS h_Φ	30.12 ± 1.10	11.82 ± 3.30	0.025 ± 0.021
w/o GS r_Ψ	48.21 ± 0.68	14.84 ± 1.14	0.016 ± 0.010

dataset contained scenes with only five objects. Fig. 5.4 shows the NeRP execution in a scene arrangement task with eight objects. From these results, we can see that NeRP's object-centric planning is robust to scene clutter and can efficiently sample multiple *storage* locations along the planning sequence for achieving a complex target arrangement setup.

Ablation Studies

We then performed a set of experiments ablating various components of our method. In particular, we look at the importance of the stochastic Dropout added to our δ-proposal network π_Ω, the object selection network h_Φ, and the goal satisfaction network r_Ψ.

The results are shown in Table 5.3. We can see that all of these components are quite important. Without Dropout [SHK$^+$14], the architecture cannot find *storage* positions to place an object if the goal is blocked, making it challenging to swap two objects – a deterministic network

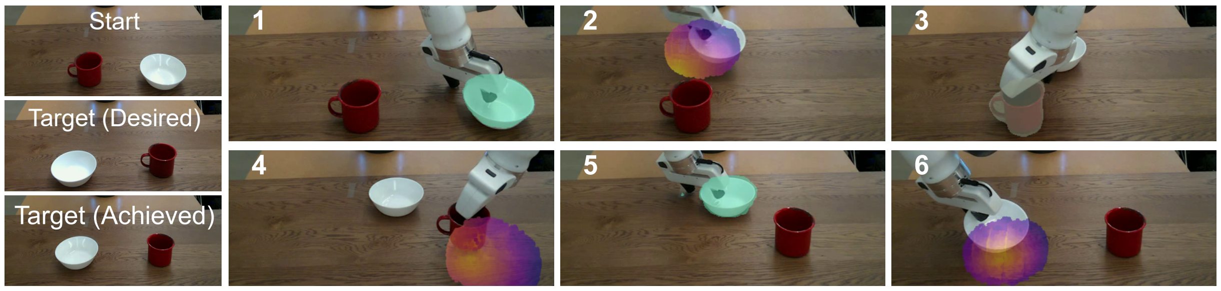


Figure 5.6: Swapping an unseen mug and bowl using NeRP: For the given $X(\text{Start})$ and $X(\text{End})$ arrangements, NeRP selects an object in the given scenarios (e.g., 1, 3 & 5) and predicts its next placement with a cost map c_{map} (e.g., 2, 4 & 6).

could potentially learn this behavior, but it would be a much more difficult and less generalizable solution, and perhaps harder to learn from random imitation data. Without r_Ψ, we see much longer plans as the model makes numerous small adjustments without terminating, as evidenced by the lower final error. Without object selection, we select random objects that are not at their final positions, also leading to more inefficiency.

Real Robot Experiments

Finally, we performed a set of real-world experiments using a Franka Panda robot arm with an externally mounted Intel Realsense L515 camera. We set up the two types of testing scenarios i) swapping objects to achieve a given target arrangement and ii) sorting objects into different categories similar to the given target observation. Fig. 5.1 shows the robot sorting bowls and mugs, Fig. 5.5 shows the swapping of two objects and Fig. 5.6 demonstrates the swapping tasks based on NeRP's predicted actions. However, the major limiting factors in real-robot tasks were noisy scene segmentation and feature extractions, which often led to rearrangement failure due to incorrect object correspondences, as also highlighted in the supplementary video.

We see in all these different planning setups that despite being trained on synthetic data with only five object categories, NeRP generalizes to novel problem settings with high performance. In sim-to-real transfer, it generates actions for the low-level controller to move the object, while in other scenarios, the objects were directly teleported to their best placement.

5.5 Acknowledgements

Chapter 5, in part, is a reprint of **A.H.Qureshi**, A.Mousavian, C.Paxton, M.C.Yip, and D.Fox, "NeRP: Neural Rearrangement Planning for Unknown Objects", *Robotics: Science and Systems 2021.* The dissertation author is the primary author of this paper.

Chapter 6

Conclusions & Future Works

This dissertation presents deep learning-based methods inspired by the human development process that learn constraint or cost functions and their motion policies from expert demonstrations and compose them into new complex skills to solve new problems across different domains. It also highlights that proposed methods form a mutualistic relationship with existing learning-based task planning approaches. In addition, it presents a novel rearrangement task planning approach that operates in unknown environments from raw sensory information.

In constraint learning, also known as cost or reward learning, this dissertation presents an approach to adversarial reward and policy learning from expert demonstrations by regularizing the maximum-entropy inverse reinforcement learning through empowerment. The proposed method learns the empowerment through variational information maximization in parallel to learning the reward and policy. The policy is trained to imitate the expert behavior as well to maximize the empowerment of the agent over the environment. The proposed regularization prevents premature convergence to local behavior and leads to a generalized policy that in turn guides the reward-learning process to recover near-optimal reward. The results show that the proposed method successfully learns near-optimal rewards, policies, and performs significantly better than state-of-the-art IRL methods in both imitation learning and challenging transfer learning problems. The learned rewards are shown to be transferable to environments that are dynamically or structurally different from training environments. A possible future extension to this approach can be to learn rewards and policies from diverse human/expert demonstrations as the proposed method assumes that a single expert generates the training data. Another exciting direction would be to build an algorithm that learns from suboptimal demonstrations that contains both optimal and non-optimal behaviors.

In learning robot motion skills for given task objectives, this thesis presents a learning-based approach to motion planning using deep neural networks called Neural Motion Planning (NMP). For a given planning problem, NMP is capable of i) finding collision-free near-optimal paths under various kinodynamic, and manifold kinematic constraints; ii) they can also generate samples for underlying sampling-based motion planners in a subset of a given configuration space that potentially contains solutions including the optimal path. The dissertation also presents an active continual learning strategy to train NMP models with a significant improvement in training data-efficiency compared to naive training approaches. The experimentation shows that the neural motion planners consistently find collision-free paths under given constraints much faster than other planners. The modular structure of NMPs naturally allow their coupling with learning-based task planners, forming a mutual symbiotic relationship to efficiently solve task and motion planning problems. In our future works, one of the primary objectives can be to tackle the environment encoding problem for motion planning. Environment encoding is one of the critical challenges in real-world robotics problems. Current perception approaches consider encoding of individual objects rather than the entire scene that retains the inter-object relational geometry which is crucial for motion planning. Henceforth, a possible research avenue is learning motion planning oriented environments' encoding from raw point-cloud data. Another future objective would be to fully harness NMPs potential, i.e., their fast computational speed, to address motion planning problems in robot surgery under contact dynamics and safety constraints.

To reuse existing motion skills and transfer them to new domains, this dissertation presents a novel policy ensemble composition method that combines a set of independent and task-agnostic primitive policies through reinforcement learning to solve the given new tasks. This method can transfer the given skills to novel problems and can compose them both sequentially (or -operation) and concurrently (and -operation) to find a solution for the task in hand. The experiments highlight that composition is vital for solving problems requiring complex motion skills and decision-making where standard reinforcement learning and hierarchical reinforcement learning methods either fail or need a massive number of interactive experiences to achieve the desired results. An exciting future work could be to extend this composition method to automatically acquire the missing skills in the given skillset that are necessary to solve the specific problems. It is also imperative to investigate deep hierarchies of these composition models by combining primitive policies into complex policies that are further composed together for a combinatorial outburst in the agent's skillset.

Apart from learning robot objective functions, motion skills, and composite models, another important area of research is task planning that decomposes a given task into a sequence of achievable subtasks for underlying motion planners. Recently, rearrangement task planning has been highlighted as an important research direction as real-world robots would have to re-arrange things in unstructured environments. This dissertation presents a novel Neural Rearrangement Planning (NeRP), a deep neural network-based rearrangement approach for unknown objects. NeRP can rearrange unseen objects without models, and works in an end-to-end fashion, given segmented point-clouds from an RGB-D camera. NeRP is evaluated on challenging problems and results demonstrate its sim-to-real generalizations. NeRP relies on scene segmentation and correspondence matching techniques to generate a scene graph between the current and target observations. The scene graph is used to generate a sequence of intermediate object selection and their placement actions for reaching the given target arrangement. However, in the sim-to-real transfer experiments, it is observed that existing scene segmentation and feature-based object correspondence techniques often fail in the real-setup. This results in an incorrect scene graph and, therefore, NeRP behavior. Hence, an important future direction would be to augment NeRP with robust segmentation and feature matching algorithms to enhance its real-world settings' performance. Another future objective would be to extend NeRP for model-free planning in SE-3 space to compute both relative translation and rotations for transforming point sets into complex shapes and arrangements.

Acknowledgements

This chapter, in part, is a reprint of the following publications:

- **A.H.Qureshi**, B. Boots, and M.C.Yip,"Adversarial Imitation Via Variational Inverse Reinforcement Learning", *International Conference on Representation Learning (ICLR)*, 2019. The dissertation author is the primary author of this paper.
- **A.H.Qureshi**, Y.Miao, A.Simeonov, and M.C.Yip,"Motion Planning Networks: Bridging the Gap Between Learning-based and Classical Motion Planners", *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48-66, Feb. 2021, doi: 10.1109/TRO.2020.3006716. The dissertation author is the main of this paper.
- **A.H.Qureshi**, A.Simeonov, M.J.Bency, and M.C.Yip, "Motion Planning Networks", *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2118-2124, Montreal, Canada, 2019. The dissertation author is the primary author of this paper.
- **A.H.Qureshi** and Michael.C.Yip, "Deeply Informed Neural Sampling For Robot Motion Planning", *IEEE International Conference on Intelligent Robot and Systems (IROS)*, pp. 6582-6588, 2018. The dissertation author is the primary author of this paper.
- L. Li, Y. Miao, **A. H. Qureshi** and M. C. Yip,"MPC-MPNet: Model-Predictive Motion Planning Networks for Fast, Near-Optimal Planning Under Kinodynamic Constraints",

IEEE Robotics and Automation Letters, vol. 6, no. 3, pp. 4496-4503, July 2021, doi: 10.1109/LRA.2021.3067847. The dissertation author is the co-author of this paper.

- **A.H.Qureshi**, J.Dong, A.Baig, and M.C.Yip,“Constrained Motion Planning Networks X”, *IEEE Transactions on Robotics*, 2021. The dissertation author is the primary author of this paper.

- **A.H.Qureshi**, J.Dong, A.Choe, and M.C.Yip, “Neural Manipulation Planning on the Constraint Manifolds”, *IEEE Robotics and Automation Letters*, 2020. The dissertation author is the primary author of this paper.

- **A.H.Qureshi**, J. J. Johnson, Y. Qin, T. West, B. Boots, and M.C.Yip. “Composing Task-Agnostic Policies via Deep Reinforcement Learning”, *International Conference on Representation Learning (ICLR)*, 2020. The dissertation author is the primary author of this paper.

  A.H.Qureshi, A.Mousavian, C.Paxton, M.C.Yip, and D.Fox, “NeRP: Neural Rearrangement Planning for Unknown Objects”, *Robotics: Science and Systems 2021*. The dissertation author is the primary author of this paper.

Bibliography

[AAST12] Hiroaki Arie, Takafumi Arakaki, Shigeki Sugano, and Jun Tani. Imitating others by composition of primitive actions: A neuro-dynamic model. *Robotics and Autonomous Systems*, 60(5):729–741, 2012.

[ABB$^+$15] Christopher G Atkeson, Benzun P Wisely Babu, Nandan Banerjee, Dmitry Berenson, Christoper P Bove, Xiongyi Cui, Mathew DeDonato, Ruixiang Du, Siyuan Feng, Perry Franklin, et al. No falls, no resets: Reliable humanoid behavior in the darpa robotics challenge. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 623–630. IEEE, 2015.

[ACVB09] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[Aga04] David Barber Felix Agakov. The im algorithm: a variational approach to information maximization. *Advances in Neural Information Processing Systems*, 16:201, 2004.

[AN04] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[AOJJ13] Muhannad AR Ai-Omari, Mohammad A Jaradat, and Mohammad Jarrah. Integrated simulation platform for indoor quadrotor applications. In *2013 9th International Symposium on Mechatronics and its Applications (ISMA)*, pages 1–6. IEEE, 2013.

[AP19] Ross E Allen and Marco Pavone. A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance. *Robotics and Autonomous Systems*, 115:174–193, 2019.

[AQF21] Chris Paxton Michael Yip Ahmed Qureshi, Arsalan Mousavian and Dieter Fox. Nerp: Neural rearrangement planning for unknown objects. In *Proceedings of Robotics: Science and Systems*, 2021.

[AYYS16] Aliasghar Arab, Kaiyan Yu, Jingang Yi, and Dezhen Song. Motion planning for aggressive autonomous vehicle maneuvers. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 221–226. IEEE, 2016.

[BAG12] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3671–3678. IEEE, 2012.

[BCB$^+$19] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, Thomas Lew, and Marco Pavone. Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[BCC$^+$20] Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, et al. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020.

[BCS17] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. Learning heuristic search via imitation. *arXiv preprint arXiv:1707.03034*, 2017.

[BFM$^+$20] Daniel M Bear, Chaofei Fan, Damian Mrowca, Yunzhu Li, Seth Alter, Aran Nayebi, Jeremy Schwartz, Li Fei-Fei, Jiajun Wu, Joshua B Tenenbaum, et al. Learning physical graph representations from visual scenes. *arXiv preprint arXiv:2006.12373*, 2020.

[BHP17] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[BKP11] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[BKRE] Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Faculty Of Industrial Engineering. The cross-entropy method for optimization.

[BLP85] Rodney A Brooks and Tomas Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, (2):224–233, 1985.

[BQY19] Mayur J Bency, Ahmed H Qureshi, and Michael C Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. *arXiv preprint arXiv:1904.11102*, 2019.

[BRP18] Ricard Bordalba, Lluís Ros, and Josep M Porta. Randomized kinodynamic planning for constrained systems. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7079–7086. IEEE, 2018.

[BSK11] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460, 2011.

[Can88] John Canny. *The complexity of robot motion planning*. MIT press, 1988.

[CHES11] Akansel Cosgun, Tucker Hermans, Victor Emeli, and Mike Stilman. Push planning for object placement on cluttered table surfaces. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 4627–4632. IEEE, 2011.

[CHF$^+$19] Hao-Tien Lewis Chiang, Jasmine Hsu, Marek Fiser, Lydia Tapia, and Aleksandra Faust. Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies. *IEEE Robotics and Automation Letters*, 4(4):4298–4305, 2019.

[CHL$^+$05] Howie M Choset, Seth Hutchinson, Kevin M Lynch, George Kantor, Wolfram Burgard, Lydia E Kavraki, and Sebastian Thrun. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[CKY$^+$17] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3381–3388. IEEE, 2017.

[CS00] Richard Cooper and Tim Shallice. Contention scheduling and the control of routine activities. *Cognitive neuropsychology*, 17(4):297–338, 2000.

[CŞM$^+$15] David Coleman, Ioan A Şucan, Mark Moll, Kei Okada, and Nikolaus Correll. Experience-based planning with sparse roadmap spanners. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 900–905. IEEE, 2015.

[CWD$^+$18] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[DCH$^+$16] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[DMEF21] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. Object rearrangement using learned implicit collision functions. *International Conference on Robotics and Automation*, 2021.

[DNP$^+$13] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[DXCR93] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.

[FCAL16] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016.

[FDA17] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[FLA16] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[FLL17] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[FvHM18] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[GCH$^+$20] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *arXiv preprint arXiv:2010.01083*, 2020.

[GG16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[GHLL17] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[GPAM$^+$14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[GPLP$^+$20] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5678–5684. IEEE, 2020.

[GPPK13] Gustavo Goretkin, Alejandro Perez, Robert Platt, and George Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *2013 IEEE International Conference on Robotics and Automation*, pages 2429–2436. IEEE, 2013.

[GSB14] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2997–3004. IEEE, 2014.

[GSB15] Jonathan D Gammell, Siddhartha Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3067–3074. IEEE, 2015.

[Han00] Li Han. A kinematics-based probabilistic roadmap method for closed chain systems. In *In Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR*, 2000.

[Hau15] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2951–2957. IEEE, 2015.

[HBKP18] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[HE16] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[Hen02] Michael E Henderson. Multiple parameter continuation: Computing implicitly defined k-manifolds. *International Journal of Bifurcation and Chaos*, 12(03):451–476, 2002.

[HLF$^+$19] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[HMP$^+$17] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, volume 3, 2017.

[HMP$^+$18] Nika Haghtalab, Simon Mackenzie, Ariel D Procaccia, Oren Salzman, and Siddhartha S Srinivasa. The provable virtue of laziness in motion planning. In

Twenty-Eighth International Conference on Automated Planning and Scheduling, 2018.

[HNTH10] Kris Hauser and Victor Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *2010 IEEE international conference on robotics and automation*, pages 2493–2498. IEEE, 2010.

[HNX$^+$19] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8565–8574, 2019.

[HPZ$^+$18] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1803.06773*, 2018.

[HS] Brian Hou and Siddhartha Srinivasa. Deep conditional generative models for heuristic search on graphs with expensive-to-evaluate edges.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HZAL18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition*, 2015.

[IC18] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[IHP18] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[INH$^+$13] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[IP19] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 2019.

[ISL20] Brian Ichter, Pierre Sermanet, and Corey Lynch. Broadly-exploring, local-policy trees for long-horizon task planning. *arXiv preprint arXiv:2010.06491*, 2020.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[JLL$^+$20] Jacob J Johnson, Linjun Li, Fei Liu, Ahmed H Qureshi, and Michael C Yip. Dynamically constrained motion planning networks for non-holonomic robots. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[JP13a] Léonard Jaillet and Josep M Porta. Asymptotically-optimal path planning on manifolds. *Robotics: Science and Systems VIII*, pages 145–152, 2013.

[JP13b] Leonard Jaillet and Josep M Porta. Efficient asymptotically-optimal path planning on manifolds. *Robotics and Autonomous Systems*, 61(8):797–807, 2013.

[JP16] Lucas Janson and Marco Pavone. Fast marching trees: A fast marching sampling-based method for optimal motion planning in many dimensions. In *Robotics Research*, pages 667–684. Springer, 2016.

[JP17] Léonard Jaillet and Josep M Porta. Path planning with loop closure constraints using an atlas-based rrt. In *Robotics Research*, pages 345–362. Springer, 2017.

[JSB$^+$15] Matthew Johnson, Brandon Shrewsbury, Sylvain Bertrand, Tingfan Wu, Daniel Duran, Marshall Floyd, Peter Abeles, Douglas Stephen, Nathan Mertins, Alex Lesman, et al. Team ihmc's lessons learned from the darpa robotics challenge trials. *Journal of Field Robotics*, 32(2):192–208, 2015.

[JSCP15] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[Kac02] Bolesław Kacewicz. Complexity of nonlinear two-point boundary-value problems. *Journal of Complexity*, 18(3):702–738, 2002.

[KBP13] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[KC17] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5974–5983, 2017.

[KF11] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[KGB15] Markus Kuderer, Shilpa Gulati, and Wolfram Burgard. Learning driving styles for autonomous vehicles from demonstration. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2641–2646. IEEE, 2015.

[Kha86] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[KHL17] Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. *arXiv preprint arXiv:1703.06692*, 2017.

[KKGB12] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012.

[KKKL94] Yoshihito Koga, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning motions with intentions. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 395–408, 1994.

[KL98] Lydia E Kavraki and Jean-Claude Latombe. Probabilistic roadmaps for robot path planning. 1998.

[KL00] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *2000 IEEE international conference on robotics and automation*, volume 2, pages 995–1001. IEEE, 2000.

[KLP17] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Learning composable models of parameterized skills. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 886–893. IEEE, 2017.

[KLR$^+$20] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. *arXiv preprint arXiv:2006.05768*, 2020.

[KMK18] Zachary Kingston, Mark Moll, and Lydia E Kavraki. Sampling-based methods for motion planning with constraints. *Annual review of control, robotics, and autonomous systems*, 1:159–185, 2018.

[KMK19] Zachary Kingston, Mark Moll, and Lydia E Kavraki. Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research*, 38(10-11):1151–1178, 2019.

[KPRS13] Mrinal Kalakrishnan, Peter Pastor, Ludovic Righetti, and Stefan Schaal. Learning objective functions for manipulation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1331–1336. IEEE, 2013.

[KRS17] Jennifer E King, Vinitha Ranganeni, and Siddhartha S Srinivasa. Unobservable monte carlo planning for nonprehensile rearrangement tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4681–4688. IEEE, 2017.

[KUSP16] Beobkyoon Kim, Terry Taewoong Um, Chansu Suh, and Frank C Park. Tangent bundle rrt: A randomized algorithm for constrained motion planning. *Robotica*, 34(1):202–225, 2016.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[KWKLP19] Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. *The International Journal of Robotics Research*, 38(7):793–812, 2019.

[Lat12] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[LaV98] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[LaV06] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[LFDA16] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[LHP$^+$15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[LKJ01] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[LLB16] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[LMQY21] Linjun Li, Yinglong Miao, Ahmed H Qureshi, and Michael C Yip. Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints. *IEEE Robotics and Automation Letters*, 6(3):4496–4503, 2021.

[LPK$^+$18] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[LPR17] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[LPW79] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[LUTG17] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

[LZK$^+$20] Yann Labbé, Sergey Zagoruyko, Igor Kalevatykh, Ivan Laptev, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Monte-carlo tree search for efficient visually guided rearrangement planning. *IEEE Robotics and Automation Letters*, 5(2):3715–3722, 2020.

[MEF19] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2901–2910, 2019.

[MHM11] Takamitsu Matsubara, Sang-Ho Hyon, and Jun Morimoto. Learning parametric dynamic movement primitives from multiple demonstrations. *Neural networks*, 24(5):493–500, 2011.

[MIB00] Ferdinando A Mussa-Ivaldi and Emilio Bizzi. Motor learning through the combination of primitives. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 355(1404):1755–1769, 2000.

[MKP10] Katharina Muelling, Jens Kober, and Jan Peters. Learning table tennis with a mixture of motor primitives. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 411–416. IEEE, 2010.

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[MME$^+$20] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Chris Paxton, and Dieter Fox. 6-dof grasping for target-driven object manipulation in clutter. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6232–6238. IEEE, 2020.

[MML$^+$17] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. Dex-net 3.0: computing robust robot vacuum suction grasp targets in point clouds using a new analytic model and deep learning. *arXiv preprint arXiv:1709.06670*, 2017.

[MR15] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 2125–2133, 2015.

[MRF$^+$19] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[NGLL18] Ofir Nachum, Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018.

[NHR99] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[NLC$^+$19] Changjoo Nam, Jinhwi Lee, Younggil Cho, Jeongho Lee, Dong Hwan Kim, and ChangHwan Kim. Planning for target retrieval using a robotic manipulator in cluttered and occluded environments. *arXiv preprint arXiv:1907.03956*, 2019.

[NR$^+$00] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[OJO$^+$20] Kei Ota, Devesh K Jha, Tadashi Onishi, Asako Kanezaki, Yusuke Yoshiyasu, Yoko Sasaki, Toshisada Mariyama, and Daniel Nikovski. Deep reactive planning in dynamic environments. In *4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.*, 2020.

[OSJ$^+$20] Kei Ota, Yoko Sasaki, Devesh K Jha, Yusuke Yoshiyasu, and Asako Kanezaki. Efficient exploration in constrained environments with goal-oriented reference path. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6061–6068. IEEE, 2020.

[PBK$^+$19] Chris Paxton, Yotam Barnoy, Kapil Katyal, Raman Arora, and Gregory D Hager. Visual robot task planning. In *2019 international conference on robotics and automation (ICRA)*, pages 8832–8838. IEEE, 2019.

[PCM12] 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*, pages 3859–3866, 2012.

[PCZ$^+$19] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. Mcp: Learning composable hierarchical control with multiplicative compositional policies. *arXiv preprint arXiv:1905.09808*, 2019.

[PDPN13] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems*, pages 2616–2624, 2013.

[PHL+17] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[PKP14] Georgios Papadopoulos, Hanna Kurniawati, and Nicholas M Patrikalakis. Analysis of asymptotically optimal sampling-based motion planning algorithms for lipschitz continuous dynamical systems. *arXiv preprint arXiv:1405.2872*, 2014.

[Pom91] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[PPK+12] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *2012 IEEE International Conference on Robotics and Automation*, pages 2537–2542. IEEE, 2012.

[Pre00] Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[QA15] Ahmed Hussain Qureshi and Yasar Ayaz. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 68:1–11, 2015.

[QA16] Ahmed Hussain Qureshi and Yasar Ayaz. Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, 40(6):1079–1093, 2016.

[QBY19] Ahmed H. Qureshi, Byron Boots, and Michael C. Yip. Adversarial imitation via variational inverse reinforcement learning. In *International Conference on Learning Representations*, 2019.

[QDBY21] Ahmed H Qureshi, Jiangeng Dong, Asfiya Baig, and Michael C Yip. Constrained motion planning networks x. *IEEE Transactions on Robotics*, 2021.

[QDCY20] A. H. Qureshi, J. Dong, A. Choe, and M. C. Yip. Neural manipulation planning on constraint manifolds. *IEEE Robotics and Automation Letters*, 5(4):6089–6096, 2020.

[QJQ+20] Ahmed H. Qureshi, Jacob J. Johnson, Yuzhe Qin, Taylor Henderson, Byron Boots, and Michael C. Yip. Composing task-agnostic policies with deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

[QMI+13] Ahmed Hussain Qureshi, Saba Mumtaz, Khawaja Fahad Iqbal, Badar Ali, Yasar Ayaz, Faizan Ahmed, Mannan Saeed Muhammad, Osman Hasan, Whoi Yul Kim, and Moonsoo Ra. Adaptive potential guided directional-rrt. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1887–1892. IEEE, 2013.

[QMSY20] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 2020.

[QNYI17] Ahmed. H Qureshi, Yutaka Nakamura, Yuichiro Yoshikawa, and Hiroshi Ishiguro. Show, attend and interact: Perceivable human-robot social interaction through neural attention q-network. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1639–1645. IEEE, 2017.

[QNYI18] Ahmed Hussain Qureshi, Yutaka Nakamura, Yuichiro Yoshikawa, and Hiroshi Ishiguro. Intrinsically motivated reinforcement learning for human–robot interaction in the real-world. *Neural Networks*, 107:23–33, 2018.

[QSBY19] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE, 2019.

[QY18] Ahmed H Qureshi and Michael C Yip. Deeply informed neural sampling for robot motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6582–6588. IEEE, 2018.

[QYSG17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[RAS$^+$19] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, pages 350–360, 2019.

[RBZ06] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006.

[RC11] Radu B Rusu and S Cousins. Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4, 2011.

[RFG01] Giacomo Rizzolatti, Leonardo Fogassi, and Vittorio Gallese. Neurophysiological mechanisms underlying the understanding and imitation of action. *Nature reviews neuroscience*, 2(9):661, 2001.

[RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[RHB15] C. Rösmann, F. Hoffmann, and T. Bertram. Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *2015 European Control Conference (ECC)*, pages 3352–3357, 2015.

[RVM$^+$11] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 833–840. Omnipress, 2011.

[RZBS09] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009.

[SB$^+$98] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SBFP19] Basak Sakcak, Luca Bascetta, Gianni Ferretti, and Maria Prandini. Sampling-based optimal kinodynamic planning with motion primitives. *Autonomous Robots*, 43(7):1715–1732, 2019.

[SDH$^+$14] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[SDK$^+$20] Anthony Simeonov, Yilun Du, Beomjoon Kim, Francois R Hogan, Joshua Tenenbaum, Pulkit Agrawal, and Alberto Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. *arXiv preprint arXiv:2011.08177*, 2020.

[SGP14] Christoph Salge, Cornelius Glackin, and Daniel Polani. Empowerment–an introduction. In *Guided Self-Organization: Inception*, pages 67–114. Springer, 2014.

[SHK$^+$14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[SJA$^+$18] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

[Ski] S Skiena. Implementing discrete mathematics: combinatorics and graph theory with mathematica1991.

[SKTI17] Himanshu Sahni, Saurabh Kumar, Farhan Tejani, and Charles Isbell. Learning to compose skills. *arXiv preprint arXiv:1711.11289*, 2017.

[SLA$^+$15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[ŞMK12] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.

[SMTF21] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[Spi99] Michael Spivak. A comprehensive introduction to differential geometry. *A Comprehensive Introduction to Differential Geometry. Publish or Perish*, (3), 1999.

[SPNI05] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. In *Robotics research. the eleventh international symposium*, pages 561–572. Springer, 2005.

[Spo98] Mark W Spong. Underactuated mechanical systems. In *Control problems in robotics and automation*, pages 135–150. Springer, 1998.

[SPS99] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[SS83] Jacob T Schwartz and Micha Sharir. On the “piano movers” problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in applied Mathematics*, 4(3):298–351, 1983.

[Sti07] Mike Stilman. Task constrained motion planning in robot joint space. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3074–3081. IEEE, 2007.

[Sti10] Mike Stilman. Global manipulation planning in robot joint space with task constraints. *IEEE Transactions on Robotics*, 26(3):576–584, 2010.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[TKH19] Takayoshi Takayanagi, Yusuke Kurose, and Tatsuya Harada. Hierarchical task planning from object goal state for human-assist robot. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1359–1366. IEEE, 2019.

[TMTR10] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[Tod09] Emanuel Todorov. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems*, pages 1856–1864, 2009.

[TQAN18] Zaid Tahir, Ahmed H Qureshi, Yasar Ayaz, and Raheel Nawaz. Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments. *Robotics and Autonomous Systems*, 108:13–27, 2018.

[TWT+16] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[VMO+16] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, pages 3486–3494, 2016.

[VOS+17] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[VV08] Harini Veeraraghavan and Manuela Veloso. Teaching sequential tasks with repetition through demonstration. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1357–1360. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[WBMW18] Wouter J Wolfslag, Mukunda Bharatheesha, Thomas M Moerland, and Martijn Wisse. Rrt-colearn: towards kinodynamic planning without numerical trajectory optimization. *IEEE Robotics and Automation Letters*, 3(3):1655–1662, 2018.

[WFS07] Mike Vande Weghe, Dave Ferguson, and Siddhartha S Srinivasa. Randomized path planning for redundant manipulators without inverse kinematics. In *2007 7th IEEE-RAS International Conference on Humanoid Robots*, pages 477–482. IEEE, 2007.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[WPC$^+$20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[WVDB13] Dustin J Webb and Jur Van Den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*, pages 5054–5061. IEEE, 2013.

[XMMM$^+$20] Danfei Xu, Ajay Mandlekar, Roberto Martín-Martín, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Deep affordance foresight: Planning through what can be done in the future. *arXiv preprint arXiv:2011.08424*, 2020.

[XNZ$^+$18] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[XvdBPA15] Christopher Xie, Jur van den Berg, Sachin Patil, and Pieter Abbeel. Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4187–4194. IEEE, 2015.

[XXMF20a] Yu Xiang, Christopher Xie, Arsalan Mousavian, and Dieter Fox. Learning rgb-d feature embeddings for unseen object instance segmentation. *arXiv preprint arXiv:2007.15157*, 2020.

[XXMF20b] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on robot learning*, pages 1369–1378. PMLR, 2020.

[YA17] Gu Ye and Ron Alterovitz. guided motion planning. In *Robotics research*, pages 291–307. Springer, 2017.

[YG05] Zhenwang Yao and Kamal Gupta. Path planning with general end-effector constraints: Using task space to guide configuration space search. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1875–1880. IEEE, 2005.

[YKH04] Katsu Yamane, James J Kuffner, and Jessica K Hodgins. Synthesizing animations of human manipulation tasks. In *ACM SIGGRAPH*, pages 532–539, 2004.

[YLK01] Jeffery Howard Yakey, Steven M LaValle, and Lydia E Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17(6):951–958, 2001.

[YLK$^+$17] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 79–86. IEEE, 2017.

[ZHL18] Clark Zhang, Jinwook Huh, and Daniel D Lee. Learning implicit sampling distributions for motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3654–3661. IEEE, 2018.

[ZKB08] Matt Zucker, James Kuffner, and J Andrew Bagnell. Adaptive workspace biasing for sampling-based planners. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3757–3762. IEEE, 2008.

[ZLU18] Chris Zhang, Wenjie Luo, and Raquel Urtasun. Efficient convolutions for real-time semantic segmentation of 3d point clouds. In *2018 International Conference on 3D Vision (3DV)*, pages 399–408. IEEE, 2018.

[ZMBD08] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[ZSS$^+$07] Jeffrey M Zacks, Nicole K Speer, Khena M Swallow, Todd S Braver, and Jeremy R Reynolds. Event perception: a mind-brain perspective. *Psychological bulletin*, 133(2):273, 2007.