Exploring Data Driven approaches for Robot Control : Real Time Visual Servoing and Value Function based Inverse Kinematics

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

Mohammad Nomaan Qureshi
2018111027
mohammad.nomaan@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
March, 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled **"Exploring Data Driven approaches for Robot Control : Real Time Visual Servoing and Value Function based Inverse Kinematics '** by Mohammad Nomaan Qureshi, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. K. Madhava Krishna

To My Family

Acknowledgments

I would like to express my sincere gratitude to everyone who supported me throughout my journey at IIIT Hyderabad. Specifically, I want to thank Prof. Madhava Krishna for his constant support and invaluable guidance. His mentorship played a crucial role in shaping my academic and professional development, and I'm truly grateful for his contributions. Additionally, I would also like to extend my thanks to Dr. Harit Pandya and Prof. Arun Singh for their excellent mentorship, which helped me broaden my knowledge and develop new skills. I would also like to express my appreciation towards Pushkal Katara, Abhinav Gupta, YVS Harish, Prajwal Thakur, Harshit Sankhla, and Shankara Narayana for their support and encouragement throughout my time at IIIT Hyderabad. Their contributions have been instrumental in making my journey at IIIT Hyderabad a truly enriching experience.

I am immensely grateful to my friends at IIIT for adding so much joy to my life. Tanmay, KV, Shivaan, Vedant, Jay, Dhruv, Aakash, and Abhijit have all contributed to making every single day at IIIT a truly delightful experience, and for that, I want to express my sincere thanks. We've had some really fun memories together, and I'll always cherish the moments we spent laughing, joking, and hanging out. They've helped create an incredibly positive and enjoyable environment that made my time at IIIT truly special.

I would like to express my heartfelt gratitude to my parents, who have made countless sacrifices over the years to provide me with a quality education. Their unwavering support and encouragement have been a constant source of strength and inspiration for me, and I cannot thank them enough for everything they have done. I am also immensely grateful to my beloved sisters, who have always been there for me whenever I needed them. Their love and kindness have meant the world to me, and I feel blessed to have them in my life. Lastly, I want to extend my thanks to all the other members of my extended family for their love and support.

Abstract

Data-Driven approaches are becoming increasingly popular for robot control problems. In this thesis, we present two approaches that make use of deep learning frameworks to solve classic robot control problems.

In the first part(chapter 3), we propose a lightweight visual servoing MPC framework that generates optimal control near real-time at a frequency of 10.52 Hz. This work utilizes the differential cross-entropy sampling method for quick and effective control generation along with a lightweight neural network, significantly improving the servoing frequency. We also propose a flow depth normalization layer that ameliorates the issue of inferior predictions of two view depths from the flow network. We conduct extensive experimentation on the Habitat simulator and show a notable decrease in servoing time compared to other approaches that optimize over a time horizon. We achieve the right balance between time and performance for visual servoing in six degrees of freedom (6DoF), while retaining the advantageous MPC formulation.

In the second part(chapter 4), we present a real-time algorithm for computing the optimal sequence and motion trajectories for a fixed-base manipulator to pick and place a set of given objects. The optimality is defined in terms of the total execution time of the sequence or its proxy, the arc length in the joint space. The fundamental complexity stems from the fact that the optimality metric depends on the joint motion, but the task specification is in the end-effector space. Moreover, mapping between a pair of end-effector positions to the shortest arc-length joint trajectory is not analytic; instead, it entails solving a complex trajectory optimization problem. Existing works ignore this complex mapping and use the Euclidean distance in the end-effector space to compute the sequence. In this chapter, we overcome the reliance on the Euclidean distance heuristic by introducing a novel data-driven technique to estimate the optimal arc-length cost in joint space (a.k.a the value function) between two given end-effector positions. We parametrize the value function as a Neural Network and motivate a niche choice for its architecture, inspired by the works on metric learning. The learned value function is then used as an edge cost in a capacitated vehicle routing problem (CVRP) set-up to compute the optimal visitation sequence. Finally, we optimize over the input space of the learned value function network to propose a novel Inverse Kinematics (IK) algorithm that produces substantially shorter joint arc-length trajectories than existing approaches while executing the computed optimal sequence. We show that our sequence planner, in combination with our proposed IK contoller, offers a substantial improvement in joint arc length over existing state-of-the-art while maintaining scalability to a large number of objects.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning-based frameworks provide a powerful paradigm that can be applied to solve problems across domains. Recently there has been a lot of interest in solving classical robot control problems with data-driven function approximators. While classical robot control approaches can solve many tasks in controlled environments with convergence guarantees, they suffer from poor generalization to even small perturbations to scene configuration. They also often lack the ability to handle complex input such as raw images. Deep learning-based frameworks offer the ability to generalize across scenes and can handle complex inputs. Hence solving robot control problems is compelling.

In this thesis, we explore learning approaches for two of the central control problems in robotics. In the first part of the thesis (Chapter3) we study visual servoing which uses a feedback information from the vision sensor for navigating the robot to a desired goal image. This involves generating a set of actions that move the robot in response to the observation from the camera. Traditionally, visual servoing has relied on manually engineered features such as points, lines, and contours, as described in previous works [12, 13, 14]. Nevertheless, the use of appearance-based features in these methods tends to result in imprecise matching when the camera undergoes significant transformations. To address this limitation, contemporary data-driven visual servoing methodologies have turned to utilize deep neural features, as exemplified in recent works [1, 2]. In these initial learning-based approaches, the objective is to acquire knowledge of the relative camera pose through supervised learning with a pair of images as input, leading to precision within the sub-millimeter range. Nevertheless, since these techniques were excessively trained on a single environment, it proved challenging to generalize them to new, unfamiliar environments. Recently, a more principled approach for combining deep flow features with a classical image-based controller was proposed in [5] to improve generalization to novel environments without any retraining.

One of the key aspects of visual servoing is how we design the underlying controller. Katara et al.[15] proposed an approach that employs deep flow features and proposes a kinematic model based on flow. For solving their deep MPC formulation, they further proposed a recurrent neural network (RNN) based optimization routine that generates velocities to optimize their flow-based loss. However, generating control commands for an RNN-based approach is extremely time-consuming and not suit-

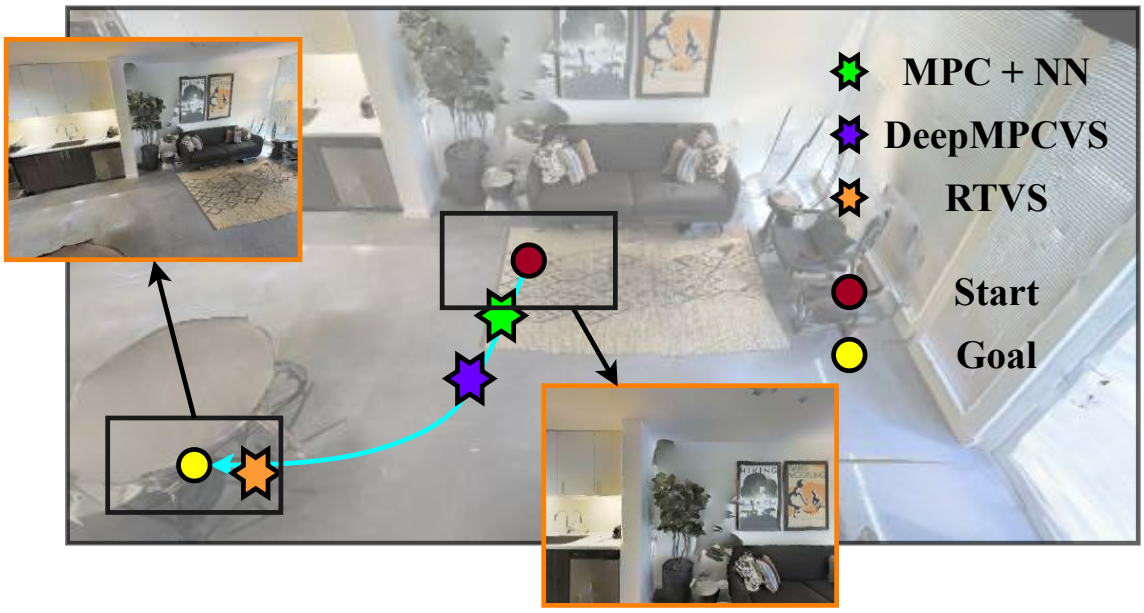

Figure 1.1: RTVS method proposed in Chapter3 shows a significant improvement in servoing frequency vis-à-vis other deep MPC-based visual servoing approaches in 6DoF without compromising its ability to attain precise alignments. Our controller generates optimal control in real-time at a frequency of 66 Hz (excluding optical flow overheads) and successfully servos to its destination, while other approaches still lag behind.

able for real-world deployment. Single-step methods are known to be faster in computing immediate control. However, these approaches do not include the benefits of the MPC formulation. However, these approaches do not include the benefits of the MPC optimization formulation. We aim to make the best of both worlds by improving the control generation frequency while retaining the advantageous MPC formulation. Fig. 1.1 showcases this boost obtained by our controller for an example scene in our benchmark, where our controller beats the recent deep MPC-based approaches for reaching the goal.

Our contributions in Chapter 3 are summarised as follows:

- We propose a lightweight and real-time visual servoing framework, which can be trained on-the-fly in an unsupervised fashion and improve the work of [6]. This work achieves a significant increase in the servoing frequency, computing control near real-time at 0.095 seconds per MPC iteration, and is around 10 times faster than existing deep MPC-based approaches.
- The decrease in time is made possible with the help of our control generator network, which uses a slim and lightweight neural network that significantly reduces the number of iterations required in each MPC step. We use the differential cross-entropy method [16] which helps us sample intelligently and generate optimal control commands in 6DoF.

- Furthermore, we introduce a flow depth normalization layer that accounts for inferior flow predictions from FlowNetC [17], thereby reducing our network's dependency on the accuracy of the flow.

We then look into the problem of performing a sequence of pick and place operation which is a vital component in a multitude of industries, including but not limited to retail and manufacturing. In this part of the thesis (Chapter4) we focus on two main modules of the problem: the first involves obtaining the optimal sequence in which to visit the objects and the second pertains to the execution of the optimal sequence i.e. control of joint trajectories to execute the optimal sequence. We henceforth call it as the **Sequencer** and the **Executor** modules and bring in novelties in both of them to move beyond the state-of-the-art.

A fundamental challenge in computing an optimal visitation sequence for pick and place is estimating the cost-to-go between two manipulator end-effector positions. A useful cost-to-go metric can be the optimal (or shortest) joint space arc length between two positions. However, this cost-to-go is not known analytically and in fact, requires one to solve a non-convex optimization problem. To see why this is challenging, imagine that the manipulator end-effector needs to visit 4 positions and thus there are $4!$ sequences possible. To estimate the cost of all the sequences, one needs to solve the same number of optimization problems. This is obviously intractable beyond sequences for visiting a small number of objects. Existing works [11] bypass this intractability by using the simple Euclidean distance as the cost-to-go between two positions. However, we show in the present work that such approaches can lead to sub-optimal performance. To be precise, the shortest Cartesian/Euclidean space trajectory that connects two end effector positions can have an unduly long arc length in the joint space. Vice-versa, the shortest arc-length trajectory in the joint space need not be a straight line in the end-effector space. This is exemplified in Fig. 1.2.

Beyond the **Sequencer**, a successful pick and place also need an efficient **Executor** that can compute the joint velocities (or simply joint angles) to realize the optimal visitation sequence. Typically, this is done by invoking one of the many Inverse Kinematics (IK) algorithms, although more sophisticated choices such as optimal control planners are possible. However, existing IK algorithms, either implicitly or explicitly use the Euclidean distance heuristic to move between two given positions. Thus they are inherently sub-optimal with respect to minimizing joint arc length and consequently pick and place completion time.

Our contributions in Chapter 4 are summarised as follows:

- *Algorithmic:* This chapter overcomes the reliance on the Euclidean distance heuristic by adopting a data-driven approach. In particular, we train a neural network in a supervised fashion to estimate the optimal arc length/value function in joint space for a given pair of end-effector positions. [1]. This estimate can then be integrated into any off-the-shelf sequencing algorithm. For example, in our case, we formulate pick and place sequence computation using the template of capacitated

[1]The estimate of the optimal cost is often referred to as the value function in optimal control literature

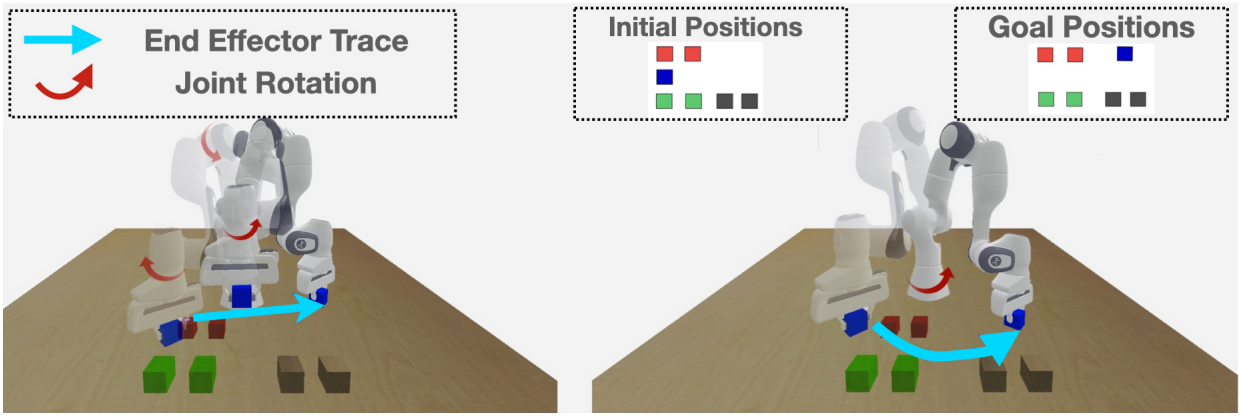

Figure 1.2: Our approach in Chapter4 shows a significant improvement in joint space arc length vis-a-vis other approaches. At the core of our approach is a learned value function that provides an estimate of the optimal joint space arc length between two manipulator positions. The importance of this value function is shown in the figures. A straight-line trajectory between a start and a goal manipulator position provides the shortest path in Euclidean space but involves rotating multiple joints. In contrast, a trajectory that has the shortest arc length in the joint space and consequently shorter execution time appears curved in the end-effector space. Our learned value function allows us to obtain the optimal sequence to pick and place a given set of objects and the associated joint velocities to execute the sequence.

vehicle routing problem (CVRP). We define the edge weight of the CVRP graph as our estimated value function between the graph nodes. [2] We further use optimization over the input space of the learned value function to develop a novel IK that retains real-time applicability and at the same time results in shorter joint trajectories between two given positions.

- *State-of-the-art Performance:* We vindicate the efficacy of the proposed **Sequencer** and **Executor** modules by showcasing their superior performance over various baselines as well as existing state-of-the-art (SOTA) RoboTSP[11]. We outperform the baselines stemming from different ablations of **Sequencer** and **Executor**, where the latter could be ROS-KDL[9], PyBullet[10] IK planner, RoboTSP[11]).

[2]Note that although CVRP is extensively used for sequence planning, ours is the first approach to augment it with a learned edge cost to achieve remarkable improvements in pick and place completion time.

1.1 Thesis Organisation

We first desribe on a high level in Chapter2 problems explored in the thesis i.e. Image Based Visual Servoing and Optimal Pick-Place Sequence Planning and relevant literature to the problems. In Chapter3 discuss the Real Time Visual Servoing(RTVS) method for fast multi-step visual servoing. We conduct extensive experimentation on the Habitat simulator and show a notable decrease in servoing-time compared to other approaches that optimize over a time horizon. In Chapter4 we describe our method for Pick-and-Place sequence planning. We show that our sequence planner, in combination with our proposed IK, offers a substantial improvement in joint arc length over existing state-of-the-art while maintaining scalability to a large number of objects.

Chapter 2

Background

In this chapter, we will give a brief overview of the visual servoing and optimal pick and place sequencing problems and relevant literurature related to the problems.

2.1 Visual Servoing

Visual Servoing is a fundamental problem in the field of robotics, which involves controlling the motion of a robot using information obtained from a camera or vision sensor. The objective is to achieve a final pose of the robot, specified in terms of a 'target image', by using an 'initial image' as a starting point. In a standard Image Based Visual Servoing Pipeline, two key components are present: a feature generator and a controller. The feature generator extracts relevant features from the 'target' and 'current' images, whereas the controller aims to minimize the difference between the extracted features. The ultimate objective of the servoing problem is to design an effective feature extractor and controller, enabling the agent to reach the 'target image'. A typical Image-based Visual Servoing Pipeline is illustrated in Fig. 2.1.

2.1.1 Related Work

2.1.1.1 Feature Generation

Classically, hand-crafted features such as points [12], lines [13] and contours [14] were employed for visual servoing. However, such appearance based features result in inaccurate matching for larger camera transformations. To circumvent this bottleneck, recent data driven visual servoing approaches resort to deep neural features [1, 2]. Initial learning based approaches [1, 2] aim to learn the relative camera pose from a pair of images in a supervised fashion and as a result, they were able to achieve a sub-millimeter precision. However, as they were over-trained on a single environment, generalisation to new environments was difficult. Saxena et al. [3] trained their approach on multiple environments and thus, their approach generalised to a certain extent. But such approaches still require supervision and cannot be trained on-the-fly. Recently, a more principled approach for combining deep flow features with a

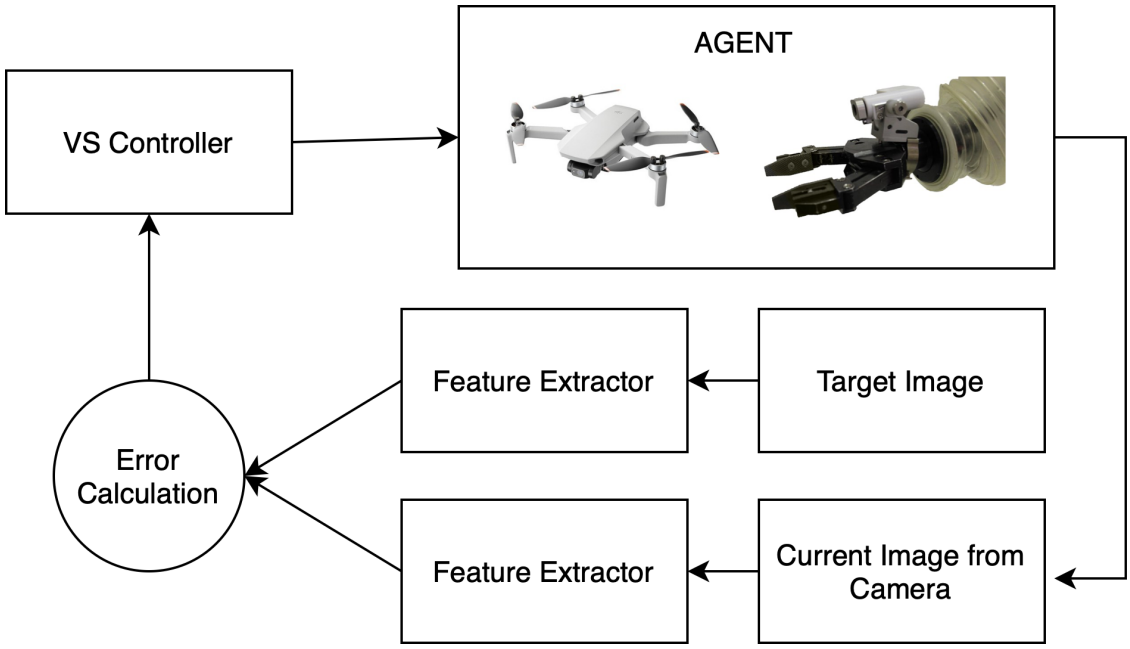

Figure 2.1: A generic image-based visual servoing pipeline. The goal is to reach the desired pose which is specified using the target image. A camera is mounted on an agent, and feedback from it is used to generate the motor velocities to reach the desired image.

classical image based controller was proposed in [5] to improve generalisation to novel environments without any retraining.

2.1.1.2 Controller Design

Controller design is another crucial aspect of visual servoing; classically, image based and position based controllers were presented in the literature. However, these controllers take the action greedily and do not consider a long term horizon, which could lead to problems such as the loss of features from field of view, getting stuck in local minima and larger trajectory lengths. Thus, to achieve a better controller performance, optimal control [18] and model predictive control [19] based formulations were proposed in the visual servoing domain. Another advantage of casting visual servoing in a model predictive control is to cater for additional constraints such as robot dynamics, field of view and obstacle avoidance. While the MPC formulations of visual servoing are lucrative, they assume accurate feature correspondences. Furthermore, they rely on classical MPC solvers that do not scale up to high dimensional features such as images or flows, which makes it difficult for classical MPC based visual servoing approaches to be employed for modern deep features.

Recently, several deep reinforcement-based visual navigation approaches were proposed [20, 21] that present a neural path planning framework for visual goal-reaching. By the virtue of being a model-free reinforcement learning framework, they are sample inefficient, and do not present results in 6DoF planning. On the other hand, deep model-based reinforcement learning frameworks [19], [22] aim to jointly learn the controller as well as the underlying system dynamics (model) through data, thus making it difficult to generalize for new environments.

Katara et al. [6] use an approach which employs deep flow features and propose a kinematic model based on flow. For solving their deep MPC formulation, they further proposed a recurrent neural network (RNN) based optimisation routine, that generates velocities to optimise their flow based loss. As a baseline, they also employ a vanilla neural network instead of an RNN, for solving their MPC problem. While their proposed approach was able to solve the MPC problem on-the-fly in a receding horizon fashion and achieve convergence, it has costly overheads with respect to the total servoing time. DeepMPCVS [6] requires the training of a recurrent neural network online which is computationally expensive, narrowing down their scope of performance in real world scenarios. Single step methods like [5] are known to be faster in computing immediate control. However, these approaches do not include the benefits of the MPC optimisation formulation. We aim to make the best of both worlds, by improving the control generation frequency while retaining the advantageous MPC formulation.

2.2 Optimal Pick and Place Sequence Planning

Performing a sequence of pick and place operation is an extricable part of many industries like retail, manufacturing etc. Typical industrial scenarios include sorting and packing objects, assembly of parts

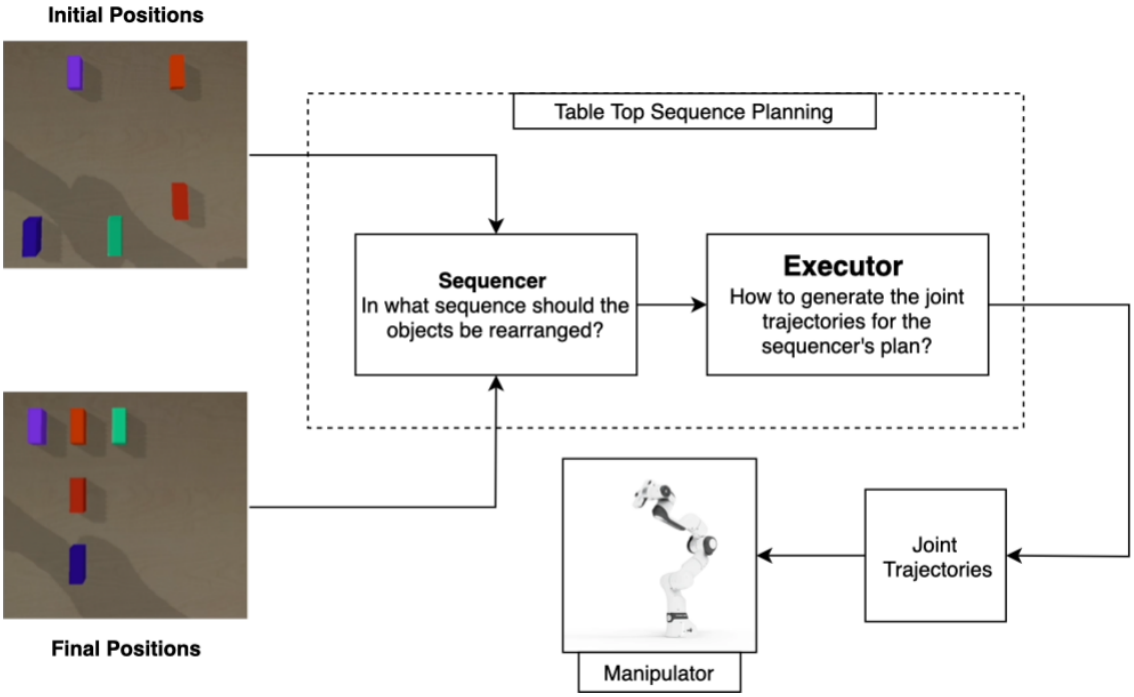

Figure 2.2: A generic pick and place sequence planning Pipeline.

in a certain order, tiding a meeting table, making a service robot for a variety of space management work. A typical pick and place sequencing scenario is shown in Fig. 2.2 where the Franka Panda arm maneuvers the objects on the table to reach the desired configuration from the initial one. Our formulation, however has applicability beyond pick and place. For example, it can be used to compute the sequence in which manipulator should drills at a set of marked positions [11].

There are two main modules of the pick-place problem: the first involves obtaining the optimal sequence in which to visit the objects and the second pertains to the execution of the optimal sequence. We henceforth call it as the **Sequencer** and the **Executor** modules.

2.2.1 Related Works

2.2.1.1 Task and Motion Planning for Pick and Place Operations

We begin by pointing out an extensive summary of the state-of-the-art [23] that presents an extensive review of classical sequencing and motion planning algorithms

Pick and Place sequence planning is often seen as a subset of a larger category of manipulator task planning problems. There are a variety of applications that surround it such as drilling [11], fruit picking [24] or table-top rearrangement [25]. Inevitably, a **Sequencer** and a **Executor** modules are an essential part of the solution pipeline. The **Sequencer** in existing works have relied on a variety of combinatorial

optimization methods such as the regular Travelling Salesman Problem (TSP) [26], Asymmetric-TSP (ATSP) [27] [28], Generalized TSP (GTSP), [29], Shortest Sequencing Problem (SSP++), [30, 11]. The solvers used range from Branch and Bound based global optimization, Constraint Solvers, and Genetic Algorithms. A detailed review can be seen in [23] that also explains the underlying principles behind the TSP variants briefly. Pick and Place sequencing is also closely related to the problem of table-top re-arrangement with overhand grasps and non-overlapping start and goal positions [31], [32].

Once the sequence is computed, the execution involves another optimization layer over multiple joint trajectories available to reach a particular end-effector position. Authors in [11] propose a combination of multiple IK searches along with Dijkstra shortest path planning as their **Executor** module. A graph search technique is proposed in [30] while [33] uses the Probabilistic Road-Map-based planner. Authors in [32] use Randomly Exploring Random trees as their **Executor** module.

All the above-cited works relied on the Euclidean distance heuristic between the start and goal position of the object for both sequencings and in executing the sequence. Besides the very niche set-up of a Cartesian robot in [31], such heuristics can provide misleading results for a generic manipulator. As stated earlier, our work replaces the Euclidean heuristic with a data-driven metric.

2.2.1.2 Metric Learning

Metric learning has a rich history in the motion planning literature. A classic use of learned metrics can be found in sampling-based motion planning. For example, authors in [34], [35] learned the distance between two sampled nodes of Randomly Exploring Random Trees for robots with non-holonomic kinematics. Interestingly, the motivation in these cited works is very similar to our own in the sense that the Euclidean distance between a pair of positions provides a poor estimate of the true cost-to-go for many robotic systems. A similar attempt can be found in [36], where the authors learn an embedding of the state-space where the Euclidean distance will prove sufficient.

Our work is inspired by metric learning literature as we also intend the learn the true cost-to-go between a pair of start and a goal positions. However, the difference stems from the use of structured deep neural networks.

2.2.1.3 Estimating Value Function

Our **Executor** builds upon the connection between value function learning and model predictive control. In particular, works like [37], and [38] show how offline computed optimal trajectories can be used to learn a global value function, which, in turn, can be used to reduce the planning horizon of the MPC. Our **Executor** minimizes the one-step value function learned from recorded data to provide real-time, reactive motion generation with the optimal joint space arc length.

Chapter 3

RTVS: A Lightweight Differentiable MPC Framework for Real-Time Visual Servoing

3.1 INTRODUCTION

In this Chapter, we extend and improve the work of [6] and present a lightweight MPC framework for real-time image based visual servoing (IBVS) which generates control in real-time and can be trained online in an unsupervised fashion. We leverage the deep flow-based MPC framework proposed in [6] and in order to make the approach faster and feasible for real-time systems, we present a novel lightweight *directional neural network* which aims to encode the relative pose between the current and final image in its parameters, thereby significantly reducing the number of iterations required in each MPC step as explained in 3.2.2.2. Further, we utilise the differential cross entropy method (DCEM) proposed in [16] for differentiable sampling of control. Our controller can attain the desired pose around 10 times faster than the recent deep multi-step ahead formulations [6], with a 74% reduction in servoing time. This decrease in the servoing frequency leads to a significantly shorter delay between successive control commands, leading to minimal number of jerks, which is crucial for aerial robots. We compute immediate control at the same rate as single-step methods like [5], which are also fast but lack the advantages of the MPC optimization formulation and do not optimize over a time horizon. Our proposed architecture is sample-efficient and uses an optimal control generator network, which improves the servoing rate without a heavy compromise on its performance. Furthermore, our architecture also includes a flow normalization layer, which reduces the error in flow depth estimates, thereby accounting for inaccurate flow predictions.

3.2 Approach

Existing deep learning based MPC approaches like [6] formulate visual servoing as an MPC optimisation which consider a long term horizon and can be trained on-the-fly in an unsupervised fashion. However, [6] requires the online training of an RNN, which is computationally expensive and faces

Approach	Unsupervised?	Real Time?	Multi-Step Ahead?
Yu et al. [4]	✗	✓	✗
Bateux et al. [5]	✗	✓	✗
PhotoVS [13]	✓	✓	✗
ServoNet [6]	✗	✓	✗
DFVS [7]	✓	✓	✗
DeepMPCVS [14]	✓	✗	✓
RTVS [Ours]	✓	✓	✓

Figure 3.1: Approaches ([1], [2], [3]), although real-time, are not multi-step ahead and require supervision. PhotoVS [4] and ServoNet [3] fail to converge on our benchmark, as shown in 3.5. DFVS [5] shows strict convergence in real-time, but optimises greedily and does not consider a long term horizon. DeepMPCVS [6] uses the advantageous MPC formulation and generates control in an unsupervised fashion, but is computationally expensive. Our approach makes the best of all worlds: it generates control in real-time and trains on-the-fly, whilst retaining the MPC formulation.

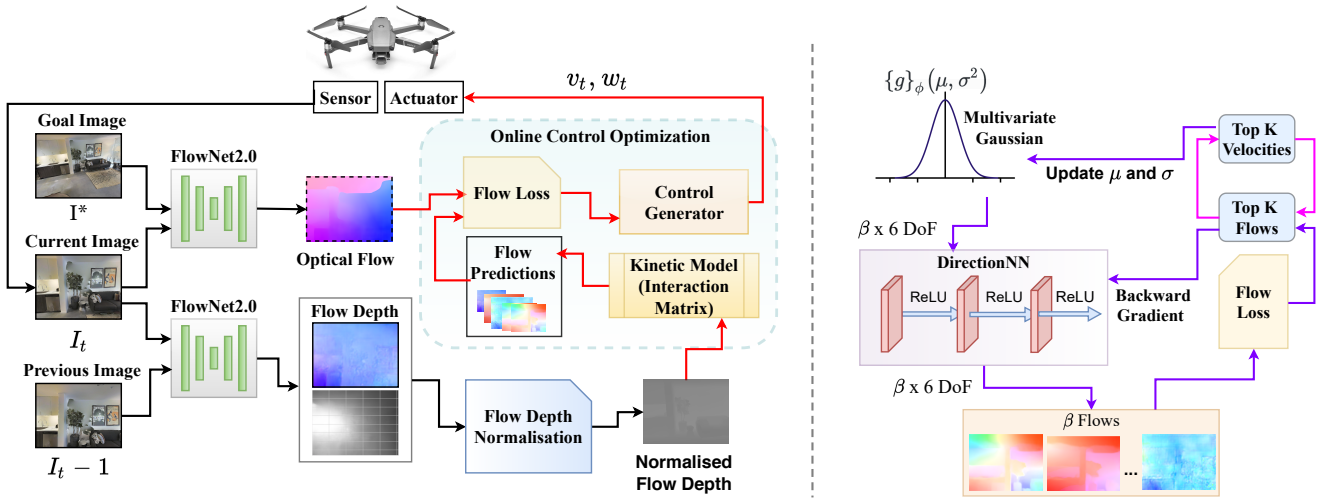

Figure 3.2: **Left**: We demonstrate our MPC based optimisation framework to perform visual servoing in 6DoF. In each MPC optimization step, we use FlowNetC to predict the optical flow between the current image I_t and goal image I_*, which acts as the target flow for control generation. The flow computed between the current and previous image acts as a proxy for depth, which after normalisation, is fed into the interaction matrix. Our control generator then optimises the flow loss to generate control commands. **Right**: This figure depicts our control generation architecture. We sample a (β x 6) vector from a multivariate Gaussian distribution and pass it through our Directional Neural Network. The kinetic model generates β pseudo-flows and the flow loss is computed for each, against the target flow. In each MPC step, we pick the top K velocities corresponding to the K least flow errors to update the Gaussian parameters, while the mean velocity is fed to the actuator.

challenges for real-time systems. In order to mitigate this issue, we propose a novel lightweight control generation architecture which effectively samples candidate velocities, leading to a significant improvement in the control generation time and hence, the servoing time. Our approach achieves the right trade-off between the servoing rate and performance (attainment of precise alignments) through an intelligent sampling strategy (Section 3.2.2.1) and a slim neural network architecture trained on-the-fly (Section 3.2.2.2). We compute the scene's depth Z_t as an inverted scale representation of the magnitude of optical flow between the current image I_t and previous image I_{t-1} as in [5], and pass it through the flow depth normalisation layer to generate effective scene-depth from optical flow, as described in (Section 3.2.3). We illustrate our approach in Fig. 3.2 and summarise it through algorithm 1. In algorithm 1, the steps in red signify our modifications to the MPC algorithm proposed in [6] to make it feasible for real-time systems.

3.2.1 Problem Formulation

Given that the measurements of the robot sensor are monocular RGB images, with I^* being the desired image and I_t being the current observation at any time instant t, our aim is to generate optimal control commands $[v_t, \omega_t]$, where v_t is the linear velocity and ω_t is the angular velocity at time step t, in order to solve the fundamental IBVS objective of minimising the photometric error $e_t = \|I_t - I^*\|$ between I^* and I_t. We build our approach upon the kinetic model and MPC objective proposed in [6]. The MPC objective is given as

$$\mathbf{v}_t^* = \arg\min_{\mathbf{v}_t} \|\mathcal{F}(I_t, I^*) - \widehat{\mathcal{F}}(\mathbf{v}_t)\| \tag{3.1}$$

where $\mathcal{F}(I_t, I^*)$ is the target flow between I_t and I^* and $\widehat{\mathcal{F}}(\mathbf{v}_t)$ is the pseudo-flow generated through the predictive model:

$$\widehat{\mathcal{F}}(\mathbf{v}_{t+1:t+T}) = \sum_{k=1}^{T} [L(Z_t)\mathbf{v}_{t+k}] \tag{3.2}$$

where $L(Z_t)$ is the interaction matrix which relates the rate of change of features in the camera plane to the image plane, and $\mathbf{v}_t$ is the optimal control. The image coordinates x, y and Z_t being the scene's depth, the interaction matrix is generated as

$$L(Z_t) = \begin{bmatrix} -1/Z_t & 0 & x/Z_t & xy & -(1+x^2) & y \\ 0 & -1/Z_t & y/Z_t & 1+y^2 & -xy & -x \end{bmatrix}. \tag{3.3}$$

In [6], the online trajectory optimisation exploits the additive nature of optical flow and employs a recurrent neural network to generate velocity commands, which intuitively, tries to optimise over a fixed time-horizon in each MPC optimisation step. The approach tries to solve each time-step individually, exploiting the recurrence property of LSTM architectures which are computationally expensive and data-hungry. We modify the predictive model shown in eq. 3.2 to achieve an optimal trade-off between

Algorithm 1 Visual Servoing MPC Framework with Fast Control Generation Optimisation.

Require: I^*, ϵ ▷ Goal Image, convergence constant

Initialise μ, σ^2 ▷ Initialise Gaussian Distribution sampling parameters

while $\|I - I^*\| \leq \epsilon$ **do** ▷ Convergence criterion

  $I_t :=$ get-current-obs() ▷ Obtain the current RGB observation from sensor

  predict-target-flow ($\mathcal{F}(I_t, I^*)$) ▷ Predict target flow using flow network

  $L_t :=$ compute-interaction-matrix($\mathcal{F}(I_t, I_{t-1})$) ▷ Kinetic Model Generation

  for $m = 0 \rightarrow M$ **do** ▷ On the fly training for DirectionNN

    $[\beta_i]_{i=1}^{N} \sim g_\phi(\mu, \sigma^2)$ ▷ Sampling a β x 6 vector from Gaussian Distribution

    $[\mathbf{v}_{t,i}]_{i=1}^{N} = f_\theta([\beta_i]_{i=1}^{N})$ ▷ Predict β velocities from DirectionNN

    $[\widehat{\mathcal{F}}(\mathbf{v}_{t,i})]_{i=1}^{N} = [L_t(Z_t)\mathbf{v}_{t,i}]_{i=1}^{N}$ ▷ Generate β pseudo-flow using predictive model

    $[\mathcal{L}_{flow}]_{i=1}^{N} := [\|\widehat{\mathcal{F}}(\mathbf{v}_{t,i}) - \mathcal{F}(I_t, I^*)\|]_{i=1}^{N}$ ▷ Compute Flow Loss

    $\theta_{m+1} := \theta_m - \eta \nabla \mathcal{L}_{flow}$ ▷ Update DirectionNN parameters

    $\mu_{t+1} = 1/k \sum_i^k v_{t,i}$ ▷ Update mean of the Gaussian Distribution

    $\sigma_{t+1}^2 = 1/k \sum_i^k (\mu_{t,i} - \mu_{t+1})^2$ ▷ Update standard deviation of the Gaussian Distribution

  end for

  $\widehat{\mathbf{v}}_{t+1} := \arg\min_{v \in V} \mathcal{L}_{flow}$ ▷ Execute control command in the environment

end while

speed and accuracy, by connecting the MPC objective (which is to minimise the error between the predicted flow and target flow) with a differential sampling based strategy through a slim and lightweight neural network. We introduce a *horizon* parameter 'h' in order to scale the predicted flow to learn to predict the mean optical flow, rather than predicting over a time-horizon.

$$\widehat{\mathcal{F}}(\mathbf{v}_t) = h * L(Z_t)(\mathbf{v}_t) \tag{3.4}$$

Thus, we formulate our loss function as shown in eq. 3.5 to train our control generation network on-the-fly.

$$\mathcal{L}_{flow} = \|\widehat{\mathcal{F}}(\widehat{\mathbf{v}}_t) - \mathcal{F}(I_t, I^*)\| = \|[L(Z_t)\widehat{\mathbf{v}}] - \mathcal{F}(I_t, I^*)\| \tag{3.5}$$

We regress the pseudo flow $\widehat{\mathcal{F}}(\mathbf{v}_t)$ with the target flow $\mathcal{F}(I_t, I^*)$ using a mean squared error loss. We summarize the adaptive sampling and network training process of our control generation architecture in Section 3.2.2. After each MPC optimisation step, we apply the velocity to the agent and get the new measurements from the sensor, repeating the optimisation process for each subsequent step until the IBVS objective is met. We summarize the MPC optimisation process through Fig. 3.2.

3.2.2 Optimal Control Generation Architecture

In this section, we provide details about the differential sampling-based strategy to generate optimal control and our lightweight neural network architecture.

3.2.2.1 Intelligent DCEM-based Sampling

Due to the high dimensionality of the flow representations, it becomes difficult for classical MPC solvers to optimise the objective function in equation 3.1. The cross entropy method (CEM) [39] is an attractive formulation to solve complex control optimisation problems involving objective functions which are non-convex, deterministic and continuous in nature by solving the equation,

$$\hat{F} = \arg\min_{F} \mathcal{E}_v(F) \tag{3.6}$$

where $\mathcal{E}_v(F)$ is the objective function having parameters *v* over the n-dimensional space. However, there are a few shortcomings to using this approach. It is a gradient-free optimisation approach where Gaussian parameters are refitted only in a single optimisation step, which does not allow adaptive sampling over consecutive MPC steps i.e. it is unable to preserve the memory of effective sampling. Moreover, the parameter optimisation is not based on the downstream task's objective function which might lead to a sub-optimal solution, which in our case would lead to an increase in the number of MPC steps, directly affecting the servoing time. In this work, we take inspiration from the differential cross entropy method (DCEM) [16] which parameterises the objective function $\mathcal{E}_v(F)$, making it differentiable with respect to v. We introduce non-linearity in our control generation process with the addition of a neural

network (3.2.2.2), in order to retain information throughout all MPC steps. This connects the sampling strategy with the objective function, making CEM an end-to-end learning process. Hence, the Gaussian parameters are updated with subsequent MPC optimisation steps, enabling adaptive sampling over the process based on the MPC objective. This allows us to sample from a latent-space of more reasonable control solutions.

3.2.2.2 DirectionNN

We introduce a slim neural network called the 'DirectionNN', whose design aims to encode a sense of relative pose between the current image I_t and the goal image I_*. We keep the network lightweight in order to achieve a high servoing rate and incorporate online learning of relative pose updates in each MPC optimisation step. Since we execute small steps in each iteration, the network is capable of learning the change in relative pose quickly. Moreover, the weights of the network are also retained in each MPC iteration, which can be reused since there is a minimal change in the agent's image measurements.

Multiplying with the horizon h (eq. 3.4) checks the merit of direction predicted by the DirectionNN over an extended time. Hence, we can train our approach for lesser number of iterations in each MPC step, which significantly decreases the total servoing time.

The network architecture has an input layer consisting of 6 neurons followed by 4 fully connected hidden layers with 16, 32, 256, 64 neurons respectively and an output layer with 6 neurons, which represents the 6DoF velocity vector. We apply ReLU activation at the output of each hidden layer and Sigmoid activation on the last layer. We further scale the 6-D output between -1 and 1 to vectorise it as a 6DoF velocity vector. We train the network in each MPC step. The inputs to the network are intelligently sampled from a Gaussian distribution.

3.2.2.3 Sampling and Learning

We use the DCEM sampling strategy explained in 3.2.2.1 along with DirectionNN explained in 3.2.2.2 to generate optimal control. In each MPC optimisation step, the network samples a β (batch size) x 6 dimensional vector from a Gaussian distribution $g_\phi(\mu, \sigma^2)$ and carries out a forward pass, generating β samples of 6 DoF velocity commands. We compute the pixel-wise target flow $\mathcal{F}(I_t, I^*)$ between I_t and I^* using a pretrained FlowNetC [17] model. Moreover, we apply a kernel with a filter size of (7x7) and a stride of 7 to the target flow $\mathcal{F}(I_t, I^*)$ and pseudo-flow $\widehat{\mathcal{F}}(\mathbf{v}_t)$. The kernel K, consists of only one non-zero value with K[0, 0] = 1.

The weights of the DirectionNN are updated through gradient descent. We use the Adam optimiser with a learning rate of 0.005 to train our network. The sampling parameters of the Gaussian distribution $g_\phi(\mu, \sigma^2)$ are optimised for subsequent steps. We update (μ, σ) by the mean and variance of top K velocities corresponding to the top K least flow errors. For our approach, we sample 8 velocities and compute the flow loss for each, and choose the velocity corresponding to the least flow loss, which is applied to our agent. We also multiply the horizon parameter with the generated velocity before

computing the flow loss. The MPC optimisation steps are repeated until the convergence criteria to reach the goal location is met.

3.2.3 Two View Depth Normalisation

We further enhance the two view depth estimation method proposed in [5] to account for inferior flow predictions, since we use FlowNetC [17] without any retraining/fine-tuning. The magnitude of flow values predicted by [17] has high variance from pixel to pixel due to the non-planar structure of the scene and as a result, there are heavy outliers while using the predicted flow as a proxy for depth. These outliers can hurt the performance of the controller, because the magnitude of velocity is very sensitive to depth values. Hence, we require a stable flow depth to guide our approach. We use the sigmoid function (equation 3.7) to scale and normalise the flow values before feeding them to our kinetic model - the interaction matrix. We compare the effect of flow normalisation on the mean squared error for three scenes, as shown in TABLE 3.1.

$$Z_s(x, y) = \nu(\frac{1}{1 + e^{-Z}} - 0.5) \tag{3.7}$$

Here, Z is the two view depth proposed in [5], ν is the scaling factor which we have selected as 0.4 and Z_s is the scaled inverse used as proxy for depth in the interaction matrix.

Scene	MSE Flowdepth	MSE Normalised Flowdepth
Quantico	160.04	49.72
Arkansaw	469.72	63.44
Ballou	508.81	122.78

Table 3.1: Comparison of effect of flow normalisation on MSE for different scenes. A significant decrease in Mean Squared Error is observed when the flowdepth is normalised

3.3 Experiments

The motivation behind our work is to present an online control generation strategy that can generate optimal 6DoF robot commands on-the-fly and in real-time, while not compromising on performance in terms of convergence and alignment. We benchmark our strategy on 8 indoor 3D photo-realistic baseline environments from the Gibson dataset [40] in the Habitat simulation engine [41] similar to [5]. These baseline environments span various levels of difficulty based on parameters such as the extent of overlap, the amount of texture present and the rotational and translational complexities between initial

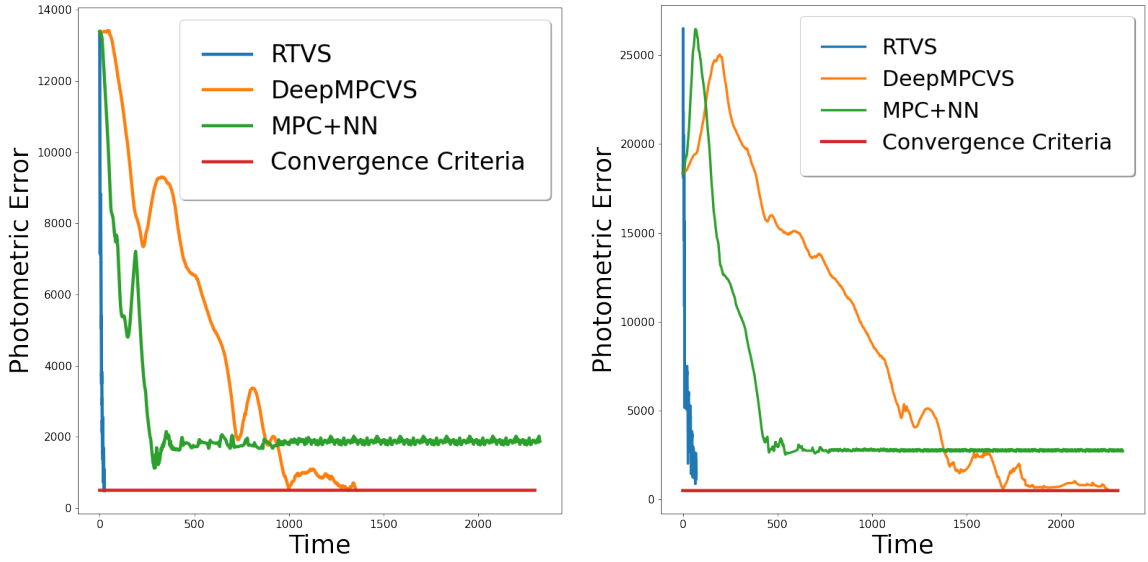

Figure 3.3: **Time Comparisons**: We obtain a significant improvement in servoing rate over other deep MPC based visual servoing approaches. Here, we show the evolution of photometric error with time on two environments. Our approach is the fastest to achieve convergence, without any compromise on the convergence criteria.

and desired image. We use a free-flying RGB camera as our agent, which can navigate in all six degrees of freedom.

Through this section, we show quantitative and qualitative results to validate our approach. This work generates control at 0.015 seconds per MPC step (excluding flow overheads) and attains photometric convergence faster than the other methods which optimise over a time horizon. While achieving a significant improvement in servoing rate, our approach does not compromise on performance in terms of pose and photometric errors, which is comparable to the established baselines. Our method makes a sacrifice on its trajectory length, but generates control in real-time and achieves precise alignments, thereby finding the right trade-off between speed and performance.

We compare our approach with (a) DeepMPCVS [6] and (b) MPC+NN [6], both of which are MPC formulations that optimise over a time horizon. The approach in [6] is optimal but computationally heavy due to its bulky architecture. We achieve a significant improvement in servoing rate vis-à-vis these baselines. Approaches like [5] are fast in computing the immediate control, but they optimise greedily and cannot incorporate additional constraints. We achieve similar servoing rates with such single-step approaches as well.

3.3.1 Convergence Study and Time Comparison

We test our approach across all environments in our simulation benchmark and report the per MPC-step time (per IBVS-step time in case of [5]) with and without overheads from flow computation, the number of steps taken for convergence and the total time for a visual servoing run to reach the goal image, averaged over all scenes as shown in TABLE 3.2. Our lightweight control generation architecture and intelligent sampling strategy help achieve a per MPC-step time of 0.015 seconds (this is the time taken for the MPC optimisation step, excluding the overheads from the flow network). We only train for 1 iteration in every MPC step (as opposed to 100 required in [6]) and retain the weights of our neural network, since there is no substantial change in the velocities for immediate MPC steps, which helps lower the per MPC-step time. The entire time taken for a MPC-step, after including flow overheads, is 0.095 seconds, resulting in a near real-time control generation at a frequency of 10.52 Hz. We attain strict convergence of photometric error $<$500 and outperform the other multi-step ahead approaches [6] in terms of the total servoing time. We use a *Nvidia Geforce GTX 1080-Ti Pascal* GPU for our experimentation.

Approaches	Time w.o flow	Time w. flow	Total Iters.	Total Time
MPC+NN [6]	0.75	1.10	344.22	378*
DFVS [5]	**0.001**	0.21	994.88	209
DeepMPCVS [6]	0.8	1.15	**569.63**	655
RTVS [Ours]	0.015	**0.095**	1751.25	**166**

Table 3.2: **Quantitative Comparison**: We compare our approach with other deep MPC based methods from [6] and a single-step approach [5]. Apart from MPC+NN [6], all methods attain convergence on the simulation benchmark. We report the time per servoing iteration excluding overheads (Time w.o. flow), the time per servoing iteration including flow computations (Time w. flow), the average number of iterations taken to reach convergence (Total Iters.) and the average time required to servo to the destination, including overheads (Total Time). DeepMPCVS [6] is optimal and requires the least number of iterations, but has a costly per-iteration time overhead. Our method has a very low per MPC-step compute time and takes the least amount of time to attain convergence, comparable to the performance of DFVS [5]. All times are given in seconds. (* means does not converge in some scenes.)

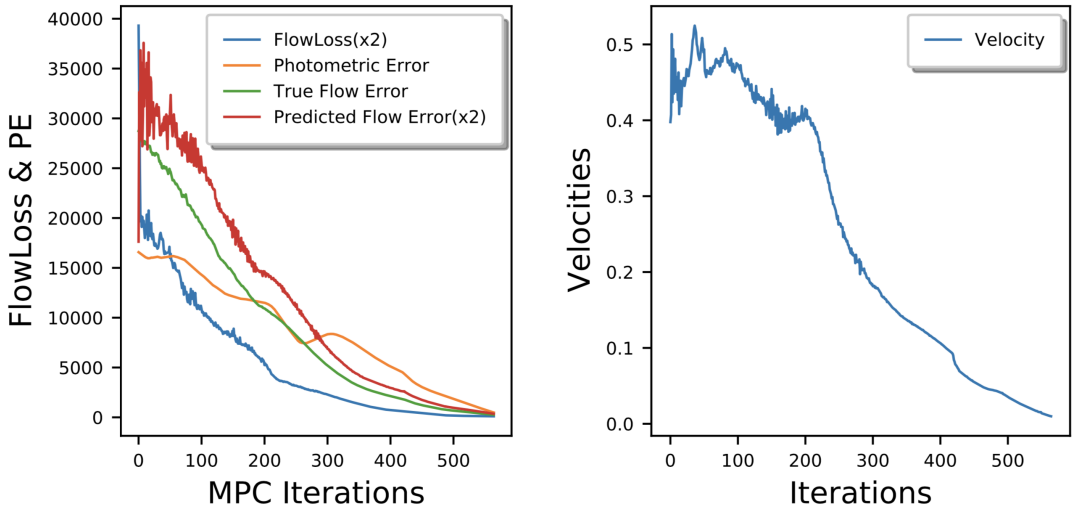

Figure 3.4: **Left**: All the important errors are being reduced concurrently. The drop in FlowLoss indicates that our network can accurately predict flows as the number of MPC iterations increase. The drop in flow errors and photometric errors indicates that the agent is able to reach near the goal. **Right**: The magnitude of velocity reduces as we near the goal, resulting in a smooth and stable convergence.

3.3.2 Qualitative Results

While we achieve a significant improvement in time, our approach does not compromise on photometric convergence. We test out our approach across all environments in our simulation benchmark. Our controller successfully achieves convergence and is able to servo to the desired location. The control actions are learnt unsupervised and the network is trained on-the-fly in an unsupervised fashion. We attain a strict photometric error of strictly $<$500 in all scenes and report the photometric error representation computed between the attained and the desired image, in Fig. 3.5.

3.3.3 Pose Error and Trajectory Lengths

We perform more quantitative tests and report the initial pose error, translation error(T. Error), rotational error(R. Error) and trajectory length(Traj. Length) averaged out over all environments across our simulation benchmark, as depicted in TABLE 3.3. We achieve very low pose errors which is comparable to the other 6DoF servoing approaches. Our approach does not compromise on this and achieves precise alignments with high servoing frequency. We are able to capture a strong correlation between the photometric error and the flow loss, and a steady decrease in the velocities, as depicted in Fig. 3.4(Left).

Our trajectory lengths are slightly inferior to [6], but shorter than single-step approaches such as [5], which signifies the importance of our control optimization steps. We compare the trajectory followed by our agent with other multi-step ahead approaches, as depicted in Fig. 3.6(left). Making a slight

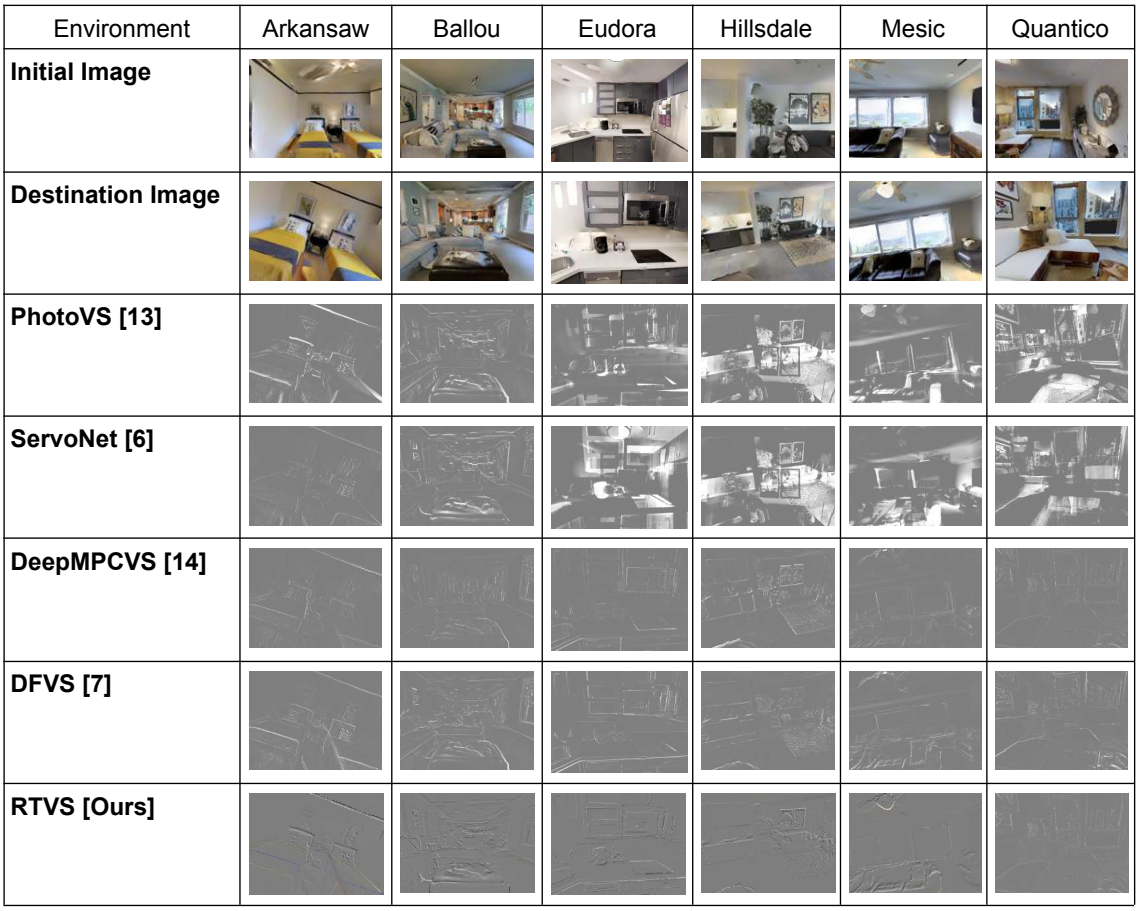

Figure 3.5: **Qualitative Results**: Our controller successfully achieves convergence with a strict photometric error of $<$500 across all environments. Here, we show the photometric error image representation computed between the goal and attained images on termination for six environments in the simulation benchmark. PhotoVS [4] and ServoNet [3] fail to converge even with large number of iterations, thus showing large photometric errors. DFVS [5] successfully converges, but is a single-step approach that does not consider a long term horizon. DeepMPCVS [6] also meets the photometric convergence criteria, but is extremely slow since the online training of its RNN architecture is computationally expensive. Our work achieves strict convergence with control generated in real-time at a frequency of 66 Hz (excluding flow overheads), whilst optimising over a receding horizon.

Approaches	T. Error (m)	R. Error (deg.)	Traj. Length (meters)
Initial Pose Error	1.6566	21.4166	-
MPC+NN* [6]	0.1020	2.6200	**1.1600**
DeepMPCVS [6]	0.1200	**0.5500**	1.1800
RTVS [Ours]	**0.0200**	0.5900	1.732

Table 3.3: **Quantitative Comparison**: We compare the average performance in terms of pose error and trajectory lengths for different approaches across all environments in our benchmark.(* means does not converge in some scenes.)

compromise on the trajectory length, we achieve superior servoing frequency as compared to other time-consuming approaches like [6], thereby achieving the right trade-off between servoing rate and performance.

3.3.4 Generalisation to Real-World Scenarios

We were unable to verify our approach for real-world scenarios due to Covid restrictions at our university and inability to access lab hardware. Previous approaches like [5] which have a similar run-time as our work and were tested on the same simulation environment [41], have shown successful deployment on drones in the real world. We achieve similar a servoing rate as [5] and are confident that our approach can also be deployed to real-life drones. In order to simulate a real world setup, we perform tests with actuation noise.

Approaches	Time Per MPC-step	MPC Steps	Total Time
RTVS [Noiseless]	0.095	1774.28	168.53
RTVS [Induced Noise]	0.095	1850.22	175.77

Table 3.4: **Actuation Noise**: We achieve convergence even in the presence of actuation noise, thereby demonstrating the ability of our controller to adapt to a real world setup. All times are in seconds.

Robot commands are generally noisy in a real-world setup. In order to simulate such conditions, we add a Gaussian noise with μ=0 and σ=0.1 (m/s for translational and rad/s for rotational) to the control commands in Habitat in all six degrees of freedom, before applying it to the agent. We test this out across

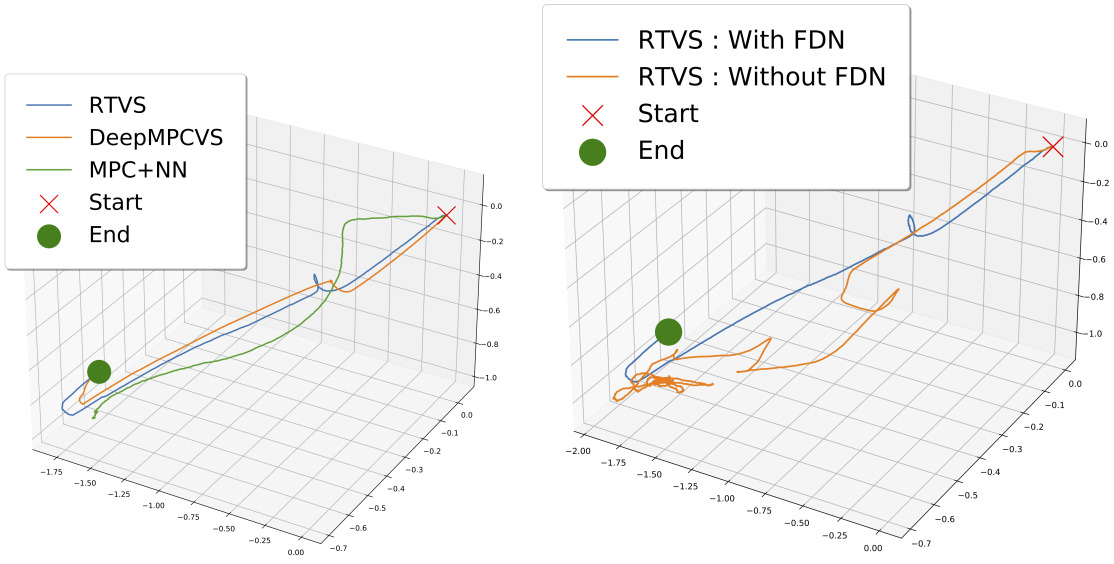

Figure 3.6: **Left:** We plot the trajectories for the various approaches in our benchmark. **Right:** Our lightweight controller generates optimal velocity commands and is able to servo to the desired location in the presence of FDN (flow depth normalisation), which effectively handles inaccuracies in the flow predictions. Grid size in the plots is (0.2m X 0.2m X 0.2m).

all scenes in our benchmark and successfully achieve convergence to a photometric error of $<$500. Our method adapts well and converges in an average number of 1850.21 MPC steps, which is 4.28% more than those required in a noiseless setup as depicted in TABLE 3.4. Our approach is thus capable of handling actuation noise and generalising in real-world experiments.

3.3.5 Evaluating Flow Depth Normalisation

Since we use FlowNetC [17] without any retraining/finetuning, using a lightweight architecture is vulnerable to inaccuracies from the flow network. To counter this, we have proposed the flow depth normalisation layer. As depicted in Fig. 3.6(Right), our architecture is unable to achieve convergence and reach its destination when the flow depth is not normalised. With this layer in place, the flow is stabilised and we achieve robust performance in the servoing rate as well as accuracy.

Chapter 4

Learning Arc-Length Value Function for Fast Time-Optimal Rearrangement Sequence Planning and Execution

4.1 INTRODUCTION

A fundamental challenge in computing an optimal visitation sequence for pick and place is estimating the cost-to-go between two manipulator end-effector positions. A useful cost-to-go metric can be the optimal (or shortest) joint space arc length between two positions. However, this cost-to-go is not known analytically and in fact, requires one to solve a non-convex optimization problem. To see why this is challenging, imagine that the manipulator end-effector needs to visit 4 positions and thus there are $4!$ sequences possible. To estimate the cost of all the sequences, one needs to solve the same number of optimization problems. This is obviously intractable beyond sequences for visiting a small number of objects. Existing works [11] bypass this intractability by using the simple Euclidean distance as the cost-to-go between two positions. However, we show in the present work that such approaches can lead to sub-optimal performance. To be precise, the shortest Cartesian/Euclidean space trajectory that connects two end effector positions can have an unduly long arc length in the joint space. Vice-versa, the shortest arc-length trajectory in the joint space need not be a straight line in the end-effector space.

Beyond the **Sequencer**, a successful pick and place also need an efficient **Executor** that can compute the joint velocities (or simply joint angles) to realize the optimal visitation sequence. Typically, this is done by invoking one of the many Inverse Kinematics (IK) algorithms, although more sophisticated choices such as optimal control planners are possible. However, existing IK algorithms, either implicitly or explicitly use the Euclidean distance heuristic to move between two given positions. Thus they are inherently sub-optimal with respect to minimizing joint arc length and consequently pick and place completion time.

4.1.1 Layout of the Chapter and Mathematical Notations

The rest of the Chapter is divided into three parts. Section 4.2.1 presents our value function learning approach. The results of this section are then utilized in 4.3 to develop a **Sequencer** and **Executor** for

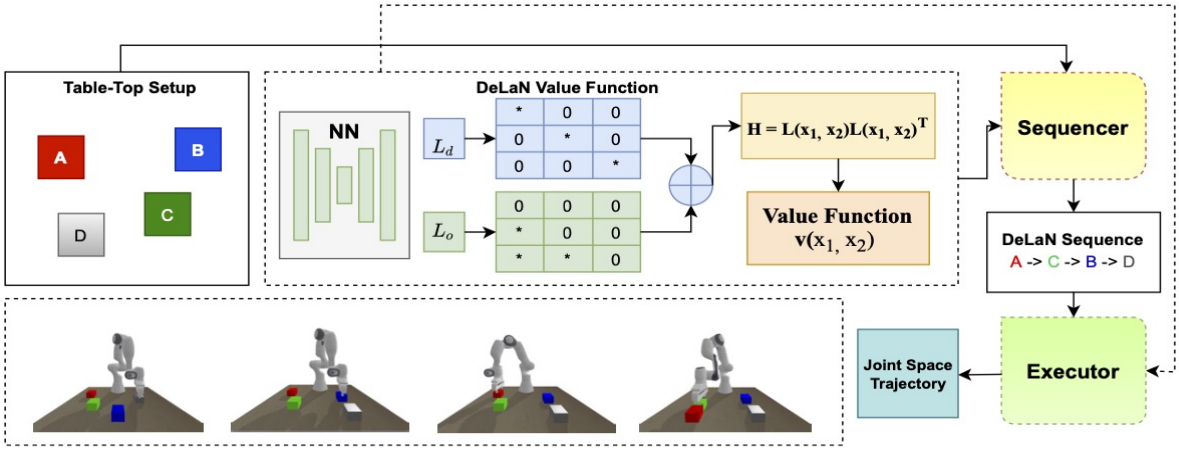

Figure 4.1: Pipeline for optimal pick and place sequence planning and execution. The **Sequencer** takes in the object locations and the value function estimates from a learned neural network. The computed sequence is then passed onto the **Executor** that again uses the value function estimates to incrementally plan trajectories between two end-effector positions. As shown, our neural network has a special form and has been adapted from [7]. We construct two matrices $\mathbf{L}_d$, $\mathbf{L}_o$ whose elements are feedforward neural networks. The matrices are summed to obtain $\mathbf{L}$ and the value function is represented as $\mathbf{L}\mathbf{L}^T$

time-optimal pick and place motion planning. The validation of our algorithmic results is presented in Section 4.4.

In the rest of the paper, we will use normal font lower case letters to represent scalars, while bold font variants will represent vectors. Matrices will be represented through bold face upper-case letters. The superscript T will be used to represent transpose of a vector or a matrix.

4.2 Value Function Learning

4.2.1 Trajectory Optimization

Given a start $\mathbf{x}_i$ and a goal $\mathbf{x}_j$ position, we can compute multi-step joint trajectory with the optimal (shortest) arc-length through the following non-convex optimization problem.

$$\mathbf{q}^*_{t_i:t_j} = \arg\min_{\mathbf{q}_{t_i}} \sum_{t_i}^{t_j} \|\mathbf{q}_{t_{i+1}} - \mathbf{q}_{t_i}\|_2^2 \tag{4.1a}$$

$$g_{fk}(\mathbf{q}_{t_i}) = \mathbf{x}_i, \qquad g_{fk}(\mathbf{q}_{t_j}) = \mathbf{x}_j \tag{4.1b}$$

$$\dot{\mathbf{q}}_{min}\Delta t \leq \mathbf{q}_{t_{i+1}} - \mathbf{q}_{t_i} \leq \dot{\mathbf{q}}_{max}\Delta t, \forall t_i \in [t_i \;\; t_j] \tag{4.1c}$$

Where, $\mathbf{q}_{t_i}$ refers to the joint position at time t_i. The cost function (4.1a) optimizes the arc length in the joint space by minimizing the increment between two subsequent joint positions. The equality constraints (4.1b) ensures that the joint trajectory connects the start and goal positions of the end-effector. The function g_{fk} represents the forward kinematics mapping of the manipulator. The inequality constraint enforces the velocity limit on the optimal trajectory. We represent the optimal solution of the trajectory optimization as $\mathbf{q}^*_{t_i:t_j}$. Note that the subscript implies that joint trajectory is defined in some time interval $[t_i \;\; t_j]$.

Definition 1 *We define the optimal arc-length value function as* $v(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{t_i}^{t_j} \|\boldsymbol{q}^*_{t_{i+1}} - \boldsymbol{q}^*_{t_i}\|_2^2$ *that captures the shortest joint arc-length possible while moving the end-effector from* $\boldsymbol{x}_i$ *to* $\boldsymbol{x}_j$ *in time interval* $[t_i \;\; t_j]$.

Remark 1 *Redundancy resolution is already incorporated in optimization* (4.1a)-(4.1b) *as among many solutions, it automatically outputs ones that lead to the lowest arc-length trajectory in the joint space.*

Remark 2 *The space of self-motion, i.e joint angles that do not move the end-effector is automatically avoided in optimization* (4.1a)-(4.1b).

4.2.2 Learning Arc-length Value Function

As clear from the discussion in the last sub-section, obtaining the optimal arc-length value function requires us to solve an optimization problem. However, this presents a critical bottleneck for pick and place sequence planning wherein many combinations of start and goal positions need to be searched before arriving at the optimal sequence. Solving a trajectory optimization for each possible sequence will be computationally prohibitive. Thus, in this subsection, we present a data-driven alternative. We hypothesize that the arc-length value function can be parameterized in the following form for some positive semi-definite matrix $\mathbf{H}$.

$$v(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{H}(\mathbf{x}_i, \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j) \tag{4.2}$$

Our choice of representation for the value function is motivated by works on metric learning [34], [36], wherein the functions like the r.h.s of (4.2) are used to quantify the distance between two features. However, works on metric learning typically assume a constant $\mathbf{H}$ matrix which is, in turn, learnt from the data. In sharp contrast, we model $\mathbf{H}$ as a neural network. This substantially increases the complexity

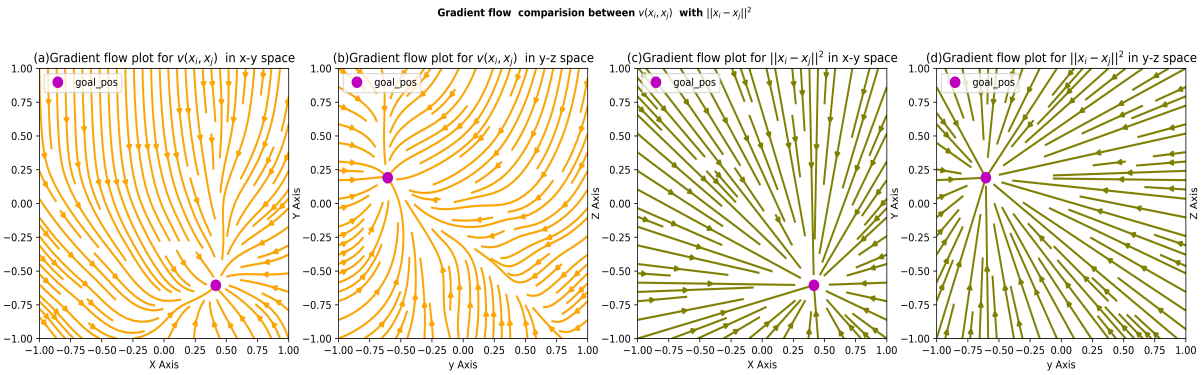

Figure 4.2: Gradient flow of learned value function $v(\mathbf{x}_i, \mathbf{x}_j)$ and Euclidean heuristic. The non-linear gradient flow lines of $v(\mathbf{x}_i, \mathbf{x}_j)$ capture the premise that the manipulator end-effector motion lies in a Riemannian Manifold. In contrast, the gradient flow of Euclidean distance is trivial straight lines, all converging to the goal.

of our learning problem, especially because we need to impose the structure of positive definiteness in our learned network. We take inspiration from the network architecture proposed in [7]. The authors in this cited work derive a network topology called Deep Lagrangian Network (DeLaN) encoding the Euler-Lagrange equation originating from Lagrangian Mechanics. We use a simplified version of the network proposed in [7] which is presented in Fig. 4.1 to ensure the positive definiteness of $\mathbf{H}$ by construction. As shown, we model $\mathbf{H}$ as the following outer product.

$$\mathbf{H}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{L}(\mathbf{x}_i, \mathbf{x}_j)\mathbf{L}(\mathbf{x}_i, \mathbf{x}_j)^T \tag{4.3}$$

The matrix $\mathbf{L}(\mathbf{x}_i, \mathbf{x}_j)$ is obtained by the summation of a diagonal matrix $\mathbf{L}_d$ and a lower diagonal matrix $\mathbf{L}_o$. Each non-zero element of these matrices are feed-forward neural networks with learnable parameters. The elements of $\mathbf{L}_o$ are modeled as neural networks with linear activation function. In contrast, the diagonal elements of $\mathbf{L}_d$ are constructed as neural-networks with Relu activation function at every layer. This by-construction, ensures that the diagonal elements of $\mathbf{L}_d$ are always positive and consequently leads to guaranteed positive semi-definiteness in $\mathbf{H}$.

4.2.3 Network Training

We train the value function network (DeLaN) described in the previous section in a supervised manner by regressing it against optimal ground truth arc-length computed off-line through trajectory optimiza-

tion. The input to our network is the pair of positions $\mathbf{x}_i, \mathbf{x}_j$ between which we need to estimate the value function. The output of the network is the value function $v(\mathbf{x}_i, \mathbf{x}_j)$.

To prepare the dataset, we sample multiple start and goal positions from a table task space. The x, y and z coordinates of these points are sampled uniformly and vary between $[-0.8, 0.8]$, $[-0.3, 0.3]$ and $[0.2, 0.8]$ respectively. We use Scipy-SLSQP [42] to solve optimization (4.1a)-(4.1b) which generates the joint trajectory for sampled start-goal pairs. The associated arc-length traversed by the manipulator is considered to be the ground-truth value function. However, it should be noted that our approach is agnostic to the choice of the trajectory optimizer or even the manipulator. We sample around 13000 pairs of training points from the confined task space and regress the network with the ground truth arc length. The DeLaN networks have 2 hidden layers each consisting of 64 neurons. We train the network for 5000 epochs, with Adam optimizer [43] and a learning rate of 0.0001 and a batch size of 256. The testing data consists of randomly drawn samples from the task space. Since the task space of a manipulator is bounded, the DeLaN network generalized well on the testing data.

4.2.4 Visualizing Value Function

Fig.4.2 shows the gradient flow of our learned value function $v(\mathbf{x}_i, \mathbf{x}_j)$ for Franka Panda arm and compares it with that of the Euclidean distance metric. Unsurprisingly, the gradient flow of the Euclidean distance metric consists of straight lines all converging to the goal. In contrast, the gradient flow of our $v(\mathbf{x}_i, \mathbf{x}_j)$ exhibits the expected non-linearity stemming from the fact that the end-effector motion of a manipulator lies in a Riemannian Manifold. The nonlinearity of the gradient flow also further reiterates why the straight-line end-effector trajectory connecting two positions does not lead to shorter trajectories in the joint space.

4.3 Application to Pick and Place Sequence Planning and Execution

In this section, we used the $v(\mathbf{x}_i, \mathbf{x}_j)$ to compute the optimal pick and place sequence (we call this DeLaN **Sequencer**) and at the same time develop motion plans for executing the computed sequence (we call this the DeLaN Metric IK **Executor**). Our overall pipeline can be summarised using Fig. 4.1.

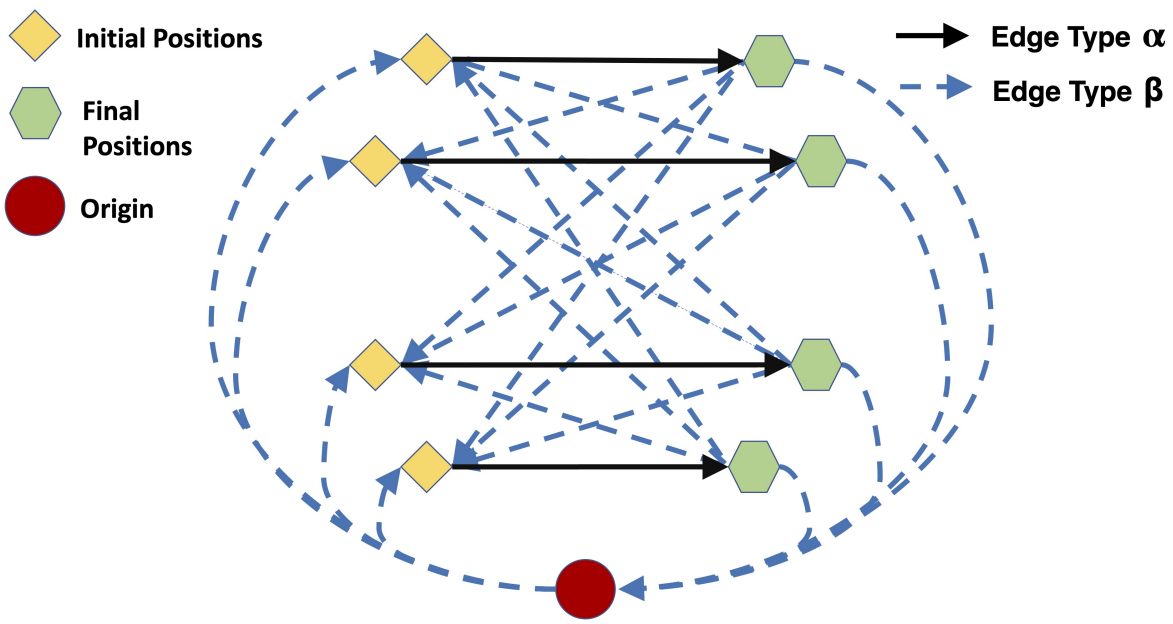

Figure 4.3: The above shown figure depicts the graph formulation explained in 4.3.1 for 4 objects. The edges-α is from the object's initial position to its respective goal locations and the edges-β join the object u_i's goal to start locations of other objects u_j ($\forall\ j\ != i$). The manipulator, starting at Origin *O* picks up the fist object from $\mathcal{U}$ & returns back from last objects goal location.

4.3.1 Sequencer as Capacitated Vehicle Routing Problem (CVRP) with edge-weight $v(\mathbf{x}_i, \mathbf{x}_j)$

Consider a task-space $\mathcal{T}$ containing k number of movable objects $\mathcal{U} = \{u_1, u_2, ..., u_k\}$ and a fixed manipulator $\mathcal{M}$. We transform this task space into graph representation *G = (V, E, A)* where the object's current and goal positions are the vertices *V*. The edges *E* (edge-α's) are directed from the object's current (yellow diamonds in Fig.4.3) to their respective goal positions (green hexagons in Fig.4.3). The edges *A* (edge-β's) are also directed but joins the object u_i's goal to start locations of any other objects u_j ($\forall\ j\ != i$). We further add a point of origin *O* (red circle) to this graph *G* where the end-effector of the manipulator($\mathcal{M}$) starts and returns back after performing the pick and place operations.

The edge weights (*E*,*A*) of the graph *G* are taken from a adjacency matrix $\mathbf{W}$ where the element w_{ij} represent the edge cost between two locations i & j. Fig. 4.3 represents the defined graph. The manipulator initially is at point *O* and explores its options to pick up the first object, denoted by edge-β's in Fig. 4.3. Once the object u_i is picked up from its initial position it needs to be delivered to its goal position, this is denoted by edge-α's. Once the manipulator has reached the goal position of an object

u_i, it will now go back to the origin O to start exploring its options to go at a start position of any other object u_j. At the end once it has placed the last object it comes back to the point Origin O.

Our graph set-up is exactly like the CVRP with capacity equal to one since only one object can be picked up at a time. We use Google OR-TOOLS[44] to find an optimal pick-place sequence. We use the learned $v(\mathbf{x}_i, \mathbf{x}_j)$ as the edge-cost of our CVRP graph, i.e $w_{ij} = v(\mathbf{x}_i, \mathbf{x}_j)$. We reiterate this is the first of the core novelties of our work as many existing works model w_{ij} as the Euclidean distance between the start and a goal position.

The next subsection presents our second novelty that leverages the learned value function to improve the IK algorithm of the manipulator itself to further reduce pick and place execution time.

4.3.2 Proposed Executor: DeLaN Metric IK

Suppose the manipulator is currently at position $\mathbf{x}$ with joint $\mathbf{q}_t$ and is required to move to $\mathbf{x}_f$. The manipulator motion from $\mathbf{x}$ to $\mathbf{x}_f$ can be planned in an incremental manner through the classical differential Inverse Kinematics (4.4-4.7), wherein $\mathbf{J}(\mathbf{q}_t)$ represents the manipulator Jacobian evaluated at $\mathbf{q}_t$, Δt represents time interval and k is some scalar constant.

$$\min_{\dot{\mathbf{q}}_t} \|\mathbf{J}(\mathbf{q}_t)\dot{\mathbf{q}}_t - \dot{\mathbf{x}}_d\|_2^2 + \|\dot{\mathbf{q}}_t\|_2^2 \tag{4.4}$$

$$\mathbf{q}_{min} \leq \mathbf{q}_{t-1} + \dot{\mathbf{q}}_t \Delta t \leq \mathbf{q}_{max} \tag{4.5}$$

$$\dot{\mathbf{q}}_{min} \leq \dot{\mathbf{q}}_t \leq \dot{\mathbf{q}}_{max} \tag{4.6}$$

$$\dot{\mathbf{x}}_d = k(\mathbf{x} - \mathbf{x}_f) \tag{4.7}$$

The cost function (4.4) brings the end-effector velocity close to some desired velocity given by (4.7). The constraints (4.5)-(4.6) enforces the bounds of joint angles and velocities. It is also possible to include acceleration bounds approximated through finite difference of velocities and final orientation constraints. We have not included it here to keep the notation and exposition simple. But our final implementation do include orientation constraints.

The desired velocity (4.7) is in fact the scaled gradient of the Euclidean distance $\frac{1}{2}\|\mathbf{x} - \mathbf{x}_f\|_2^2$. Thus, it is clear that the classical differential IK uses the Euclidean distance as the heuristic for the cost-to-go between $\mathbf{x}, \mathbf{x}_f$. However, as discussed in section (4.1), the Euclidean heuristic can lead to a sub-optimal performance in terms of minimizing the arc length in the joint space

The main idea behind our proposed IK is to make the end-effector move incrementally along the true cost-to-go or in other words the value function $v(\mathbf{x}, \mathbf{x}_j)$ (recall eqn no 4.2). To this end, we propose the following optimization problem as a substitute for the classical IK

$$\min_{\dot{\mathbf{q}}_t, \mathbf{x}_{next}} \frac{1}{2}(\mathbf{x}_{next} - \mathbf{x}_f)^T \mathbf{H}(\mathbf{x}_{next}, \mathbf{x}_f)(\mathbf{x}_{next} - \mathbf{x}_f) + \|\dot{\mathbf{q}}_t\|_2^2 \tag{4.8a}$$

$$g_{fk}(\dot{\mathbf{q}}_t \Delta t + \mathbf{q}_{t-1}) = \mathbf{x}_{next} \tag{4.8b}$$

$$\mathbf{q}_{min} \leq \dot{\mathbf{q}}_t \Delta t + \mathbf{q}_{t-1} \leq \mathbf{q}_{max} \tag{4.8c}$$

$$\dot{\mathbf{q}}_{min} \leq \dot{\mathbf{q}}_t \leq \dot{\mathbf{q}}_{max} \tag{4.8d}$$

The cost function seeks to compute the next possible state $\mathbf{x}_{next}$ that the manipulator with a joint velocity $\dot{\mathbf{q}}_t$ will reach in time duration Δt. Importantly, $\mathbf{x}_{next}$ is sought to be as close as possible to $\mathbf{x}_f$ in the metric space of $\mathbf{H}$ (eqn 4.3), which we recall captures the value function or true cost-to-go between two given end-effector position (eqn. 4.2). The equality constraints (4.8b) stems from the forward kinematics while inequalities (4.8c)-(4.8d) are the standard bounds on joint limits and velocities.

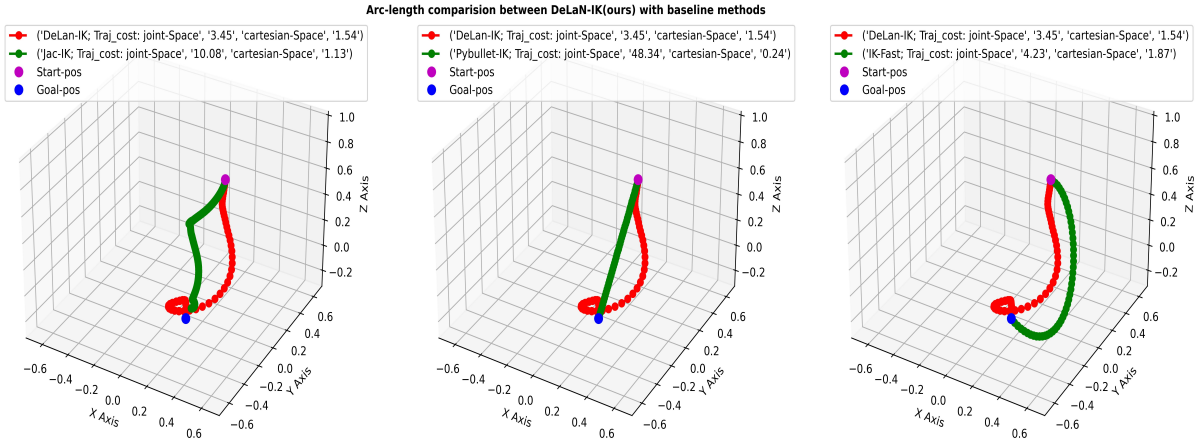

Figure 4.4: Comparison between our DeLaN Metric IK and baselines approaches for Franka Panda manipulator. Our approach uses the learned value function to move the end-effector along a path that requires smaller joint motions. In contrast, the baselines aim to just minimize the arc-length in the end-effector space which requires excessive joint motions.

Given a learned $\mathbf{H}$ (i.e weights of the neural-network fixed), (4.8a)-(4.8d) involves optimization over the input space of neural network. Furthermore, the functional form of $\mathbf{H}$ over $\mathbf{x}_{next}$ can be highly non-smooth and non-convex and thus solving (4.8a)-(4.8d) could be extremely challenging. Thus, we now present our final simplification that leverages the positive definite characteristic of $\mathbf{H}$. We replace

the $\mathbf{x}_{next}$ in the argument of $\mathbf{H}$ with the current $\mathbf{x}$. This in turn makes $\mathbf{H}$ a constant and convexifies the cost function turning (4.8a)-(4.8d) to a smooth non-linear programming problem. The proposed novel metric aware trajectory optimization formulation (4.8a)-(4.8d) is henceforth called DeLaN Metric IK Executor.

4.4 Validation and Benchmarking

The objective of this section is two-fold.

- First, we benchmark our **Executor**: value-function-based DeLan metric IK against existing approaches for task-space control. Note that we do not consider the trajectory optimization approaches that computes the full motion plan connecting start to goal positions but are computationally expensive. Rather, we focus on one-step reactive planners that provides real-time adaptability.
- Second, we perform an empirical analysis on the performance obtained by different combinations of our **Sequencer** and **Executor** (IK) over different baselines.

Our neural network training was done in PyTorch with ADAM [43] optimizer. Our **Executor** was implemented in both Python and C++. The latter was done to evaluate the best-case computational performance.

4.4.1 Bench-marking our Executor

We compare our DeLaN Metric IK against the following baselines that differ in their methodology for computing the joint motion needed to drive the end-effector to the specified goal position. We use the joint space arc-length of the resulting trajectory as our metric. We performed two sets of comparisons on Franka Panda and UR5e manipulator.

- IK-Fast with Linear Interpolation in joint-space (IK-Fast): In this baseline, we compute multiple joint angle vectors for a given goal position using the IK-Fast algorithm [45] computes closed-form symbolic expressions for IK. We then compute the norm of the difference of each of the computed solutions to the current joint position and choose the one with the least norm. Subsequently, we perform a linear interpolation between the current joint position and the chosen solution to get the joint space trajectory.

- Jacobian-based IK (Jac-IK): This is the most popular method for controlling end-effector motion. Its mathematical formulation is described in (4.4)-4.6 and we implemented it in Python using robotics toolbox [46] for the Jacobian Computation.

- Py-Bullet IK with Linear Cartesian Interpolation (PyBullet-IK): In this baseline, we incrementally build a short linear end-effector trajectory from the current position towards the goal and then use the IK-Fast algorithm to compute the joint angles necessary to reach the end of that trajectory. This process is repeated till the end-effector reaches the goal position.

Some of the qualitative results obtained for the Franka Panda manipulator are summarized in Fig.4.4. Unsurprisingly both Jac-IK and Pybullet-IK baselines result in shorter Cartesian space trajectories than our DeLaN Metric IK but fare poorly in terms of arc length in the joint space. In this specific example, the IK-Fast approach leads to poor performance in both Cartesian and joint space. We reiterate that joint space motion is what we care about since it directly affects the task completion time. Thus, our DeLaN Metric IK will ensure the quickest motion between two given positions if all the IK approaches are allowed the same velocity and acceleration bounds. We further validate this in the supplementary video (`https://tinyurl.com/4u37x33r`), where we perform an additional time-optimal reparameterization [47] of the joint space trajectories obtained with all the baselines and our DeLaN Metric IK. Our approach retained the quickest task completion time even under such a transformation.

Fig.4.5-4.6 provides further validation of the qualitative insights obtained above. We sampled 1000 random start and goal end-effector positions to evaluate all the baselines and our DeLaN Metric IK. For both Franka Panda (Fig.4.5) and UR5e manipulator (Fig.4.6), our approach consistently outperformed all the baselines in the joint space arc-length metric.

4.4.2 Benchmarking Pick and Place Arc-Lengths/Completion Time

We constructed different baselines by using alternate combinations of **Sequencer** and **Executor** to benchmark total task completion time or its surrogate joint space arc-length for pick and place operation. For **Executor**, we used RoboTSP [11], Jac-IK as implemented in ROS-KDL[9], and PyBullet-IK [10]. The latter two were already detailed in the previous section. Thus, here we just give a brief overview of RoboTSP. It operates by first computing multiple IK solutions at each node position of the pick and place sequence. It then uses Dijkstra algorithm to compute the shortest joint space trajec-

Sequencer	Executor	NUMBER OF OBJECTS				
		4	**6**	**10**	**20**	**50**
Euclidean	ROS-KDL[9]	17.78	16.59	29.97	56.92	123.33
Euclidean	RoboTSP[11]	18.65	24.79	40.59	78.05	178.22
Euclidean	PyBullet IK[10]	51.37	73.23	128.65	231.40	567.21
Euclidean	DeLaN Metric IK (**ours**)	11.41	15.71	23.10	38.70	90.88
Multi Layer Perceptron	ROS-KDL[9]	17.52	16.59	29.33	55.81	126.15
Multi Layer Perceptron	RoboTSP[11]	18.48	25.72	40.80	79.63	177.80
Multi Layer Perceptron	PyBullet IK[10]	51.37	73.23	128.65	231.40	567.77
Multi Layer Perceptron	DeLaN Metric IK (**ours**)	11.41	15.71	**23.10***	38.70	90.88
DeLaN (**ours**)	ROS-KDL[9]	16.56	16.30	27.20	55.80	120.47
DeLaN (**ours**)	RoboTSP[11]	17.85	22.79	40.14	76.82	176.45
DeLaN (**ours**)	PyBullet IK[10]	53.01	70.63	115.12	225.11	442.67
DeLaN (**ours**)	DeLaN Metric IK (**ours**)	**10.22***	**11.49***	24.72	**34.07***	**84.45***

Table 4.1: Comparison of Mean arc-length in joint space (over 10 runs) for different combination of **Sequencers** and **Executors** proposed in section 4.4.2 for Franka Panda arm. All the planners are successfully able to execute the sequence. DeLaN Sequencer (on average) shows improvement in arc-length even when combined with ROS-KDL[9] , RoboTSP[11] and PyBullet[10] IK Planners planner. Similarly, DeLaN Metric IK planner is able to significantly reduce the arc-length even when combined with Euclidean Sequencer and Standard MLP Sequencer. Our proposed solution combining DeLaN Sequencer and DeLaN Metric IK based planner is able to achieve least arc-length cost. The start and goal positions for each object is randomly sampled in the table top task space. The difference becomes more pronounced as the number of objects increase.

tory connecting the IK solutions. In free-space, RoboTSP will behave very similar to IK-Fast baseline discussed in the previous section.

The following different **Sequencer** were combined with each of these **Executor**s

- Euclidean: This **Sequencer** uses the Euclidean distance between two end-effector positions as the edge-weight $w(\mathbf{x}_i, \mathbf{x}_j)$ in the CVRP algorithm for sequence planning.
- Multi-Layer Peceptron (MLP): This sequencer is conceptually similar to ours and is based on estimating the value function between two manipulator end-effector positions. The core difference stems from the underlying neural-network architecture. This **Sequencer** uses a network with an MLP architecture while ours proposed used the specialized DeLaN network (recall Fig.4.1).

The quantitative results summarized in Table 4.1 present a clear pattern. For any given **Executor**, the shortest arc-length in joint space is obtained using the DeLaN **Sequencer**, followed by the MLP

Executor	NUMBER OF OBJECTS				
	4	6	10	20	50
DeLaN Metric IK	**3.92***	**4.40***	**9.47***	**13.05***	**34.73***
ROS-KDL[9]	6.34	6.24	10.42	21.38	46.16
RoboTSP	6.84	8.73	15.38	29.43	67.43
PyBullet IK	20.30	27.06	44.11	86.25	169.60

Table 4.2: **Trajectory Execution Time(in sec.):** We compare the average trajectory execution time for our DeLaN Metric IK with different approaches.

Network	NUMBER OF OBJECTS				
	4	6	10	20	50
DeLaN Infrencing Time	0.0024	0.0024	0.0024	0.0024	0.0024
Google OR[44] Time	0.0049	0.0107	0.0378	0.0685	0.286

Table 4.3: **Sequence Planning Time(in sec.) :** The above table shows the time taken by the network for arc-length prediction and the optimization time taken by solver to find the rearrangement sequence.

and Euclidean baselines. For any given **Sequencer**, the most optimal **Executor** is our DeLaN Metric IK, followed by ROS-KDL [9], RoboTSP [11] and PyBullet IK [10]. Hence we get the least arc-length cost when we combine DeLaN **Sequencer** with DeLaN Metric IK. Fig.4.7 further validates our claim. It shows the mean and median of the joint space arc-length cost obtained from fixed pick and place sequence with different **Executors** approaches. It can be seen that the worst performance of our DeLaN Metric IK is better than the best of all the other baselines.

We performed more quantitative tests to show the effectiveness of our approach. We report the trajectory execution time generated by different **Executors** when all of them use our DeLan **Sequencer** for pick and place of (4, 6, 10, 20 and 50) objects. We are able to achieve the least trajectory execution time as evident in TABLE 4.2. Table 4.3 shows the inference time of DeLaN Network to estimate the edge-weights of the pick and place graph and the sequence planning time achieved by open-source Google OR-TOOLS[44].

4.4.3 Comparison with Multi Layer Perceptron

The unique structure of DeLaN network (recall Fig.4.1) is at the core of the efficiency of our **Sequencer** and **Executor**. To further validate this, we trained a 3 layer MLP to directly predict the optimal (shortest)

Network	Train Error	Test Error
Multi Layer Perceptron	0.168	0.175
DeLaN	0.0049	0.0107

Table 4.4: The above table shows the ability of DeLaN[7] to better learn the arc-length cost between any given start and goal points. This is due to its ability to capture system dynamics by inducing a semi-positive definite structure into the network architecture.

joint space arc-length between two end-effector positions. In contrast, we recall that our DeLaN network predicts a positive definite matrix **H** that scales the Euclidean distance (recall (4.2)).

Since the MLP needs to predict a positive entity we used leaky Relu and Softplus activation in the network to make the final prediction positive. The results are summarized in Table 4.4. The MLP architecture leads to both higher training and testing error as compared to DeLaN architecture. The difference in learning can be further mapped to resulting joint space arc-lengths in Table 4.1.

4.4.4 Visualization in the Latent Space

It is difficult to visualize, the optimality of joint space trajectories generated by different **Executor**s (IK). To get a better sense of joint space arc lengths, we used the popular T-SNE [8] algorithm to project joint space trajectories generated by different **Executor**s to a 2D space. The results are summarized in Fig. 4.8. Our intuition here is that joint space trajectories that have larger arc-length will have more spread-out (high variance) clusters in the 2D space. We can see that the projection of joint space trajectories generated by our DeLaN Metric IK results in clusters with smaller spread than the joint space trajectories of ROS-KDL [9] and Pybullet IK [10]. These figures validate our hypothesis and also the quantitative results shown earlier in Table 4.1.

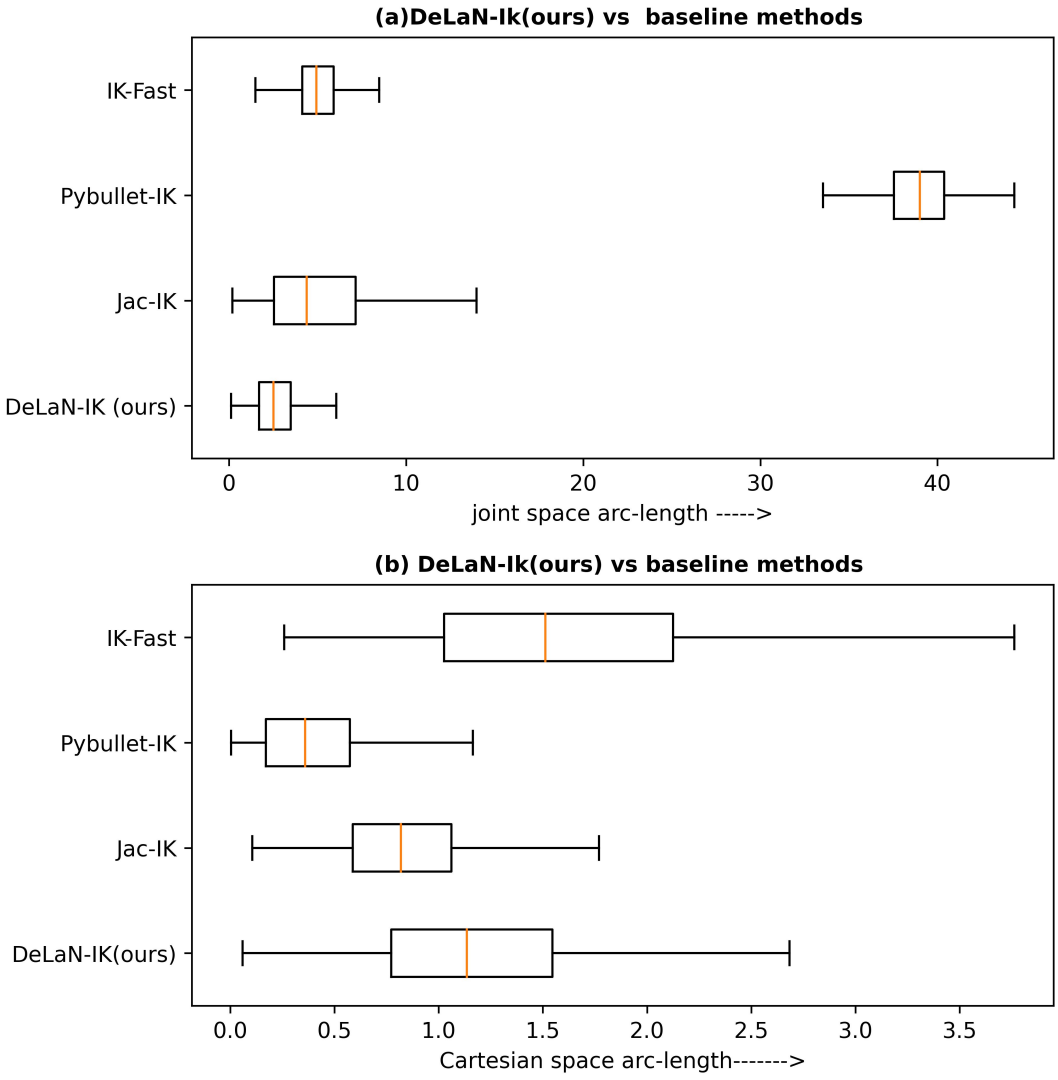

Figure 4.5: Statistical comparison between our DeLaN Metric IK and baseline approaches over 1000 randomly sampled start and goal positions for Franka Panda manipulator. Our approach leverages a learned value function that captures the true cost-to-go between two positions to produce shortest joint space trajectories.

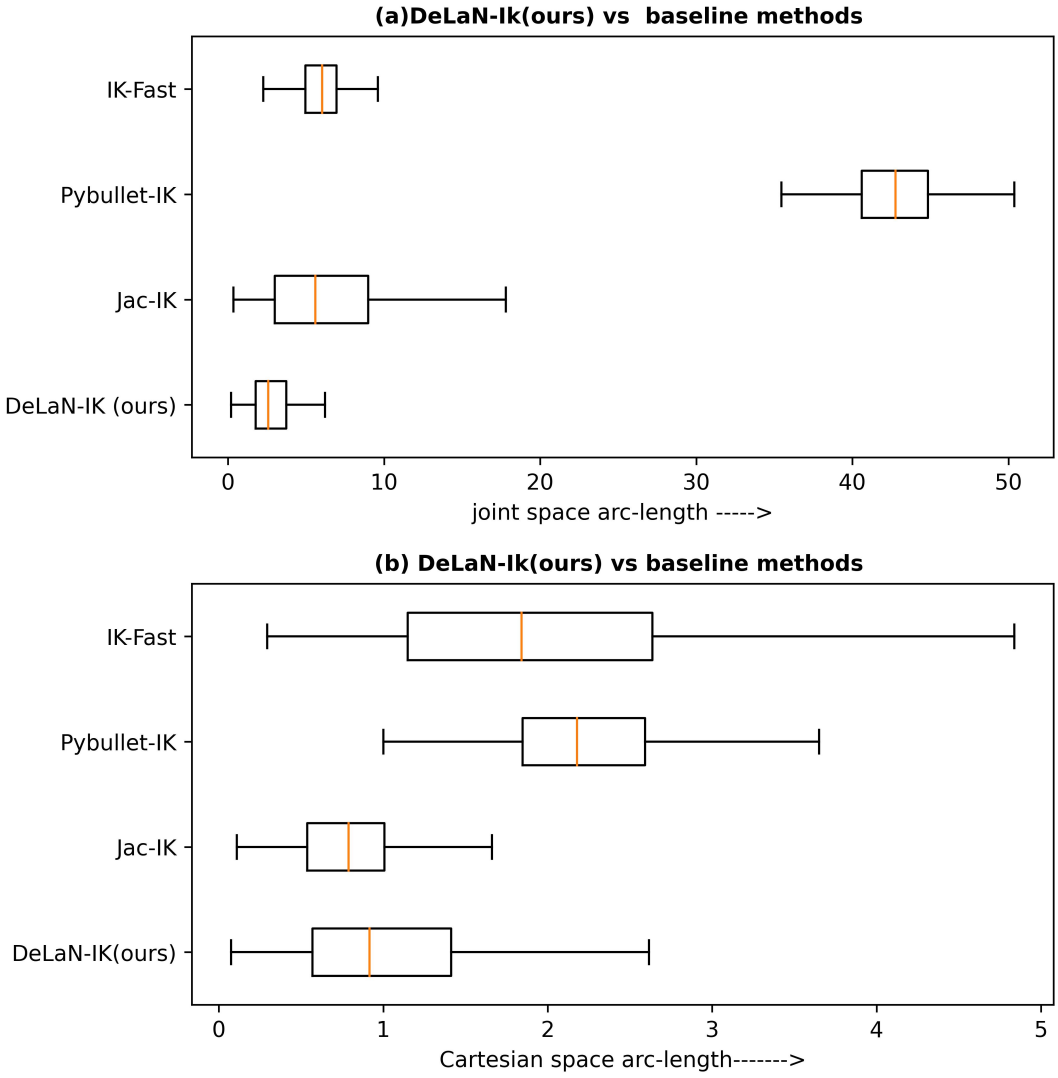

Figure 4.6: A similar result as Fig.4.5 but re-created for a UR5e manipulator. The superiority of DeLaN Metric IK in producing shorter joint motions is retained here.

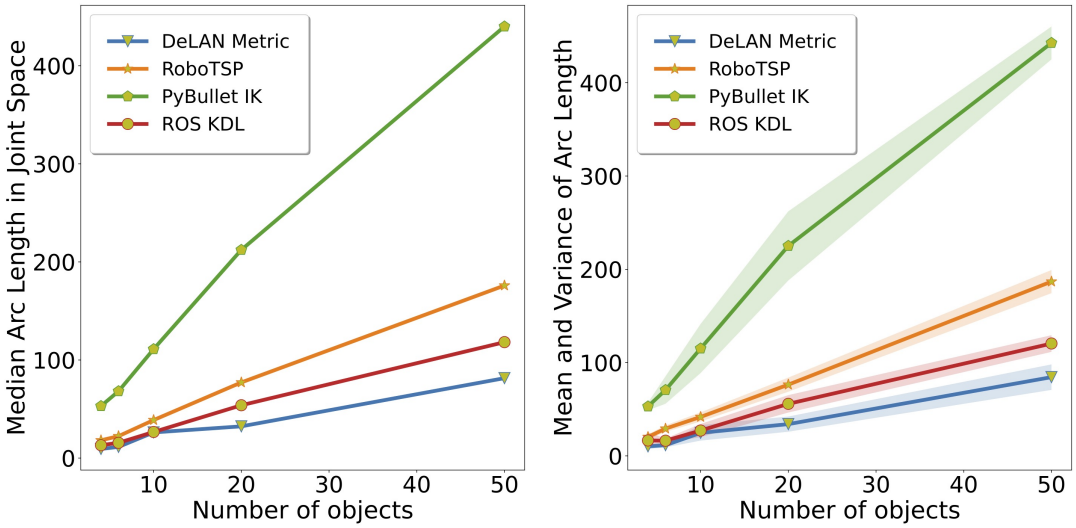

Figure 4.7: **Left**: Median arc length cost for different **Executor** keeping the **Sequencer** fixed. **Right**: For large number of objects(>20), worst arc-length obtained by DeLaN Metric IK (ours) is better than best arc-length obtained by all other baselines.

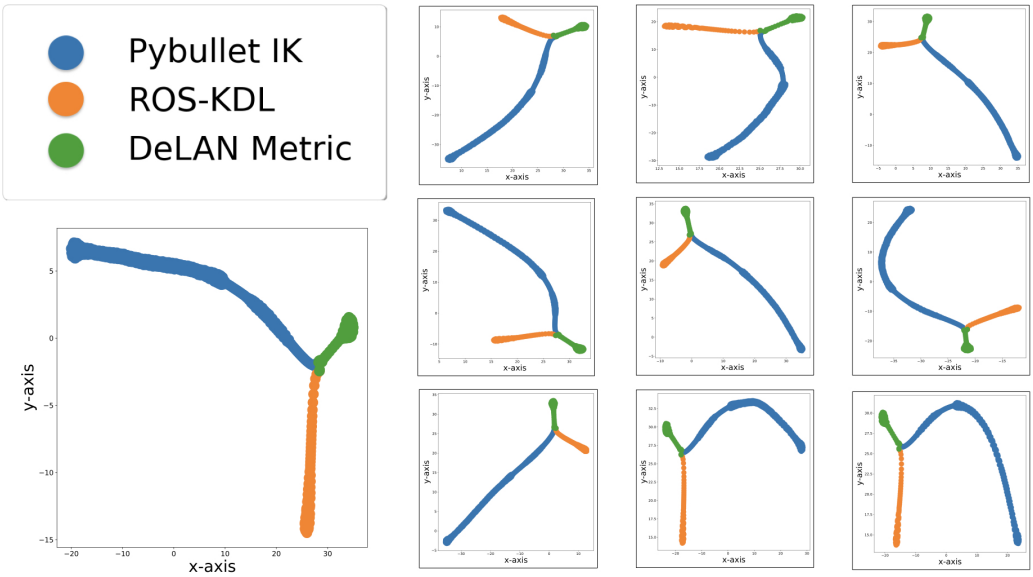

Figure 4.8: **T-SNE[8] projection of joint space trajectories**: Here we project joint space trajectories obtained from different IK (Executors) to a 2D space. Our DeLaN Metric IK is able to incorporate the cost-to-go metric and generates trajectory with much smaller arc-length in joint space. Hence, when we project its trajectories into 2D, we get clusters with small variance(green). When we project trajectories obtained by ROS-KDL [9] and PyBullet IK [10], we get much larger cluster sizes, indicating a larger arc-lengths in joint space.

Chapter 5

Conclusions

In this thesis, we explored two fundamental robotic control problems using data driven approaches. In Chapter3, we significantly improved the MPC algorithm in [6] by putting forth a lightweight and fast model predictive control framework that generates control near real-time at a frequency of 10.52 Hz. We have demonstrated a significant improvement in the total servoing time as compared to other visual servoing approaches in 6DoF. We showcase the efficiency of our control generation architecture, which uses a slim neural network architecture and an effective sampling strategy to generate optimal control in real-time, without making a heavy compromise on its performance. Our controller's ability to train in an online fashion helps it generalise and adapt well to novel environments. Our work is, to the best of our knowledge, the fastest deep MPC based approach to visual servoing in six degrees of freedom.

In Chapter4 we provided a novel approach towards pick and place sequence planning and execution. We successfully combined a data-driven methodology with model-based optimization to develop a fast **Sequencer** and a **Executor**. At the heart of our approach lies the DeLaN neural network [7] whose positive definite structure proved crucial in learning the optimal arc-length between two manipulator end-effector positions. We showed significant performance gain over the baseline formulations and over a neural network that does not have metric preserving properties. Specifically, the proposed method outperforms overall joint space trajectory arc-length and trajectory execution time over the baselines. To the best of our knowledge, this is the first method that identifies and showcases the role of learnable metrics in the optimal sequence planning and execution. Our work has the potential to improve pick and place operation in manufacturing pipelines dramatically, and we seek to evaluate this in the future experimentally. We are also currently working on integrating obstacle avoidance into our DeLaN Metric IK. We conjecture that the DeLaN value function will bias the manipulator to choose such directions for obstacle avoidance from where it becomes easier to come back to the goal or the desired trajec-

tory. We also aim to learn a richer class of value function that can capture optimality also in terms of manipulability index.

The approaches described in this thesis hold significant promise for making a meaningful impact across various industries, including medical robotics and factory pick-and-place planning. We anticipate that these methods will stimulate further investigation into integrating contemporary neural network-based techniques with traditional control problems, thereby contributing to the development of general-purpose robots.

Related Publications

1 **M. Nomaan Qureshi**, P. Katara, A. Gupta, H. Pandya, Y. V. S. Harish, AadilMehdi Sanchawala, G. Kumar, B. Bhowmick and K. M. Krishna, " RTVS: A Lightweight Differentiable MPC Framework for Real-Time Visual Servoing," in *2021 IEEE International Conference on Intelligent Robots and Systems (IROS)*.IEEE, 2021

2 Prajwal Thakur* **M. Nomaan Qureshi***, Arun Kumar Singh, Y V S Harish, Pushkal Katara, Houman Masnavi, K. Madhava Krishna and Brojeshwar Bhowmick " Learning Arc-Length Value Function for Fast Time-Optimal Pick and Place Sequence Planning and Execution.," *International Joint Conference on Neural Networks, 2023 (IJCNN, 2023)*

3 Harshit K. Sankhla*, **M. Nomaan Qureshi***, Shankara Narayanan V.*, Vedansh Mittal, Gunjan Gupta, Harit Pandya, K. Madhava Krishna " Flow Synthesis Based Visual Servoing Frameworks for Monocular Obstacle Avoidance Amidst High-Rises," in *2021 IEEE International Conference on Intelligent Robots and Systems (CASE)*.IEEE, 2022

4 **M. Nomaan Qureshi**, Ben Eisner, David Held "Deep Sequenced Linear Dynamical Systems for Manipulation Policy Learning" in *Generalized Policy Learning Workshop, International Conference on Representation Learning 2022*.

5 Vladimír Petrík, **M. Nomaan Qureshi**, Josef Sivic, Makarand Tapaswi "Learning Object Manipulation Skills from Video via Approximate Differentiable Physics" in *2023 IEEE International Conference on Intelligent Robots and Systems (IROS)*.IEEE, 2023

Bibliography

[1] Cunjun Yu, Zhongang Cai, Hung Pham, and Quang-Cuong Pham. Siamese convolutional neural network for sub-millimeter-accurate camera pose estimation and visual servoing. *arXiv preprint arXiv:1903.04713*, 2019.

[2] Quentin Bateux, Eric Marchand, Jürgen Leitner, François Chaumette, and Peter Corke. Training deep neural networks for visual servoing. In *IEEE ICRA*, pages 1–8. IEEE, 2018.

[3] Aseem Saxena, Harit Pandya, Gourav Kumar, Ayush Gaud, and K Madhava Krishna. Exploring convolutional networks for end-to-end visual servoing. In *IEEE ICRA*, pages 3817–3823. IEEE, 2017.

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

[5] Y V S Harish, Harit Pandya, Ayush Gaud, Shreya Terupally, Sai Shankar, and K Madhava Krishna. Dfvs: Deep flow guided scene agnostic image based visual servoing. In *IEEE ICRA*, pages 3817–3823. IEEE, 2020.

[6] P. Katara, 'Y V S. Harish, H. Pandya, A. Gupta, A. Sanchawala, G. Kumar, B. Bhowmick, and K. M. Krishna. Deepmpcvs: Deep model predictive control for visual servoing. *4th Annual Conference on Robot Learning (CoRL)*, 2020.

[7] Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning, 2019.

[8] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[9] R. Smits. KDL: Kinematics and Dynamics Library. http://www.orocos.org/kdl.

[10] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[11] Francisco Suárez-Ruiz, Teguh Santoso Lembono, and Quang-Cuong Pham. Robotsp - a fast solution to the robotic task sequencing problem, 2017.

[12] Peter I Corke. Visual control of robot manipulators–a review. In *Visual Servoing: Real-Time Control of Robot Manipulators Based on Visual Sensory Feedback*, pages 1–31. World Scientific, 1993.

[13] Nicolas Andreff, Bernard Espiau, and Radu Horaud. Visual servoing from lines. *The International Journal of Robotics Research*, 21(8):679–699, 2002.

[14] Ezio Malis, Graziano Chesi, and Roberto Cipolla. 212d visual servoing with respect to planar contours having complex and unknown shapes. *The International Journal of Robotics Research*, 22(10-11):841–853, 2003.

[15] Ian Lenz, Ross A. Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

[16] Brandon Amos and Denis Yarats. The differentiable cross-entropy method, 2020.

[17] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks, 2016.

[18] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. In *Conference on Robot Learning*, pages 420–429. PMLR, 2020.

[19] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion, 2016.

[20] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning, 2016.

[21] Alex X Lee, Sergey Levine, and Pieter Abbeel. Learning visual servoing with deep features and fitted q-iteration. *arXiv preprint arXiv:1703.11000*, 2017.

[22] Noriaki Hirose, Fei Xia, Roberto Martín-Martín, Amir Sadeghian, and Silvio Savarese. Deep visual mpc-policy learning for navigation. *IEEE Robotics and Automation Letters*, 4(4):3184–3191, 2019.

[23] Sergey Alatartsev, Sebastian Stellmacher, and Frank Ortmeier. Robotic task sequencing problem: A survey. *Journal of Intelligent Robotic Systems*, 03 2015.

[24] Y. Edan, T. Flash, U. M. Peiper, I. Shmulevich, and Y. Sarig. Near-minimum-time task planning for fruit-picking robots. *IEEE Transactions on Robotics and Automation*, 7(1):48–56, 1991.

[25] 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, 2020.

[26] *The Traveling Salesman Problem*, pages 527–562. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[27] Fred Glover, Gregory Gutin, Anders Yeo, and Alexey Zverovich. Construction heuristics for the asymmetric tsp. *European Journal of Operational Research*, 129:555–568, 03 2001.

[28] S. Dubowsky and T.D. Blubaugh. Planning time-optimal robotic manipulator motions and work places for point-to-point tasks. *IEEE Transactions on Robotics and Automation*, 5(3):377–381, 1989.

[29] Ewa Kolakowska, Stephen F. Smith, and Morten Kristiansen. Constraint optimization model of a scheduling problem for a robotic arm in automatic systems. *Robotics and Autonomous Systems*, 62(2):267–280, 2014.

[30] C. Wurll, D. Henrich, and H. Wörn. Multi-goal path planning for industrial robots. 1999.

[31] Shuai D Han, Nicholas M Stiffler, Athanasios Krontiris, Kostas E Bekris, and Jingjin Yu. Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps. *The International Journal of Robotics Research*, 37(13-14):1775–1795, 2018.

[32] 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.

[33] M. Saha, G. Sanchez-Ante, and J.-C. Latombe. Planning multi-goal tours for robot arms. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 3, pages 3797–3803 vol.3, 2003.

[34] L. Palmieri and K. O. Arras. Distance metric learning for rrt-based motion planning with constant-time inference. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 637–643, 2015.

[35] 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.

[36] Yanbo Li and Kostas E Bekris. Learning approximate cost-to-go metrics to improve sampling-based motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4196–4201. IEEE, 2011.

[37] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. Value function approximation and model predictive control. In *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pages 100–107. IEEE, 2013.

[38] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan—episodic mpc improvement. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 336–343. IEEE, 2017.

[39] Lih-Yuan Deng. The cross-entropy method: a unified approach to combinatorial optimization, monte-carlo simulation, and machine learning, 2006.

[40] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018.

[41] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[42] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[43] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[44] Laurent Perron and Vincent Furnon. Or-tools.

[45] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[46] Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, second edition, 2017. ISBN 978-3-319-54413-7.

[47] Hung Pham and Quang-Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 34(3):645–659, 2018.