Fast Kinodynamic Planning on the Constraint Manifold With Deep Neural Networks

Piotr Kicki, *Member, IEEE*, Puze Liu, Davide Tateo, *Member, IEEE*, Haitham Bou-Ammar, Krzysztof Walas, *Member, IEEE*, Piotr Skrzypczyński, *Member, IEEE*, and Jan Peters, *Fellow, IEEE*

***Abstract*—Motion planning is a mature area of research in robotics with many well-established methods based on optimization or sampling the state space, suitable for solving kinematic motion planning. However, when dynamic motions under constraints are needed and computation time is limited, fast kinodynamic planning on the constraint manifold is indispensable. In recent years, learning-based solutions have become alternatives to classical approaches, but they still lack comprehensive handling of complex constraints, such as planning on a lower dimensional manifold of the task space while considering the robot's dynamics. This article introduces a novel learning-to-plan framework that exploits the concept of constraint manifold, including dynamics, and neural planning methods. Our approach generates plans satisfying an arbitrary set of constraints and computes them in a short constant time, namely the inference time of a neural network. This allows the robot to plan and replan reactively, making our approach suitable for dynamic environments. We validate our approach on two simulated tasks and in a demanding real-world scenario, where we use a Kuka LBR Iiwa 14 robotic arm to perform the hitting movement in robotic air hockey.**

Manuscript received 7 May 2023; revised 9 October 2023; accepted 15 October 2023. Date of publication 23 October 2023; date of current version 15 December 2023. This paper was recommended for publication by Associate Editor E. De Momi and Editor N. Amato upon evaluation of the reviewers' comments. The work of Piotr Skrzypczyński was supported by TAILOR, a project funded by EU Horizon 2020 under Grant 952215. The work of Krzysztof Walas was supported by REMODEL, a project funded by EU Horizon 2020 under Grant 870133. This work was supported in part by the CSTT fund from Huawei Tech R&D (U.K.), in part by the China Scholarship Council under Grant 201908080039, in part by the German Federal Ministry of Education and Research (BMBF) within a subproject "Modeling and exploration of the operational area, design of the AI assistance as well as legal aspects of the use of technology" of the collaborative KIARA Project under Grant 13N16274. The work of Piotr Kicki was supported in part by the Polish National Agency for Academic Exchange (NAWA) under the STER programme "Towards Internationalization of Poznan University of Technology Doctoral School" (2022-2024) and in part by the PUT under Grant 0214/SBAD/0235. *(Corresponding author: Piotr Kicki.)*

Piotr Kicki, Krzysztof Walas, and Piotr Skrzypczyński are with the Institute of Robotics and Machine Intelligence, Poznan University of Technology, 60-965 Poznan, Poland (e-mail: piotr.kicki@put.poznan.pl; krzysztof.walas@put.poznan.pl; piotr.skrzypczynski@put.poznan.pl).

Puze Liu and Davide Tateo are with the Department of Computer Science, Technische Universität Darmstadt, 64289 Darmstadt, Germany (e-mail: puze@robot-learning.de; davide.tateo@tu-darmstadt.de).

Haitham Bou-Ammar is with the Huawei R&D London, CB4 0WG Cambridge, U.K. (e-mail: haitham.ammar@huawei.com).

Jan Peters is with the Department of Computer Science, Technische Universität Darmstadt, 64289 Darmstadt, Germany, also with the Research Department: Systems AI for Robot Learning, German Research Center for AI (DFKI), 67663 Kaiserslautern, Germany, and also with the Hessian, 64293 Darmstadt, Germany (e-mail: jan.peters@tu-darmstadt.de).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TRO.2023.3326922.

Digital Object Identifier 10.1109/TRO.2023.3326922

***Index Terms*—Kinodynamic planning, learning to plan, motion planning, neural networks.**

I. Introduction

ROBOTIC manipulators tackle a wide variety of complex tasks in dynamic environments, such as ball-in-a-cup [1], [2], table tennis [3], [4], juggling [5], diabolo [6], and air hockey [7], [8]. These tasks require quick computation of a feasible trajectory, i.e., a trajectory solving the task while respecting all kinematic and dynamic constraints, such as joint position, velocity, acceleration, and torque limits. On top of that, many tasks define a set of safety or task-specific constraints that must be enforced on the planned solutions.

Although there have been attempts to solve these problems using optimization and sampling-based motion planning algorithms, they all have significant limitations, such as long planning time, computing hard-to-follow plans, or inability to satisfy boundary conditions. One of the approaches to the planning on the constraint manifold is to represent it as a collision-free space, which makes sampling valid states highly improbable [9], [10], resulting in considerably increased planning time. To address this issue, Berenson et al. [11] introduced the concept of projecting the states onto the constraint manifold. However, this approach is restricted to plan paths, which can be hard to follow, as they disregard the system's dynamics. Recent work of Bordalba et al. [12] proposed to solve constrained kinodynamic motion planning problems by building a topological atlas of the constraint manifold and then using linear quadratic regulator (LQR) to control the system locally. Even though this approach allows for solving complex problems, the motion planning times are prohibitively long for dynamic tasks, such as the game of air hockey or table tennis. Lengthy planning is also noticeable for optimization-based motion planners, which slow down significantly when subjected to highly nonlinear constraints [13] and are prone to get stuck in local minima. An efficient alternative to the abovementioned methods can be to build a reduced actuator space that keeps the system on states satisfying the constraints [14]. However, this approach may have problems with the satisfaction of boundary constraints.

We propose a novel learning-based approach for constrained kinodynamic planning that efficiently solves all the abovementioned problems, called constrained neural motion planning with B-splines (CNP-B). Similarly to [11], we frame the problem as planning on a constraint manifold $\mathcal{M}$. Using a similar derivation to [14], we define all the kinematics, dynamics, and safety/task constraints as a single constraint manifold, defined as the zero

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2024, Yokohama, Japan. Cite as T-RO paper.

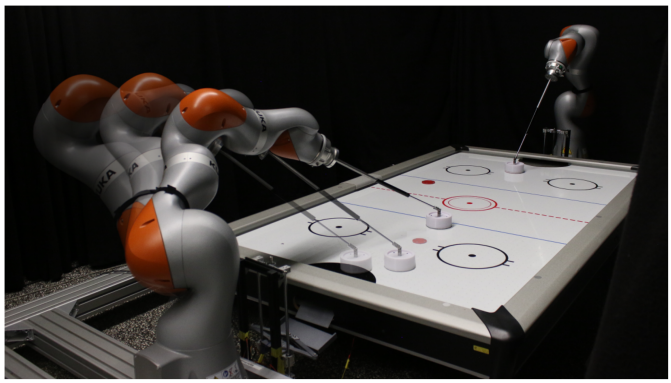


Fig. 1. Proposed learning-based motion planning approach enables rapid planning and replanning of smooth trajectories under complex kinodynamic constraints in dynamic scenarios, e.g., the robotic air hockey hitting.

level set of an arbitrary constraint function $c(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$. Our approach does not use an online projection [11] nor continuation [12]. Instead, it exploits the representation power of deep neural networks to learn a planning function [15], indirectly encoding the manifold structure. Implicitly learning the manifold enables us to rapidly plan smooth, dynamically feasible trajectories under arbitrary constraints and highly dynamic motions (see Fig. 1). Differently from [14], our approach does not require building an abstract action space, making it easy to exploit standard planning heuristics and improving the interpretability of the plan. In our work, we frame constraint satisfaction as a manifold learning problem, where the planning function is encouraged to generate trajectories that minimize not only some arbitrary task cost but also the distance from the constraint manifold. Furthermore, violating particular constraints to a certain degree can have only a minor negative impact, while violating other constraints can be unacceptable or dangerous. Hence, we estimate the metric of the constraints manifold, which allows us to attribute different priorities to different constraints.

A. Contributions

We propose a novel learning-to-plan method for fast constrained kinodynamic planning, which is based on a deep neural network generating trajectories represented using two B-spline curves. This article presents the following contributions.

1) A new approach to robotic arm motion planning that enables planning dynamic trajectories under constraints. This approach is inspired by two ideas from our recent works on car maneuver planning: learning from experience with a deep neural network [16] and using B-splines for efficient representation of the learned path [15]. This work builds on these concepts and extends them to planning a trajectory on a lower dimensional manifold of the task space while considering the robot's dynamics. To this end, we extend the learning system architecture by learning the Lagrangian multipliers in the optimization problem, resulting in the learning of the metric of our constraint manifold. This novel approach allows us to weigh each constraint by its importance in finding a feasible solution to the planning problem. We demonstrate in simulations and experiments that this new approach is not only much faster than all state-of-the-art motion planning methods we considered as baselines, but also generates trajectories that allow faster and more accurate robot motion while being executed.[1]
2) A new technique to enforce satisfaction of the boundary constraints, such that the trajectory inferred by the neural network connects precisely two arbitrary robot configurations (positions, velocities, and higher order derivatives). We demonstrate its effectiveness in both simulations and real-world experiments for two different tasks.
3) Finally, we demonstrate the practical feasibility of our novel planning technique in the air hockey hitting task on a real-world Kuka LBR Iiwa 14 robot. While this experimental benchmark was previously successfully solved by specialized motion planning algorithms [8], we show the effectiveness of our general planning method under a strict time budget, and we demonstrate its ability to replan motion on the fly. These properties allow for precise dynamic hitting motion, outperforming the task-specific baseline [8].

B. Limitations

The main drawback of our methodology is that it may return unfeasible plans. However, this issue is heavily mitigated by the fact that checking for the feasibility of a plan is often faster than generating a feasible trajectory, allowing us to prevent the execution of dangerous motions. Furthermore, from our empirical results, our planner has a much higher success rate than the other state-of-the-art planners on all the tasks considered in this work. Finally, it is possible to employ standard gradient optimization techniques to fix trajectories that slightly violate the constraints or fall back to any different motion planner with a minimal time overhead. The last drawback of the planner is that this approach is limited to local trajectory planning, and is not suited for nonconvex obstacle avoidance.

Similarly to most planning algorithms, we assume perfect knowledge of system dynamics such that the model mismatch is handled by the tracking algorithm only. Additionally, our method requires differentiable objective functions. However, in principle, it is possible to extend this approach by using estimated derivatives and turning the problem into a reinforcement learning one.

C. Problem Statement

Let $\boldsymbol{q}(t) \in \mathbb{R}^n$ be a vector of n generalized coordinates describing a mechanical system at time t. Let $\dot{\boldsymbol{q}}(t)$ and $\ddot{\boldsymbol{q}}(t)$ be the first and second derivative w.r.t. time, i.e., velocity and acceleration of the coordinate vector $\boldsymbol{q}$. Let the tuple $\boldsymbol{\zeta} = \langle \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, T \rangle$ be a trajectory of the system for $t \in [0, T]$, with T the duration of the given trajectory. In the following, we assume that $\boldsymbol{\zeta}(t) = \langle \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t) \rangle$ is a tuple containing the information of the trajectory $\boldsymbol{\zeta}$ at timestep t. We define the constrained planning

[1] Code and data can be found at https://github.com/pkicki/cnp-b

problem as

$$\begin{aligned} \operatorname{argmin}_{\boldsymbol{\zeta}} \quad & \mathcal{L}(\boldsymbol{\zeta}) \\ \text{s.t.} \quad & c_i(\boldsymbol{\zeta}(t), t) = 0 \qquad \forall t \quad \forall i \in \{1, \ldots, N\} \\ & g_j(\boldsymbol{\zeta}(t), t) \leq 0 \qquad \forall t \quad \forall j \in \{1, \ldots, M\} \end{aligned} \tag{1}$$

where N is the number of equality constraints c_i, M is the number of inequality constraints g_j, and $\mathcal{L}$ is an arbitrary loss function describing the task. Typically, to solve a motion planning task, it is enough to generate a trajectory that satisfies some locally defined properties, such as limited joint velocity, torque, or ensuring manipulation in free space, at every time step t, and minimize locally defined objectives due to composability. We can exploit this by transforming the objective function of the considered problem into an integral, and describe it by

$$\mathcal{L}(\boldsymbol{\zeta}) = \int_0^T \mathcal{L}_t(\boldsymbol{\zeta}(t), t)dt = \int_0^T \mathcal{L}_t(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \ddot{\boldsymbol{q}}(t), t)dt \tag{2}$$

where $\mathcal{L}_t$ is a locally defined loss function.

Instead of specifying the trajectory $\boldsymbol{\zeta}$ at every point, it is helpful to use parametric representations, such that the generalized coordinates and their derivatives can be determined based on some set of parameters $\boldsymbol{\theta}$. Therefore, we can formalize the problem as minimization on the parameters $\boldsymbol{\theta}$ instead of searching for $\boldsymbol{q}$ in the functional space $\mathcal{C}^2$.

II. Related Work

Motion planning is one of the core components of a robotic system, and it is essential to achieve the desired degree of autonomy. This critical role explains why motion planning is one of the most active research areas in robotics. Despite these efforts, for many robotic tasks, known planning algorithms are inadequate as they cannot handle the task-specific constraints or yield the plan within the specified amount of time. Researchers made great progress in motion planning in the last two decades, through the development of sampling-based motion planners, such as probabilistic roadmaps [17] and rapidly exploring random trees (RRT) [18], and their more recent successors, such as RRT$*$ [19], bidirectional fast marching trees* [20], or batch informed trees$*$ [21]. These methods are not only characterized by probabilistic completeness but also enable planning paths for challenging problems, such as maneuvering with a car-like vehicle [22] or reaching given poses with two robotic manipulators in a cluttered household environment [23].

In parallel with the sampling-based approaches, optimization-based motion planning methods were developed. Instead of building a search tree and constructing the solution, these methods start with an initial solution and modify it to minimize cost function subject to constraints. Methods, such as CHOMP [24], TrajOpt [25], or GPMP2 [26], showed an ability to solve tasks, such as avoiding obstacles with PR2 robot [25], [26] or planning the walking movements of a quadruped [24].

Proposing a collision-free trajectory for the considered class of planning problems does not mean that a practical solution is reached, as planning in real-world scenarios often involves additional constraints related to time (e.g., limited time for computing the plan or limited time to complete the motion), or related to the physics of the task (e.g., limited effort of the robot motors or constrained movement of the robot end effector). From these constraints, the areas of constrained and kinodynamic motion planning emerged.

A. Constrained Motion Planning

Some of the tasks we outsource to robots require constraining the robot's motion to some manifold embedded in the task space, e.g., keeping the cup held by the robot in an upright position or ensuring that the end effector of a window cleaning robot remains at some constant distance from the window plane. In recent years, two main approaches to this class of problems were developed: 1) adding task constraints to the motion optimization problem [27], 2) sampling constraint-satisfying configurations and generating constraint-satisfying motions in the sample-based motion planners [9], [10].

While including additional task constraints in motion optimization is conceptually simple, it usually results in a much harder optimization problem to solve, as the kinematics of the robot are usually nonlinear. To overcome the computational burden stemming from these nonlinearities, Schulman et al. [25] proposed to employ the sequential quadratic programming approach that transforms the nonlinear programming (NLP) problem into a sequence of its convex approximations around the current solution. In [27], this idea was extended to a non-Euclidean setting, by transforming the manifold-constrained trajectory optimization problem into an equivalent problem defined over a space with Euclidean structure. Another general approach to solve manifold-constrained problems was proposed in [28], where authors modified the update rule of CHOMP by projecting it on the tangent space of the constraint manifold and adding an offset correction term to move the solution toward the manifold. In turn, a recent approach to constrained optimization-based motion planning [29] tries to achieve faster convergence and numerical robustness by utilizing the augmented Lagrangian optimization using an iterative linear quadratic regulator (iLQR), introduced in [30], and systematically updating the Lagrangian multipliers, which is conceptually similar to our approach to neural network training. However, in practice, when we need a solution for a particular problem, exploiting the structure of the constraints may be beneficial. In [8], the problem of fast planning for the dynamic motion of a robotic arm with the end-effector constrained to move on a plane was decoupled into two more straightforward optimization problems: planning the Cartesian trajectory on a 2-D plane and then planning the trajectory in the joint space. However, this may result in a suboptimal performance due to the predefined Cartesian trajectory. In contrast, in our work, we plan directly in the robot's configuration space, and we show that it is possible to use a general solution to learn how to plan better than using handcrafted optimization.

The adaptation of sampling-based motion planners to solve constrained motion planning problems is not so straightforward. Simply rejecting the samples, which lie outside of the manifold usually results in a drastically reduced probability of drawing an

acceptable sample, also connecting two samples in a manifold-constrained way is nontrivial [9], [10]. Therefore, a number of approaches to this class of problems were proposed, such as relaxation, projection, using tangent space, or topological atlas. The relaxation emerged from the assumption that there is some acceptable level of violation of the constraints, and we can relax the surface of the constraint-manifold and give it some nonzero volume to enable sampling [31], [32]. However, this approach bypasses the problem by creating a new one—planning in narrow passages [33]. To mitigate this, constrained bidirectional RRT (CBiRRT) was proposed [11]. This method exploits projections onto the constraint manifold to make sampled and interpolated points satisfy the constraint. Wang et al. [34] modified CBiRRT to avoid the search getting stuck in local minima by exploiting the geometric structure of the constraint manifold. A significant speed-up over projection-based methods was achieved by Tangent bundle RRT [35], which, instead of sampling in the configuration space, proposes to sample in the tangent space of the constraint manifold. Instead of sampling on the tangent space, it is possible to define charts that locally parameterize the manifold and coordinate them by creating a topological atlas [36]. In this approach, RRT searches for the direction of the atlas expansion, and then sampling is performed only in the space defined by the atlas.

Nonetheless, sampling-based constrained motion planning methods usually produce only paths, disregarding the dynamics of the movement that is crucial for planning for dynamic robot movements. Therefore, in the following section, we discuss the approaches to kinodynamic motion planning.

B. Kinodynamic Motion Planning

Transforming motion optimization problems to the kinodynamic setting can be easily done by including the robot dynamics in the constraints. However, this introduces more nonlinearities to the problems, slows down the solvers, and makes optimization prone to converge to local minima unless properly initialized [13].

In turn, to adopt sampling-based motion planners to the kinodynamic setting, a significant effort has to be made in terms of both the tree extend procedure and state cost-to-go assessment. One of the first attempts to solve these issues was to linearize the system and use LQR to connect states and calculate the cost of this connection [37]. To avoid problems stemming from linearization, Stoneman and Lampariello [38] proposed to use a gradient-based method to solve the NLP problem of connecting any two configurations. A similar idea was recently presented in [39], where model predictive control (MPC) was used to perform a tree extension. One of the issues of sampling-based kinodynamic planning is the necessity of sampling high-dimensional state space (due to the inclusion of the velocities in the state). To mitigate this, in [40], a partial-final-state-free optimal controller was used for connecting the states, such that the dimensionality of the sampling space is divided by two. A different approach to RRT-based planning is to sample controls instead of states [41], [42]. This significantly simplifies keeping the constraints satisfied, however, it usually results in relatively slow planning. Unfortunately, most of the aforementioned sampling-based kinodynamic motion planning algorithms struggle when they are subjected to kinodynamic and holonomic constraints at the same time [12]. To address this issue, an adaptation of the atlas method [36] with the RRT's Extend procedure using LQR was proposed in [12]. Unfortunately, this algorithm relies strongly on the linearization of the system and requires a significant amount of time to prepare the plan.

C. Learning-Based Motion Planning

For many real-world robotic applications, conventional planning methods are not fast enough to meet the processing time constraints imposed by real-world dynamics (e.g., for rapid car maneuvers [16]) or produce solutions far from optimal. Moreover, tasks performed by the robots, even though they may require solving different planning problems, seem to reveal some similarities. Therefore, to obtain plans of better quality within tighter time bounds, learning-based approaches to robot motion planning were proposed. These types of methods originated from [43], in which, the conventional planner was used to solve the problems, however, generated plans were not only executed but also saved in the library of solved problems. After gathering plans, they can be reused to solve new but similar problems in a shorter time by simply modifying the most appropriate known path.

Most of the learning-based motion planning methods build upon the algorithms from RRT family to use their guaranties of probabilistic completeness. Some of these methods try to modify the sampling distribution based on the experience and the representation of the task. In [44], neural network decides whether the sampled state should be rejected or not, whereas in [45], neural network is used to predict the state-action value function to choose the best nodes to expand. Molina et al. [46] used a convolutional neural network to predict some crucial regions of the environment to increase the sampling in them. At the same time, in [47], a probability distribution is directly inferred by the neural network in the form of a heatmap. In [48], the sampling distribution is encoded in a stochastic neural network (due to dropout used during inference), which is trained to predict the next state toward the goal. An adaptation of biasing the sampling distribution to comply with a manifold of constraints was presented in [49], where adversarial training was used to learn how to generate data on the constraint manifold. An architecture consisting of a generator and discriminator trained jointly was also used in [50]. This work builds upon [48] and uses a generator to predict the next state, but extends it with a discriminator, which is used to predict deviation from the constraint manifold and to project the predictions on it.

Learning-based approaches are also used to improve the feasibility of solving kinodynamic motion planning problems. Wolfslag et al. [51] emphasized that solving a two-point boundary value problem for a nonlinear dynamical system is NP-hard and proposed to learn a distance metric and steering function to connect two nodes of RRT, based on the examples generated using optimal control. In turn, in [52], the learning of control policy is done indirectly by learning the state trajectory of some optimal motion planner and then using inverse dynamics of the system to determine controls. A lazy approach to limit

the number of computationally expensive NLP solver calls was presented in [53], where neural network, instead of replacing the solver, is trained to predict which nodes of an RRT are steerable to, and what will be the cost of steering. Contrary to these methods, Li et al. [54] proposed to use an MPC extensively to solve local NLP problems and learn only how to determine the next state in a way toward the goal by mimicking demonstration trajectories.

In contrast to these methods, our proposed approach offers an extremely fast way of finding a near-optimal trajectory that is able to solve constrained kinodynamic motion planning problems. Moreover, CNP-B does not require the demonstration trajectories, as it learns from its own experience similarly to [16]. Thus, its performance is not upper bounded by the quality of the demonstrations. Furthermore, our proposed approach is able to smoothly replan the motion on the fly, which is not possible with the use of state-of-the-art constrained kinodynamic motion planning algorithms.

III. Proposed Solution

In this section, we first describe the general framework of our proposed solution to the problem defined in (1). Then, we show how to use the structure of the proposed approach to solve the problem of planning a trajectory where the objective is to reach a specific end configuration in the shortest possible amount of time. Finally, we show some extensions to CNP-B to solve a real planning problem. Namely, we focus on moving a heavy object and hitting a puck with a mallet using a 7 degrees of freedom manipulator robot.

A. Constrained Planning

The original problem formulation, as shown in (1), is a complex constrained optimization problem. Directly using this formulation makes finding a solution hard and time consuming, particularly in the presence of complex equality constraints. Indeed, while an exact solution may exist, it may be hard to perfectly match the constraints, due to the use of parametrized trajectory and numerical errors.

To efficiently solve the considered constrained planning problem, we reformulate it into an unconstrained one by incorporating the constraints into the objective function and scaling them accordingly. We formalize the constraints in the notion of the constraint manifold and allow the solutions to lie close to it, taking into account the violation budget for each constraint. In our approach, we treat the constraint scaling factors as a metric of the constraint manifold and update it during the learning process.

We base our reformulation on the following steps:

1) defining the constrained manifold as the zero-level curve of an implicit function, thus unifying equality and inequality constraints;
2) relaxing the original problem by imposing that all solutions should lie in the vicinity of the constraint manifold;
3) transforming this problem into an unconstrained optimization problem by learning the metric of the constraint manifold, using the assumption of small updates.

1) Defining the Constraint Manifold: Our first step toward the definition of the constraint manifold is to eliminate inequality constraints from (1), by introducing slack variables μ_j and M new equality constraints $c_{N+j}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, t, \mu_j) = g_j(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, t) + \mu_j^2$. Thus, we obtain the following formulation:

$$\begin{aligned} &\operatorname{argmin}_{\boldsymbol{\zeta}} \quad \mathcal{L}(\boldsymbol{\zeta}) \\ &\quad \text{s.t.} \quad c_i(\boldsymbol{\zeta}(t), t, [\mu_{i-N}]) = 0 \quad \forall t \quad \forall i \in \{1, \dots, N+M\} \end{aligned}$$

where the squared brackets indicate an optional argument and the set $\{(\boldsymbol{\zeta}(t), \mu) \,|\, c_i(\boldsymbol{\zeta}(t), t, [\mu_{i-N}]) = 0, \quad \forall i \in \{1, \dots, N+M\}\}$ defines the constraints manifold $\mathcal{M}_\mu$. To drop the dependency of the manifold $\mathcal{M}_\mu$ from the vector of slack variables $\boldsymbol{\mu} \in \mathbb{R}^M$, we first introduce the manifold loss function

$$\begin{aligned} \mathcal{L}_{\mathcal{M}}(\boldsymbol{\zeta}(t), t, \boldsymbol{\mu}) = \|\boldsymbol{c}(\boldsymbol{\zeta}(t), t, \boldsymbol{\mu})\|^2 &= \sum_{i=1}^{N+M} c_i(\boldsymbol{\zeta}(t), t, [\mu_{i-N}])^2 \\ &= \sum_{i=1}^{N+M} \mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t, [\mu_{i-N}]) \end{aligned} \tag{3}$$

where $\|\cdot\|$ denotes an L2 norm, such that the constraint manifold $\mathcal{M}$ will be defined by its 0-level set. For ith relaxed inequality constraint the manifold loss function is defined by

$$\begin{aligned} \mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t, \mu_i) = c_i(\boldsymbol{\zeta}(t), t, \mu_i)^2 &= \left(c_i(\boldsymbol{\zeta}(t), t) + \mu_i^2\right)^2 \\ &= c_i(\boldsymbol{\zeta}(t), t)^2 + 2c_i(\boldsymbol{\zeta}(t), t)\mu_i^2 + \mu_i^4 \end{aligned}$$

therefore, to drop the dependency of μ_i, we redefine our loss for inequality constraint as follows:

$$\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t) = \min_{\mu} \mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t, \mu_i).$$

We can find a value that minimizes $\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), \mu_i)$ by taking the derivative w.r.t. the slack variable and setting it to zero

$$\frac{d}{d\mu}\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t, \mu_i) = 4c_i(\boldsymbol{\zeta}(t), t)\mu_i + 4\mu_i^3 = 0$$

thus, we obtain $\mu_i = 0 \vee \mu_i = \pm\sqrt{-c_i(\boldsymbol{\zeta}(t), t)}$, where the second solution exists only for $c_i(\boldsymbol{\zeta}(t), t) < 0$. Plugging back the obtained stationary points in the solution, by basic reasoning about the obtained values and conditions, we obtain our inequality loss

$$\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t) = \begin{cases} c_i(\boldsymbol{\zeta}(t), t)^2 & c_i(\boldsymbol{\zeta}(t), t) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

This loss can be computed compactly, in an equivalent form

$$\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t) = \left(\max\left(c_i(\boldsymbol{\zeta}(t), t), 0\right)\right)^2 \tag{4}$$

which is continuous and differentiable everywhere: notice that in 0, the derivative is 0. As a result, we obtain a new definition of the constraint manifold

$$\mathcal{M} \triangleq \{\boldsymbol{\zeta}(t) \,|\, \mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t) = 0 \quad \forall t \quad \forall i \in \{1, \dots, N+M\}\}.$$

2) Approximated Optimization Problem: Our newly defined manifold loss expresses the distance of a given trajectory $\boldsymbol{\zeta}$ from the manifold $\mathcal{M}$. Ideally, we could use the $\mathcal{L}_{\mathcal{M}}(\boldsymbol{\zeta}, t) = 0$ as a single constraint of our optimization problem. However, numerical issues and trajectory parametrization may cause small

unavoidable errors. Furthermore, often the task loss $\mathcal{L}$ and manifold constraints $\mathcal{L}^i_{\mathcal{M}}$ will counteract each other, making this optimization particularly difficult and prone to constraint violations imbalances. Indeed, the optimization may favor reducing some constraints violation at the expense of others. When considering real-world tasks, it is often not crucial to have zero constraint violation as long as it is below an acceptable threshold. Thus, we simplify the problem by introducing an acceptable level of constraints violation. For simplicity, we bound the elements of the nonnegative valued manifold loss function [see (3) and (4)], i.e., $\mathcal{L}^i_{\mathcal{M}} \leq \bar{C}_i$, where $\bar{C}_i = \bar{c}_i^2$ is a square of the desired acceptable constraint violation level $\bar{c}_i$. Therefore, we can write the following optimization problem:

$$\begin{aligned} &\operatorname{argmin}_{\boldsymbol{\zeta}} \quad \mathcal{L}(\boldsymbol{\zeta}) \\ &\quad \text{s.t.} \quad \mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t), t) \leq \bar{C}_i \quad \forall t \quad \forall i \in \{1, \ldots, N+M\}. \end{aligned} \tag{5}$$

Now, we can transform (5) into the canonical form

$$\begin{aligned} &\operatorname{argmin}_{\boldsymbol{\zeta}} \quad \mathcal{L}(\boldsymbol{\zeta}) \\ &\quad \text{s.t.} \quad \frac{\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta}(t))}{\bar{C}_i} - 1 \leq 0 \quad \forall t \quad \forall i \in \{1, \ldots, N+M\}. \end{aligned} \tag{6}$$

One possible way to solve (6) is by applying a Lagrange relaxation technique [55], framing it as an unconstrained optimization of the following function:

$$\begin{aligned} &\operatorname{argmin}_{\boldsymbol{\zeta}} \max_{\boldsymbol{\lambda}} \quad L(\boldsymbol{\zeta}, \boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\zeta}) + \sum_{i=1}^{N+M} \lambda_i \left(\frac{\mathcal{L}^i_{\mathcal{M}}(\boldsymbol{\zeta})}{\bar{C}_i} - 1 \right) \\ &\quad \text{s.t.} \quad \boldsymbol{\lambda} \geq 0 \end{aligned} \tag{7}$$

where $L(\boldsymbol{\zeta}(t), \boldsymbol{\lambda})$ is the Lagrangian and $\boldsymbol{\lambda}$ is a vector of nonnegative Lagrange multipliers.

3) Loss Function Learning: Unfortunately, (7) is not a practical optimization problem, leading to ineffective learning and convergence to local optima. We propose instead an approximate solution, inspired by [56] that learns the Lagrangian multipliers. In our approach, we decouple the min–max problem into interleaving minimization of (7) w.r.t $\boldsymbol{\zeta}$ and updating the values of the multipliers $\boldsymbol{\lambda}$. We frame the update of the multipliers as the process of learning a metric of our constraint manifold, weighting each constraint by how much it is important to solve the final task in a way that is feasible in practice.

First, we remove the maximization w.r.t $\boldsymbol{\lambda}$ from (7) and write the following minimization problem:

$$\operatorname*{argmin}_{\boldsymbol{\zeta}(t)} \mathcal{L} + \boldsymbol{c}^T \Lambda \boldsymbol{c} \tag{8}$$

where matrix Λ is a metric of the manifold $\mathcal{M}$ defined by

$$\Lambda = \operatorname{diag}\left(\frac{\lambda_1}{\bar{C}_1}, \frac{\lambda_2}{\bar{C}_2}, \ldots, \frac{\lambda_{N+M}}{\bar{C}_{N+M}} \right).$$

Note that we drop the constant 1 from the constraints, as we are not solving a min–max optimization problem, and this term has no effect on the minimization part.

Second, to simplify this metric and to maintain the positiveness of all elements of $\boldsymbol{\lambda}$ we propose the following substitution $\alpha_i = \log \frac{\lambda_i}{\bar{C}_i}$. Thus, we write Λ as

$$\Lambda = \operatorname{diag} e^{\boldsymbol{\alpha}} = \operatorname{diag}\left(e^{\alpha_1}, e^{\alpha_2}, \ldots, e^{\alpha_{N+M}}\right)$$

where $\boldsymbol{\alpha}$ is a vector of real-valued parameters defining the manifold metric. We introduce now a new loss, the manifold loss under the metric Λ

$$\mathcal{L}_{\mathcal{M},\Lambda} = \boldsymbol{c}^T \Lambda \boldsymbol{c}.$$

As a result we can rewrite (8) into

$$\operatorname{argmin}_{\boldsymbol{\zeta}} \mathcal{L}(\boldsymbol{\zeta}) + \mathcal{L}_{\mathcal{M},\Lambda}(\boldsymbol{\zeta}). \tag{9}$$

Third, we introduce the metric learning approach. While we can straightforwardly optimize (9) w.r.t. $\boldsymbol{\zeta}$ using any gradient-based optimizer, we need to derive a learning rule for the vector $\boldsymbol{\alpha}$. Let $\mathcal{L}^{\neg i}_{\mathcal{M},\Lambda} \triangleq \boldsymbol{c}^T_{\neg i} \Lambda_{\neg i} \boldsymbol{c}_{\neg i}$ be the complement of the ith manifold loss element, where index $\neg i$ means all elements except ith element. Using this notation, we analyze how the α_i should change between kth and $k+1$th iterations so as not to surpass the desired level of constraint violation $\bar{C}_i$, i.e.,

$$\mathcal{L}^{(k)} + {\mathcal{L}^{\neg i}_{\mathcal{M},\Lambda}}^{(k)} + e^{\alpha_i^{(k)}} \mathcal{L}^i_{\mathcal{M}} = \mathcal{L}^{(k+1)} + {\mathcal{L}^{\neg i}_{\mathcal{M},\Lambda}}^{(k+1)} + e^{\bar{\alpha}_i} \bar{C}_i.$$

Assuming that the changes of $\mathcal{L}^{\neg i}_{\mathcal{M},\Lambda}$ and $\mathcal{L}$ are small, e.g., by choosing a small learning rate, we obtain

$$e^{\alpha_i^{(k)}} \mathcal{L}^i_{\mathcal{M}} = e^{\bar{\alpha}_i} \bar{C}_i \Rightarrow \bar{\alpha}_i = \alpha_i^{(k)} + \log\left(\frac{\mathcal{L}^i_{\mathcal{M}}}{\bar{C}_i} \right).$$

Finally, we can define our update rule for α_i as a small step toward the desired value $\bar{\alpha}_i$, i.e.,

$$\Delta\alpha_i = \gamma(\bar{\alpha}_i - \alpha_i) = \gamma \log\left(\frac{\mathcal{L}^i_{\mathcal{M}}}{\bar{C}_i} \right)$$

with the learning rate $\gamma \in \mathbb{R}_+$.

B. Trajectory Parameterization

To represent the solution trajectory $\boldsymbol{\zeta}$ we use a decoupled representation made up of time $t = \psi(s)$ and path $\boldsymbol{p}(s) = \boldsymbol{q}(\psi(s))$, both defined as functions of the phase variable $s \in [0; 1]$, where $\psi(s)$ is monotonically increasing, $\psi(0) = 0$, and $\psi(1) = T$, where T is duration of the trajectory. Therefore, $\boldsymbol{q}(t)$ can be defined by

$$\boldsymbol{q}(t) \triangleq \boldsymbol{q}(\psi(s)) = \boldsymbol{p}(s).$$

The first derivative of $\boldsymbol{q}$ w.r.t. time can be defined by

$$\dot{\boldsymbol{q}}(t) = \frac{d\boldsymbol{p}(s)}{ds}\frac{ds}{dt} = \frac{d\boldsymbol{p}(s)}{ds} r(s)$$

where $r(s) \triangleq (\frac{dt}{ds})^{-1}(s)$ represents the inverse rate of change of the time t w.r.t. phase variable s. Thus, the second derivative of $\boldsymbol{q}$ w.r.t. time can be defined by

$$\begin{aligned} \ddot{\boldsymbol{q}}(t) &= \frac{d}{dt}\left(\frac{d\boldsymbol{p}(s)}{ds} \right) r(s) + \frac{d\boldsymbol{p}(s)}{ds}\frac{d}{dt} r(s) \\ &= \frac{d^2\boldsymbol{p}(s)}{ds^2} \left(r(s)\right)^2 + \frac{d\boldsymbol{p}(s)}{ds}\frac{dr(s)}{ds} r(s). \end{aligned}$$

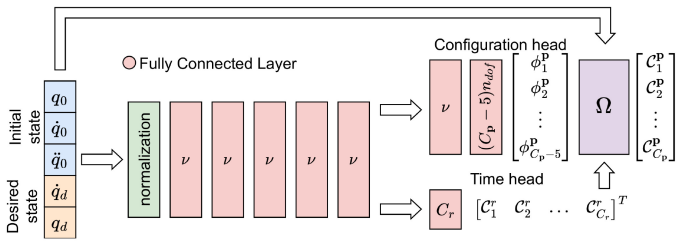


Fig. 2. Architecture of the neural network used to determine $\boldsymbol{p}(s)$ and $r(s)$ B-spline control points. The Ω Block computes the control points of the configuration B-spline $\mathcal{C}^{\boldsymbol{p}}$ using (11) and (12).

Therefore, to fully represent trajectory $\boldsymbol{\zeta}$, we need to know path $\boldsymbol{p}(s)$ and the inverse rate of change of time $r(s)$, and their derivatives w.r.t. phase variable s. In this article, we represent $\boldsymbol{p}(s)$ and $r(s)$ as B-splines, as this parametrization is rich enough to generate complex trajectories and allow us to easily compute their derivatives, which is crucial to ensure the satisfaction of the $\dot{\boldsymbol{q}}(t)$ and $\ddot{\boldsymbol{q}}(t)$ limits.

C. Trajectory Definition

To define the trajectory, we must determine the control points $\mathcal{C}^{\boldsymbol{p}}, \mathcal{C}^{r}$ of both $\boldsymbol{p}(s)$ and $r(s)$ B-splines. Most of these control points are predicted using the neural network, however, part of them can be directly computed based on the task definition. In the following, we illustrate how these points are computed in our approach.

1) Boundary Conditions: The majority of the practical motion planning problems impose some boundary constraints on their solutions. Typically, at least the initial configuration is defined. However, for practical problems, it is beneficial to consider also initial velocity and accelerations, to ensure smooth movement from the given initial state. Similarly, some tasks require accurate reaching of some desired state and velocity. These constraints can be summarized as

$$\begin{cases} \boldsymbol{q}(0) = \boldsymbol{q}_0, & \dot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}_0, & \ddot{\boldsymbol{q}}(0) = \ddot{\boldsymbol{q}}_0 \\ \boldsymbol{q}(1) = \boldsymbol{q}_d, & \dot{\boldsymbol{q}}(1) = \dot{\boldsymbol{q}}_d \end{cases} \tag{10}$$

where $\boldsymbol{q}_0, \dot{\boldsymbol{q}}_0, \ddot{\boldsymbol{q}}_0, \boldsymbol{q}_d, \dot{\boldsymbol{q}}_d$ are the initial and desired configurations and their derivatives given by the task definition. By using B-spline representations of $\boldsymbol{p}(s)$ and $r(s)$, we can impose these boundary conditions directly onto the solution of the considered planning problem. Technical details of this procedure are described in Appendix A. As a result, the first three and last two robot's configuration control points $\mathcal{C}^{\boldsymbol{p}}$ may be determined based on the boundary constraints of problem (10) instead of using the neural network outputs.

2) Neural Network: While boundary control points can be computed analytically, the rest of the control points must be determined using the learning system based on previous experience. Our proposed neural network (see Fig. 2) takes as input the normalized initial and desired configurations $\boldsymbol{q}_0, \boldsymbol{q}_d$, velocities $\dot{\boldsymbol{q}}_0, \dot{\boldsymbol{q}}_d$ and initial joint accelerations $\ddot{\boldsymbol{q}}_0$, and outputs the control points $\mathcal{C}^{r}$ of the $r(s)$ B-spline, and some parameters $\boldsymbol{\phi}^{\boldsymbol{p}}$, which can be used to compute the control points of the configuration B-spline $\mathcal{C}^{\boldsymbol{p}}$. Our proposed network is parametrized by a number of neurons in each layer ν, which is a tradeoff between the inference time and model expressiveness. Activation functions for all layers are $\tanh$, except the layer in the *Time head*, which is an exponential function (to ensure the positiveness for all control points of the $r(s)$ B-spline).

To compute control points of the configuration B-spline, we employ the Ω block that utilizes the outputs $\boldsymbol{\phi}^{\boldsymbol{p}}$ of the *Configuration head* and bias them by the boundary control points of the $\boldsymbol{p}(s)$ B-spline, determined using the task boundary conditions. For the case where the initial robot's state is defined with the configuration and its derivatives up to n_ith (exclusively), and the desired state with the configuration and its derivatives up to n_dth (inclusively), we can define inner configuration control points by

$$\mathcal{C}^{\boldsymbol{p}}_{i+n_i} = \left(\left(1 - \frac{i}{B}\right)\mathcal{C}^{\boldsymbol{p}}_{n_i} + \frac{i}{B}\mathcal{C}^{\boldsymbol{p}}_{C_{\boldsymbol{p}}-n_d}\right) + \pi\phi^{\boldsymbol{p}}_i \quad \text{for} \quad 1 \le i \le B-1 \tag{11}$$

where $C_{\boldsymbol{p}}$ is the total number of configuration control points $\mathcal{C}^{\boldsymbol{p}}$, $B = C_{\boldsymbol{p}} - (n_i + n_d - 1)$, and $\phi^{\boldsymbol{p}}_i$ is the ith output from the configuration head.

D. Loss Functions

The loss function is one of the most crucial parts of every learning system. In the proposed solution, we consider two types of losses, i.e., task loss $\mathcal{L}$ and manifold loss $\mathcal{L}_{\mathcal{M}}$. We include both of them into the integral formulation (2), such that the resultant loss function L can be defined by

$$\mathrm{L}(\boldsymbol{\zeta}) = \int_0^{\mathrm{T}} \mathrm{L_t}(\boldsymbol{\zeta}(\mathrm{t}), \mathrm{t})\mathrm{dt} = \int_0^{\mathrm{T}} \left(\mathcal{L}_{\mathrm{t}}(\boldsymbol{\zeta}(\mathrm{t}), \mathrm{t}) + \mathcal{L}_{\mathcal{M}_{\mathrm{t}}}(\boldsymbol{\zeta}(\mathrm{t}), \mathrm{t})\right)\mathrm{dt}$$

where $\mathcal{L}_t(\boldsymbol{\zeta}(t), t)$ and $\mathcal{L}_{\mathcal{M}_t}(\boldsymbol{\zeta}(t), t)$ are locally defined task and manifold losses at time t. To simplify the optimization process, we drop the explicit dependency of the loss L on the time t and trajectory duration T, by the following change of variables:

$$L(\boldsymbol{\zeta}) = \int_0^1 L_t(\boldsymbol{\zeta}(t), t)\frac{dt}{ds}ds = \int_0^1 L_t(\boldsymbol{\zeta}(s), s)r(s)^{-1}ds.$$

Therefore, to define the resultant loss function L, we must provide the formulas for the step losses $\mathcal{L}_t(\boldsymbol{\zeta}(s), s)$ and $\mathcal{L}_{\mathcal{M}_t}(\boldsymbol{\zeta}(s), s)$. In our experiments, to encourage the neural network to generate minimal-time (i.e., fast) trajectories, we define the basic task step loss as $\mathcal{L}_t(\boldsymbol{\zeta}(s)) = 1$. However, we can optimize any other quantity, such as effort, jerk, end-effector position tracking error, centrifugal forces, or a weighted sum of these.

To define step manifold loss $\mathcal{L}_{\mathcal{M}_t}(\boldsymbol{\zeta}(s), s)$ we need to define its components $\mathcal{L}^i_{\mathcal{M}_t}(\boldsymbol{\zeta}(s), s)$, which corresponds to all constraints imposed in the given problem. In our experiments, we consider constraints stemming from velocity $\bar{\dot{\boldsymbol{q}}}$, acceleration $\bar{\ddot{\boldsymbol{q}}}$, which associated losses can be defined by

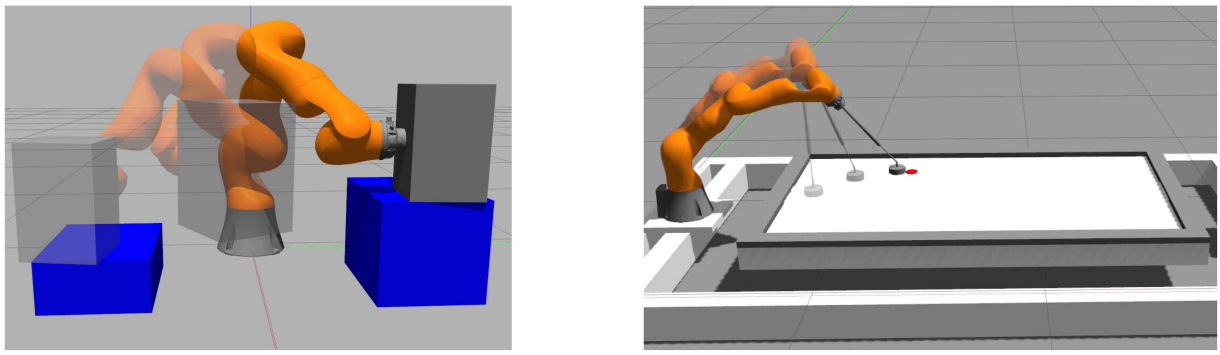


Fig. 3. Two tasks considered in this article: moving a heavy vertically oriented object between two tables (left) and high-speed air hockey hitting (right).

$$\mathcal{L}^{\dot{\boldsymbol{q}}}_{\mathcal{M}_t}(s) = \sum_{i=1}^{n_{\text{dof}}} \text{H}(\text{ReLU}(|\dot{q}_i(s)| - \bar{\dot{q}}_i))$$

$$\mathcal{L}^{\ddot{\boldsymbol{q}}}_{\mathcal{M}_t}(s) = \sum_{i=1}^{n_{\text{dof}}} \text{H}(\text{ReLU}(|\ddot{q}_i(s)| - \bar{\ddot{q}}_i))$$

where H denotes the Huber loss that we use instead of the square to reduce the impact of outliers. Similarly, we can define loss for torque limits $\bar{\boldsymbol{\tau}}$

$$\mathcal{L}^{\boldsymbol{\tau}}_{\mathcal{M}_t}(s) = \sum_{i=1}^{n_{\text{dof}}} \text{H}(\text{ReLU}(|\tau_i(s)| - \bar{\tau}_i))$$

where $\boldsymbol{\tau}(s) = \text{ID}(\boldsymbol{q}(s), \dot{\boldsymbol{q}}(s), \ddot{\boldsymbol{q}}(s))$ can be easily computed using inverse dynamics algorithm [57], [58]. Moreover, in a similar manner, we can define losses that utilize other functions of the robot configurations, such as forward kinematics.

Therefore, it is easy to define arbitrary geometrical constraints, e.g.,

$$\mathcal{L}^{\mathcal{T}}_{\mathcal{M}_t}(s) = \text{H}(d(\text{FK}(\boldsymbol{q}(s)), \mathcal{T}))$$

where $d(\text{FK}(\boldsymbol{q}(s)), \mathcal{T})$ is a distance between the robot's links configurations and some manifold $\mathcal{T}$ in the task space.

Obviously, the aforementioned losses are only examples. Using this framework, we can come up with much more complicated and sophisticated loss functions, however, the presented loss functions are enough to learn how to perform challenging practical tasks considered in this article. Note that all of these losses are differentiable functions of the trajectory $\boldsymbol{\zeta}$, enabling direct optimization of the neural network weights.

IV. Experimental Evaluation

To evaluate the proposed constrained neural motion planning framework, we introduce two challenging motion planning tasks, both considering planning for the Kuka LBR Iiwa 14 robotic manipulator: moving a heavy (i.e., of a weight close to the robot's payload limit) vertically oriented object between two tables and high-speed hitting in the game of robotic air hockey (see Fig. 3). The baseline, to which we compare the performance of CNP-B in both tasks is constituted by the following four algorithms that span the space of different solutions to the considered problem.

1) TrajOpt [25]—a motion optimization algorithm that utilizes sequential least squares programming (SLSQP).
2) CBiRRT [11]—a sampling-based path planning algorithm that uses RRTConnect with the projection of sampled points onto the constraint manifold.
3) Stable-sparse RRT (SST) [41]—a sampling-based motion planning algorithm that builds a sparse tree of robot configurations and extends them using random controls.
4) Model-predictive motion planning network (MPC-MPNet) [54]—a sampling-based motion planning algorithm, which uses a neural network to determine the next node in a search tree and cross entropy method (CEM) [59] to steer toward this configuration.

To adjust SST and MPC-MPNet to work on the constrained motion planning problems, we implemented the workspace constraints as obstacles. In the case of the equality constraints, we added margins, for moving a vertically oriented object, the object's orientation was assumed valid until the product of cosine of roll and pitch angles was bigger than 0.95, while for air hockey hitting task, we added an acceptable deviation of 1 cm from the table surface. Moreover, in the air hockey hitting task, we extend the baseline by the anchored quadratic programming (AQP) method [8], which is the current state-of-the-art in this area—an algorithm developed specifically to solve the problem of planning in the air hockey game.

Both the baselines and our method were evaluated in a simulation environment developed using ROS1 and Gazebo. The experiments in air hockey hitting were possible thanks to the in-house constructed physical setup with a Kuka LBR Iiwa 14 robot [8]. For the experiments in simulation, we used an Intel Core i7-9750H CPU, while the real-robot experiment uses AMD Ryzen 9 3900x CPU.

A. Moving a Heavy Vertically Oriented Object

1) Task Description: In this task, the goal is to quickly move a heavy cuboid (12 kg) between random positions on the left and right blue boxes using Kuka LBR Iiwa 14. The task requires planning a joint trajectory between two random configurations, minimizing movement time, and simultaneously satisfying joints' velocity, acceleration, and torque constraints. Moreover, the robot's trajectory has to ensure that both the robot and the handled object will not collide with the environment and

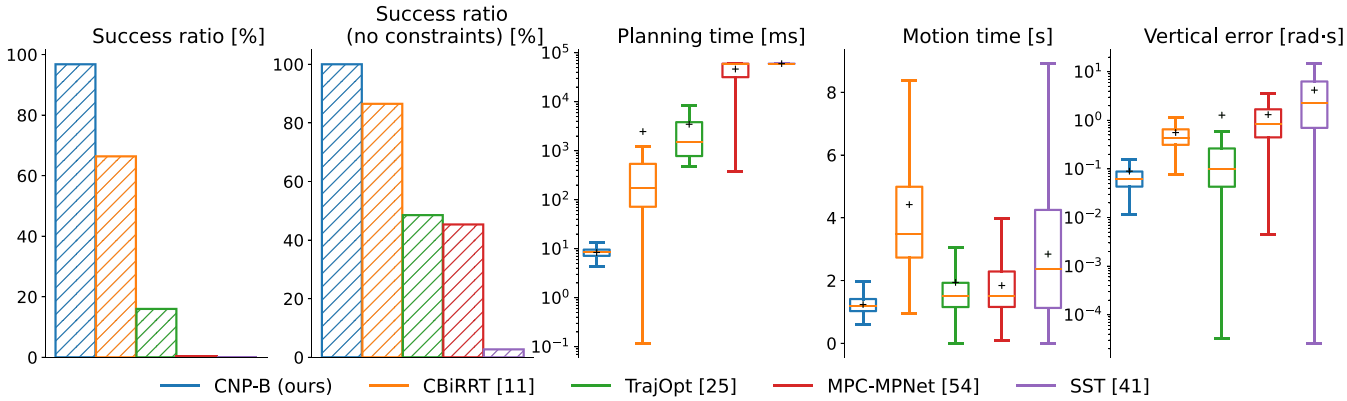


Fig. 4. Planners statistics in the task of rapid movement of a heavy object with orientation constraints and collision avoidance.

that the object will be oriented vertically throughout the whole movement.

2) Dataset and Method Adjustments: To learn how to solve the task, we generated a dataset of 26 400 planning problems of this kind, split into training (24 000) and validation (2400) subsets. In the dataset, we randomize both the initial and desired position of the object and the initial and desired robot's null-space configuration. Both initial and desired velocities are set to 0. Note that the dataset used in the training and validation contains only motion planning problems, without precomputed solutions, thanks to the use of a loss function that does not require any supervision. Therefore, the generation of the dataset does not require a significant amount of computation, and the performance of the method trained on this dataset is not bounded by the quality of the solutions included in the dataset. Moreover, we add three additional loss terms stemming from the constraints imposed by the task definition, i.e., vertical orientation loss, robot collision loss, and object collision loss. The mathematical definitions of these losses, together with all the parameters of this experiment, can be found in Appendix B1.

3) Quantitative Comparison With State-of-the-Art: To evaluate the proposed method, we compared it with several state-of-the-art motion planners. All planners were asked to plan the motions that solve 2400 randomly generated tasks, and we executed these plans in simulation. The results of this experiment are presented in Fig. 4. The first two plots show that our planner reaches the goal in all scenarios, and in nearly 97% does not violate any of the constraints. In contrast, comparable results are obtained only by CBiRRT [11], which reaches the goal in 86.5% of cases, and only about 66% are valid. However, this method plans motions that are almost 3 times longer, and it takes over 19 times longer to compute them (in terms of median values). The last plot shows the error of maintaining the object in an upward position, which we computed as an integral along the trajectory of the sum of the absolute values of roll and pitch angles. The smallest deviation from the orientation constraint is achieved by our proposed solution, while the highest violations are generated by executing the trajectories planned using SST [41]. The result of the MPC-MPNet [54] deserves special attention, as it is the state-of-the-art learning-based solution for kinodynamic motion planning. We trained it using the plans generated by our planner, however, it was unable even to come close to the results achieved by the TrajOpt [25] and CBiRRT [11], not to mention our proposed planner.

B. Planning High-Speed Hitting in Simulated Air Hockey Game

1) Task Description: In this task, the goal is to score a goal in the game of robotic air hockey from a steady-still puck, i.e., to move the Kuka LBR Iiwa 14 robot handling the mallet from some predefined initial configuration, such that it will reach the puck position with the velocity vector pointing toward the middle of the goal. Robot desired configurations and joint velocities are determined using the optimization algorithm proposed in [8] for a given puck position and velocity direction. Moreover, the planner should generate a joint trajectory minimizing movement time and satisfying joints' velocity, acceleration, and torque constraints. We also impose task space constraints, such that the mallet handled by the robot remains on the table surface and between the bands.

2) Dataset and Method Adjustments: To learn the task, we generated a dataset of 19 800 planning problems of this kind, split into 2 subsets: training (18 000) and validation (1800), while the test set is defined separately. The robot's initial configuration is randomly drawn in a neighborhood of a base configuration, such that the mallet is located in a 10×10 cm box, and its initial velocity and acceleration are set to 0. The puck position is also random, and the hitting direction is computed to approximately point toward the goal. Similarly to the previous task, the dataset used in the training and validation contains only motion planning problems, without precomputed solutions. Moreover, we add an additional constraint manifold loss term that penalizes the displacement of the robot end effector from the table surface, and a task loss term that penalizes the high centrifugal forces at the end effector, to reduce the trajectory tracking errors. The mathematical definition of these losses, together with all parameters of this experiment, can be found in Appendix B2.

3) Quantitative Comparison With State-of-the-Art: We compared our approach with state-of-the-art motion planning algorithms on the set of 1681 hitting scenarios of a puck being located on a 41×41 grid, which were not present in the training and

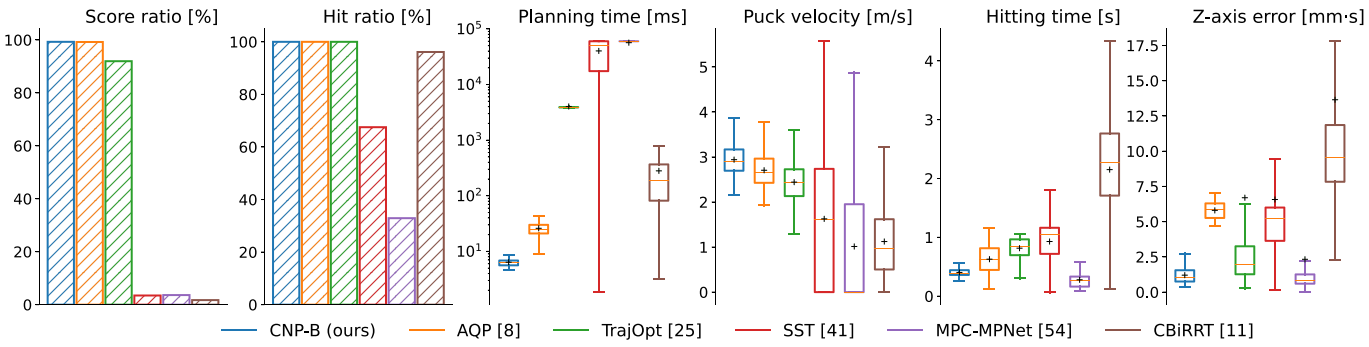


Fig. 5. Planners statistics in the task of hitting in the simulated robotic air hockey.

validation sets. One of the key challenges of this task is that the goal configuration has to be reached with a given high velocity in order to score the goal. It is hard to plan the trajectory that satisfies this kind of constraint, especially using sampling-based motion planners. Therefore for CBiRRT, MPC-MPNet, and SST algorithms, we simplified the task, such that the goal was to hit the puck by reaching the puck position without reaching the desired end velocity direction, i.e., we computed the distance function only for positional coordinates. To train MPC-MPNet, we used the trajectories generated by the CNP-B algorithm.

The results of this comparison are presented in Fig. 5. Even though the task for MPC-MPNet and SST planners is simplified, they cannot plan trajectories within a reasonable time. The only sampling-based algorithm, which is able to reach the target in almost all scenarios is CBiRRT, however, it produces plans that are very hard to follow (see z-axis error chart in Fig. 5). Far better performance is achieved by optimization-based planners (TrajOpt and AQP), which almost always hit the puck and score, respectively, in 91.97% and 99.23% of scenarios. The limitation of TrajOpt is that it needs nearly 4 s on average to compute the plan, whereas the AQP mean planning time is 26 ms. A similar planning time scale is achieved only by our proposed algorithm, which plans within 6.5 ms on average. Moreover, our solution achieves a scoring ratio of 99.28%, plans the fastest trajectories (except for MPC-MPNet, which is faster but rarely hits the puck not to mention scoring), obtains the highest puck velocities, and despite this, it generates the trajectories that are possible to follow with the accumulated z-axis deviation smaller than 3 mm $\cdot$ s. Interestingly, AQP is a state-of-the-art solution tailored specifically to planning for robotic air hockey, and yet its performance is dominated in terms of all considered criteria by the solution trained on automatically generated data using our general framework.

4) Qualitative Results for Replanning: So far, we have quantified how the proposed solution compares to state-of-the-art solutions. However, due to the short and deterministic planning time, and the ability to satisfy boundary conditions, our proposed approach allows for solving tasks that are impossible to solve using state-of-the-art planners, i.e., replanning on the fly. We now consider a situation when the robot is performing some plan, and in the meantime, the goal changes, e.g., puck's expected position or desired hitting direction has changed. For this type of task, the planning time of almost all state-of-the-art methods is too long to react. Moreover, typical motion planning methods do not give any guarantee about the maximal planning time. Unlike these classical approaches, our solution needs a small constant amount of computation to plan the motion. Therefore, we can predict a robot configuration located forward in time along the current trajectory, and plan from this configuration, taking into account the smoothness of the motion and continuity of actuation, by imposing the boundary conditions on the planned motion.

To learn how to plan for a moving robot, we prepared a more general dataset (compared to the one introduced in Section IV-B2) of hitting scenarios, which includes hitting from many different robot configurations in the accessible space, with random initial velocities and accelerations, and random desired hitting directions and velocities. A more detailed description of the used datasets can be found in Appendix B3. In Fig. 6, we show a sequence of frames from the scenario where the robot tries to hit the puck, but after about 300 ms from the beginning of the motion, the puck position changes suddenly. In response to this, the robot replans the trajectory from the point on the actually performed trajectory located a few tens of milliseconds in the future (compensating for the communication times and the fact that the used operating system is not real time) and then waits until the vicinity of this point is reached and switches to the new plan. As shown in Fig. 6, the robot smoothly changes between plans and is able to score the goal with the replanned trajectory.

C. Planning High-Speed Air Hockey Hitting on Real Robot

The most important test of the quality of the proposed solution in robotics is the experimental evaluation on a real robot. This is especially important, because of the well-known problem of the *reality gap*, which is common in systems that use machine learning in simulation or learn from a dataset of simulated examples [60]. To evaluate if this gap exists in our solution, we used exactly the same neural network for the experiments on the real robot as in the simulation, without any additional learning.

1) Quantitative Comparison With State-of-the-Art: Similarly, like in simulation, we perform the test of hitting the puck

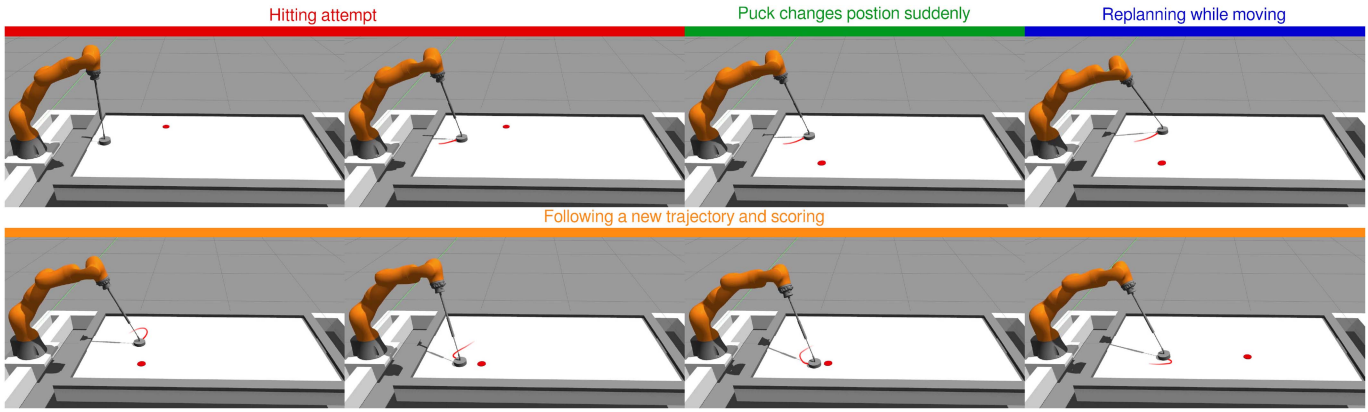


Fig. 6. Sequence of frames form the replanning scenario. The robot starts the hitting motion with the puck located in the upper part of the table, but after 300 ms puck is moved to the lower part. In response to this, CNP-B immediately replans the trajectory and scores the goal.

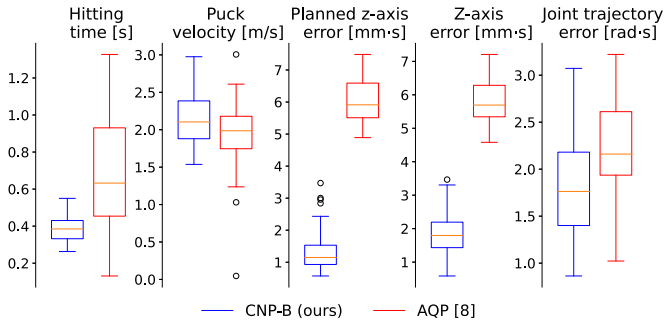


Fig. 7. Planners statistics in the task of hitting in the real robotic air hockey.

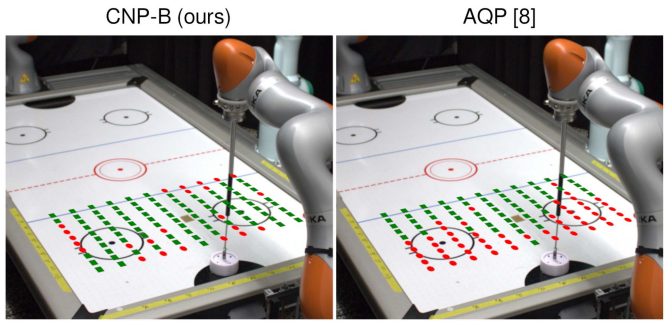


Fig. 8. Visualization of the puck positions in the scenarios on which CNP-B and AQP planners were evaluated. The green squares represent scored goals (success), while the red dots missed shots (failure).

toward the goal, starting from a steady-still manipulator in a base configuration. For each scenario, the puck is placed on 1 of the 110 predefined puck positions (10×11 grid). Experiments in simulation showed us clearly that AQP was the only baseline able to compute safe-to-follow plans in a reasonable time and to compete with the CNP-B. Therefore, in experiments on the real robot, we compared our proposed solution only to the AQP. Statistical comparison is presented in Fig. 7. The hitting movement times correspond with the ones obtained in the simulation, and we can see that CNP-B is characterized by a much smaller mean and variance. The puck velocity magnitude is higher for the proposed solution, due to the fact that AQP method scales down the hitting velocity if it cannot find a feasible plan. The biggest advantage of the proposed planner is visible in terms of the z-axis error, as the generated plans are much closer to the table surface. Also, the trajectory tracking errors are smaller for CNP-B, despite significantly faster trajectories.

Nevertheless, from the task point of view, the most important metric (besides safety) is the ratio of scored goals to all attempts. In this category, CNP-B outperforms AQP, by reaching the ratio of 78.2% compared to 52.7%. In Fig. 8, we present the grid of puck positions and indicate the scored goal from this position with green squares and miss with red dots. It is visible that AQP has problems with executing plans for the puck close to the corners of the table, while CNP-B errors seem not to show any particular correlation with the geometry of the playing field. We hypothesize that the few unsuccessful hits by CNP-B are related to the mechanical setup of the physical system (particularly the mallet and its attachment to the robot), thus illustrating rather the *reality gap* problem.

2) Trick Shots: The ability of the proposed solution to rapidly plan and replan robot motions, which we have shown in simulation, is also easily transferable to the real robot without any further learning. In Fig. 9, we present a sequence of frames from the scenario where the robot starts making feinting movements to confuse the opponent, and after some time, computes the new hitting trajectory, starting from nonzero velocity and acceleration, and scores a goal. This kind of dynamic replanning behavior is possible only because our proposed solution plans within a very short and almost constant time, and is able to plan from nonzero boundary conditions imposed on velocity and acceleration.

D. Ablation Studies

While in Appendix B, we present the parameters of our method that were used to perform all above-described experiments, in this section, we analyze how the changes of these parameters and the experiment's conditions affect the performance of the proposed approach.

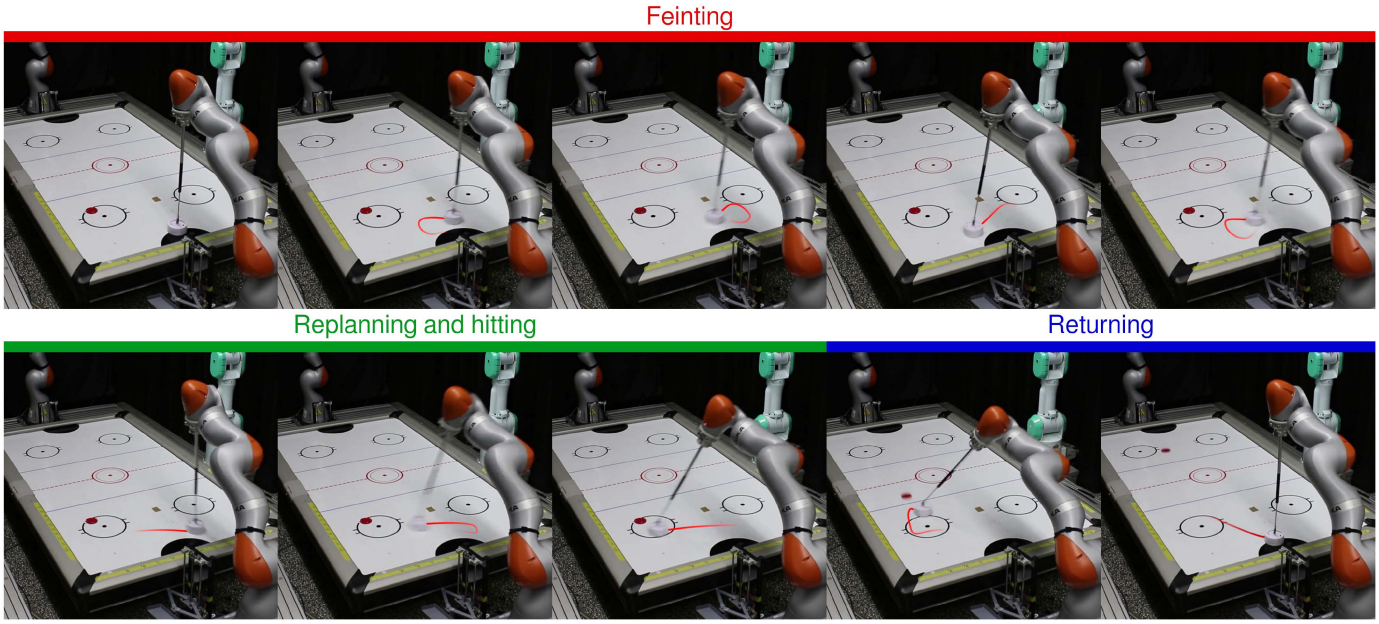


Fig. 9. Fast on the fly motion replanning can be used to smoothly change the robot's behavior from feinting to striking almost instantaneously.

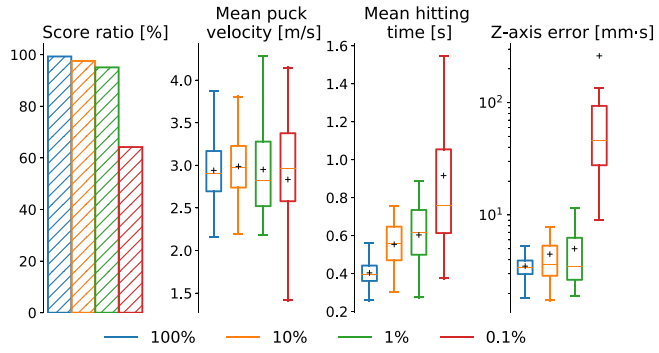


Fig. 10. Performance analysis of CNP-B trained using 100%, 10%, 1%, and 0.1% samples of the training dataset.

1) Training Set Size: The quality of the learned models depends on the data used for training. We quantitatively analyzed the impact of the dataset size on the planning performance in the task of simulated air hockey hitting. In Fig. 10, we present the results achieved by the models trained on fractions of the base dataset introduced in Section IV-B2. As expected, the best results are obtained by the planner trained on the whole dataset. However, relatively similar success ratios and puck velocities are also obtained by the models trained using just 10% and 1% of data. The impact of the reduced number of training planning problems is clearly visible in terms of the hitting times, which are on average about 50% longer for models trained on 10% and 1% of the training set, and more than 2 times longer for the one trained with 0.1% of the training set. Using less data also influences the violations of the table surface constraint. While the median stays almost the same as for the baseline model when using 10% and 1% of the training data, the distribution shifts toward bigger errors. We observe a significant difference in performance only for the model trained on the smallest fraction of training data.

2) Size of the Neural Network: The size of the model is an important parameter of any machine learning solution, as it affects the expressiveness of the model and the amount of computation needed to evaluate it. For the fixed architecture presented in Fig. 2, we changed the number of neurons ν in each layer and analyzed statistically how it impacts the performance in the task of moving a vertically oriented heavy object between boxes. Results of this experiment are presented in Fig. 11. The quantity that is affected the most is the planning time. Reducing the number of neurons results in decreasing the planning time down to about 0.6 ms for 16 neurons. Surprisingly, the timings for 16, 32, and 64 neurons are almost constant. We argue that this may be related to the neural network size fitting the cache of the CPU. The rest of the analyzed measures are almost constant for all considered sizes of neural networks. However, scaling the network down to 16 neurons in each layer causes a significant increase in the motion time and notable growth of the orientation error.

3) Number of Control Points: The number of B-spline control points is one of the key parameters that affect the expressiveness of the solution representation we used. We present the impact of this number on the motion planner performance in the task of simulated air hockey hitting in Fig. 12. We observe that the performance of our approach is robust to the number of control points. However, the general tendency is that increasing the number of control points allows for obtaining slightly higher success ratios and puck velocities but at the same time increases the movement time. In terms of the deviation from the table plane in the z-axis, both a lower and greater number of control points result in increased errors.

4) Generalization Abilities: In all previous experiments, we assumed that the manifold on which we would like to plan is

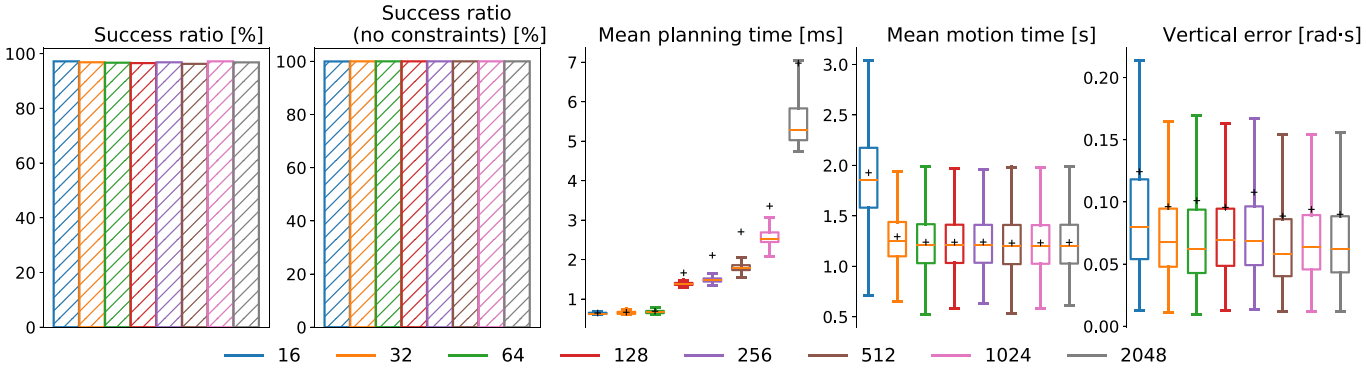


Fig. 11. Analysis of CNP-B performance in the task of moving a heavy vertically oriented object for different neural network sizes, represented by the number of neurons in each layer.

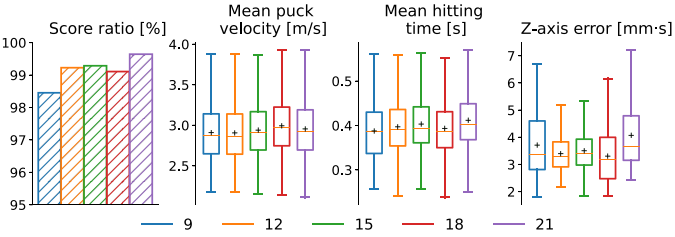


Fig. 12. Analysis of CNP-B performance in simulated air hockey hitting for different numbers of B-spline control points.

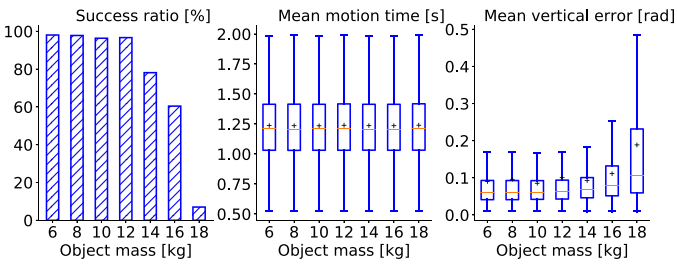


Fig. 13. Analysis of the generalization abilities of CNP-B for different object mass in the task of moving a vertically oriented heavy object.

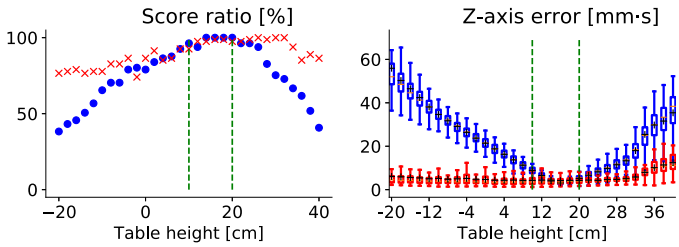


Fig. 14. Analysis of the generalization abilities of CNP-B for different table heights in simulated air hockey hitting task. Blue denotes the model trained only on height equal to 16 cm, while the red one is a parametrized model trained on random table height from the range denoted by dashed green lines.

constant for all considered motion planning problem instances. However, one can imagine a situation where the mass of the object we want to move with the robot is different from the one on which our planner was trained, or the air hockey table height has changed. These cases raise important questions: (i) is the proposed method able to generalize outside the conditions of the training set? (ii) Is it possible to learn how to plan on parametrized manifolds and how it will affect the generalization abilities of the proposed method? To answer these questions, we performed two experiments and present the results in Figs. 13 and 14.

In Fig. 13, we present the performance of the planner, trained to solve a task of moving a vertically oriented heavy object that weighs exactly 12 kg, on moving heavier and lighter ones. As one could expect, there is no problem with handling lighter objects, however, due to the reduced mass the generated trajectories are probably not optimal, as the motion could be executed faster. In turn, heavier objects pose a much harder challenge, as the increased mass easily leads to violations of the maximal torques, since the maximal robot's payload is 14 kg. This strongly affects the success rate by preventing solving some of the tasks. Infeasible torque commands also result in inaccurate trajectory tracking, resulting in collisions with obstacles and higher orientation errors. However, our method still shows reasonable success rates when the task mass is close to the training one.

To further investigate the generalization abilities of the proposed method, and answer the second question regarding manifold parametrization, we performed an experiment of simulated air hockey hitting with variable table height. In Fig. 14, we present scoring ratio and deviations from the table plane on a wide range of the table heights. The blue color denotes the model that was trained only on the table height equal to 16 cm. It generalizes very well for the heights between 14 and 20 cm, but for greater height discrepancies, the z-axis errors grow significantly. Nevertheless, it is still able to score in a noticeable amount of cases. The second part of this experiment is a parametrized model, which takes as an input height of the table and was trained on the tables of heights randomly drawn from the range between 10 and 20 cm (marked with green dashed lines). While the score ratio of this model is similar to the one obtained by the nonparametrized model on this range, we observe a tremendous difference outside this region. The parametrized model achieves very low z-axis errors on all heights between -20 and 30 cm showing broad generalization properties. It also achieves a score

ratio higher than 75% on almost the whole range of tested heights.

E. Discussion

In this section, we want to analyze the strengths and weaknesses of our approach w.r.t. the state-of-the-art planners. Our experiments have shown that our methodology outperforms every state-of-the-art method under all metrics. We believe this performance increase is due to three crucial aspects of our approach:

1) the handling of constraints;
2) the use of flexible trajectory parameterization;
3) the learning setup.

First, we treat the constraints satisfaction problem as loss minimization, instead of a hard constraint that should be satisfied at any time. This treatment allows us to accept small violations both during training and deployment. While in some scenarios, small trajectory violations are a major problem, these violations are acceptable for most practical setups. Unfortunately, our handling of the constraints does not guarantee the planned trajectory will satisfy all the constraints. However, it is fairly easy to verify that a planned trajectory complies with the requirements and abort the plan if necessary. State-of-the-art planners handle constraints in various ways, which we described in Section II, however, none of them is perfect. The SST and MPC-MPNet algorithms satisfy the robot kinodynamic constraints by relying on the control space sampling. However, for systems with many degrees of freedom, it leads to slow planning. Moreover, sampling-based planners have innate problems when other task-related constraints are present, as they are represented as obstacles that form narrow passages, which results in a further slowdown. In response to this problem, the CBiRRT algorithm was introduced, ensuring geometrical constraint satisfaction using projection on the constraint manifold. Unfortunately, this planner cannot properly handle the kinodynamic constraints in dynamic motion because it assumes motion to be quasi-static. In turn, optimization-based methods can handle constraints very well, but if antagonistic constraints are present, they may cause the optimization to get stuck at a local minimum.

Second, thanks to our B-spline parameterization, we can easily ensure that the plans will always start from the current state and reach the planned one in every trajectory computed by the network. While this property could cause issues in the learning process and plan generation, our B-spline parametrization decouples the geometrical path and the speed of execution by learning a spline for the time coordinate. This decoupling gives us great flexibility and simplifies the learning process. Using parametrized trajectories simplifies the structure of the neural network, avoiding complex learning procedures required for recurrent networks. Also, it is easy to tune the number of control points of the spline to control the maximum allowed complexity of the path.

Finally, our learning setup is simple, as it does not require any expert dataset besides the models of the robot and environment. Furthermore, it is easy to generalize to different tasks and settings. In principle, if we provide a sufficiently large training set, this approach could process any relevant information to solve the task, allowing for advanced obstacle avoidance and maneuvering, e.g., as done in [15]. However, there is no practical way to guarantee that the generated plans will not violate any of the constraints. This lack of guarantees can be potentially seen as a drawback w.r.t. other methods, as they can adapt on the fly and optimize the trajectory, ensuring that the solution found is feasible. However, it is always possible to improve the trajectory generated by our method with further optimization, e.g., using the TrajOpt framework. Further optimization of the generated trajectory is feasible because the planning time of the proposed method is negligible compared to basic trajectory optimization. Another important gain from the short planning time, combined with a constant amount of computation and boundary conditions satisfaction thanks to B-spline trajectory representation, is the ability to feasible and smooth replanning on the fly. To the best of authors' knowledge, it is the first time this kind of dynamic smooth replanning is presented for problems of this complexity, along with a demonstration on a real robot.

V. Conclusion

In this article, we presented a complete learning solution to kinodynamic planning under arbitrary constraints. Exploiting deep neural networks, we are able to learn a planning function generating plans lying in close vicinity of the constraint manifold. Our approach is easy to implement, computes plans with minimal computational requirements, and can replan from arbitrary configurations. Our method is flexible and can adapt to many different tasks, and it does not require any reference trajectories for training. While we cannot ensure the safety of all the plans generated by the neural network, it is always possible to perform postprocessing steps and discard the plans that do not satisfy the constraints. Our real robot experiment shows that the proposed method can be successfully used in real-world tasks, outperforming existing state-of-the-art solutions. CNP-B can be seen as a solution bridging the gap between a reactive neural controller and a classic motion planning algorithm. It is fast as the reactive neural controller but computes the whole plan at once, like a planning method, which is important from the safety point of view. In future works, we would like to include some more sophisticated representation of the constraint manifold in the input of the neural network to be able to plan on various manifolds without retraining. It would also be interesting to check whether CNP-B can be applied as a motion generator, plugged into a higher level reinforcement learning setup as a parametrizable action, e.g., to learn how to play the whole game of air hockey.

Appendix

A. Satisfying B-Spline Boundary Constraints

To satisfy the boundary constraints of the problem using our B-spline trajectory, we define the vector of knots $\boldsymbol{k}$ by

$$\boldsymbol{k} = \Bigg[\underbrace{0 \quad \dots \quad 0}_{} \quad \underbrace{\frac{1}{C-D} \quad \dots \quad \frac{C-D-1}{C-D}}_{} \quad \underbrace{1 \quad \dots \quad 1}_{}\Bigg]$$

where D is the degree of the B-spline and C is the number of the B-spline control points. This design of the knots vector ensures that the resultant B-spline value on the boundaries will be defined directly by the first and last control points. Thus, we can introduce formulas to calculate the first three and last two control points $\mathcal{C}^{\boldsymbol{p}}$ based on the $r(s)$ control points and reaching task constraints

$$\begin{aligned}
\mathcal{C}^{\boldsymbol{p}}_1 &= \boldsymbol{q}_0 \\
\mathcal{C}^{\boldsymbol{p}}_2 &= \mathcal{C}^{\boldsymbol{p}}_1 + \frac{\dot{\boldsymbol{q}}_0}{\mathcal{C}^r_1 \eta_{\boldsymbol{p}}} \\
K &= \frac{\ddot{\boldsymbol{q}}_0 - \eta_{\boldsymbol{p}}\eta_r(\mathcal{C}^{\boldsymbol{p}}_2 - \mathcal{C}^{\boldsymbol{p}}_1)(\mathcal{C}^r_2 - \mathcal{C}^r_1)\mathcal{C}^r_1}{(\mathcal{C}^r_1)^2} \\
\mathcal{C}^{\boldsymbol{p}}_3 &= \frac{K - 2\beta_{\boldsymbol{p}}\mathcal{C}^{\boldsymbol{p}}_1 + 3\beta_{\boldsymbol{p}}\mathcal{C}^{\boldsymbol{p}}_2}{\beta_{\boldsymbol{p}}} \\
\mathcal{C}^{\boldsymbol{p}}_{C_{\boldsymbol{p}}} &= \boldsymbol{q}_d \\
\mathcal{C}^{\boldsymbol{p}}_{C_{\boldsymbol{p}}-1} &= \mathcal{C}^{\boldsymbol{p}}_{C_{\boldsymbol{p}}} - \frac{\dot{\boldsymbol{q}}_d}{\mathcal{C}^r_{C_r}\eta_{\boldsymbol{p}}}
\end{aligned} \tag{12}$$

where $\eta_{\boldsymbol{p}}$, $\beta_{\boldsymbol{p}}$, and η_r are defined by

$$\begin{aligned}
\eta_{\boldsymbol{p}} &= D_{\boldsymbol{p}}(C_{\boldsymbol{p}} - D_{\boldsymbol{p}})^2 \\
\beta_{\boldsymbol{p}} &= \frac{D_{\boldsymbol{p}}(D_{\boldsymbol{p}} - 1)}{2}(C_{\boldsymbol{p}} - D_{\boldsymbol{p}})^2 \\
\eta_r &= D_r(C_r - D_r)^2
\end{aligned}$$

where $D_{\boldsymbol{p}}$ and D_r are the degrees of the $\boldsymbol{p}(s)$ and $r(s)$ B-splines, respectively, and $C_{\boldsymbol{p}}$ and C_r are the numbers of their control points.

B. Experimental Evaluation Details

1) Heavy Object Manipulation:

a) Data preparation: The data generation process for this experiment was done in the following way:

1) draw random initial and desired position of the heavy object;
2) assume that the pedestal boxes have fixed dimensions and are located just beneath the objects;
3) draw initial guess configuration of the robot;
4) starting from this configuration, optimize the robot's initial configuration, such that its end-effector position matches the initial position of the heavy object, and the orientation is vertical;
5) validate if the robot in the initial configuration does not collide with the environment and if it does not violate the torque constraints;
6) starting from the initial configuration, optimize the robot desired configuration, such that its end-effector position matches the desired position of the heavy object;
7) validate if the robot in the desired configuration does not collide with the environment and if it does not violate the torque constraints.

The specific parameters values and ranges are shown in Table I, where z_0, z_d represents the object's initial and desired position along z-axis, whereas $o_h = 0.15$ m is the fixed height of the object.

TABLE I
PARAMETERS OF THE DATA GENERATION PROCEDURE FOR HEAVY OBJECT MANIPULATION TASK

Parameter	Value
Initial object position	$(x_0, y_0, z_0) \in [0.2; 0.6] \times [-0.6; -0.3] \times [0.2; 0.5]$
Desired object position	$(x_d, y_d, z_d) \in [0.2; 0.6] \times [0.3; 0.6] \times [0.2; 0.5]$
Pedestal 1	$\{(x, y, z) \mid 0.2 \le x \le 0.6, -0.6 \le y \le -0.3, z \le z_0 - o_h\}$
Pedestal 2	$\{(x, y, z) \mid 0.2 \le x \le 0.6, 0.3 \le y \le 0.6, z \le z_d - o_h\}$
Initial guess robot configuration	$q \in [-\frac{\pi}{2}; \frac{\pi}{2}] \times [0; \frac{\pi}{2}] \times [-\frac{\pi}{2}; \frac{\pi}{2}] \times [\frac{\pi}{2}; 0] \times [-\frac{\pi}{2}; \frac{\pi}{2}]^2 \times [-\pi; \pi]$

b) Additional loss functions: As we already mentioned in the main text, in this task, to meet the constraints imposed on this task, we introduced 3 additional loss terms, i.e., vertical orientation loss defined by

$$\mathcal{L}^O_{\mathcal{M}_t}(s) = \mathrm{H}(1 - \mathrm{R}_{2,2}(\boldsymbol{q}(s)))$$

where $\mathrm{R}_{2,2}$ is the element of the end-effector rotation matrix with an index of $(2, 2)$, robot collision loss defined by

$$\mathcal{L}^{E_r}_{\mathcal{M}_t}(s) = \mathrm{H}\left(\sum_{p \in \mathrm{FK}_{kc}(\boldsymbol{q}(s))} \mathrm{ReLU}(0.15 - d(p, E))\right)$$

where E represents the set of the collision objects in the environment (pedestals), FK_{kc} is a set of points in the workspace located along the kinematic chain (representation of the robot geometry), and $d(X, Y)$ is a Euclidean distance between X and Y, and finally, object collision loss

$$\mathcal{L}^{E_o}_{\mathcal{M}_t}(s) = \mathrm{H}\left(\sum_{p \in \mathrm{FK}_o(\boldsymbol{q}(s))} \mathrm{ReLU}(d(p, E) \cdot \mathrm{I}(p, E))\right)$$

where FK_o represents the set of points that belong to the handled object and $\mathrm{I}(X, Y)$ is an indicator function, which is equal to 1 if $X \in Y$ and 0 otherwise. In our experiments, we defined environment E as two cuboids defined in Table I. The heavy object handled by the robot is a cuboid with dimensions $0.2 \times 0.2 \times 0.3$ m, which for collision-checking purposes is represented by its corners. The robot itself is represented by the positions of the joints in the workspace and points linearly interpolated between them, such that no point lies further than 10 cm from its neighbors. We assume that there is no collision if the obstacles are at least 0.15 m away from the robot representation.

TABLE II
PARAMETERS OF THE DATA GENERATION PROCEDURE FOR AIR HOCKEY HITTING TASK

Parameter	Value
Initial mallet position	$(x, y, z) \in [0.6; 0.7] \times [-0.05; 0.05] \times [0.155; 0.165]$
Desired mallet position	$(x, y, z) \in [0.65; 1.3] \times [-0.45; 0.45] \times \{0.16\}$
Base robot configuration	$q_0 = [0 \quad 0.697 \quad 0 \quad -0.505 \quad 0 \quad 1.93]$

2) Simulated Air Hockey Hitting

a) Data preparation: The data generation process for this experiment was done in the following way:

1) draw random initial and desired position of the mallet, such that they are at least 10 cm apart;
2) starting from the base configuration, optimize the robot's initial configuration, such that its end-effector position matches the initial position of the mallet;
3) using the desired mallet position and goal position, define the desired hitting angle, and add noise to it;
4) compute the desired joint velocity of the robot that maximizes the manipulability along the hitting direction [8];
5) in half of the cases randomly scale the magnitude of the desired joint velocity, and set its magnitude to the maximum in the other half;
6) validate the possibility of performing the hit, by analyzing if it is possible to avoid a collision after the hit, i.e. if the point defined by $p_h = p_d + v_h \cdot 50$ ms lies between the table bands, where p_d is the desired hitting point, and v_h is the hitting velocity in the workspace.

The specific parameter values and ranges are shown in Table II.

b) Additional loss functions: As we already mentioned in the main text, to meet the constraints imposed on this task, we introduced an additional constraint manifold loss term, which is responsible for maintaining the mallet position on the table surface. We define this loss function as the sum of the losses in x, y, z directions

$$\mathcal{L}^{\mathcal{T}}_{\mathcal{M}_t}(s) = \mathcal{L}^{\mathcal{T}_x}_{\mathcal{M}_t}(s) + \mathcal{L}^{\mathcal{T}_y}_{\mathcal{M}_t}(s) + \mathcal{L}^{\mathcal{T}_z}_{\mathcal{M}_t}(s)$$

where specific losses are defined by

$$\begin{aligned}
\mathcal{L}^{\mathcal{T}_x}_{\mathcal{M}_t}(s) = {} & \mathrm{H}(\mathrm{ReLU}(\mathcal{T}_{\underline{x}} - \mathrm{FK}_x(\boldsymbol{q})) \\
& + \mathrm{H}(\mathrm{ReLU}(\mathrm{FK}_x(\boldsymbol{q}) - \mathcal{T}_{\overline{x}}) \\
\mathcal{L}^{\mathcal{T}_y}_{\mathcal{M}_t}(s) = {} & \mathrm{H}(\mathrm{ReLU}(\mathcal{T}_{\underline{y}} - \mathrm{FK}_y(\boldsymbol{q})) \\
& + \mathrm{H}(\mathrm{ReLU}(\mathrm{FK}_y(\boldsymbol{q}) - \mathcal{T}_{\overline{y}}) \\
\mathcal{L}^{\mathcal{T}_z}_{\mathcal{M}_t}(s) = {} & \mathrm{H}(FK_z(\boldsymbol{q}) - \mathcal{T}_z)
\end{aligned}$$

where $\mathcal{T}_{\underline{x}}$, $\mathcal{T}_{\overline{x}}$, $\mathcal{T}_{\underline{y}}$, and $\mathcal{T}_{\overline{y}}$ are the lower and upper boundaries of the table in the x and y directions, while $\mathcal{T}_z$ is the table surface height.

Moreover, we observed that the robot controller has problems with tracking very fast trajectories, especially when trajectory curvature in the workspace is high. Thus, we introduced an additional regularization term to the typical time-minimization task loss and defined the task loss function by

$$L_t(s) = 1 + \eta \kappa_{ee}(s) v^2_{ee}(s)$$

where $\eta = 0.01$ is an experimentally chosen regularization factor, while κ_{ee} and v_{ee} are, respectively, the curvature and velocity of the end-effector trajectory.

3) Dataset for Air Hockey Hitting Replanning

The dataset for learning how to hit and be able to replan is similar to the one created just for the hitting task, however, it is far more diversified. To be able to replan, we need to know how to plan between any two configurations in the workspace. Moreover, to be able to perform different trick shots, we also randomized the desired hitting direction, initial velocity, and acceleration. The data generation procedure scheme is similar to the one shown in Appendix B2 and differs only in the following steps:

1) initial and desired mallet positions range defined by $(x, y, z) \in [0.6; 1.3] \times [-0.45; 0.45] \times \{0.16\}$;
2) draw a hitting angle which differs from the direction of the line connecting initial and desired positions no more than $\frac{2}{3}\pi$;
3) in 80% of the cases randomly scale the magnitude of the desired joint velocity, and set to maximal in the rest;
4) compute random initial joint velocity constrained to the table manifold, or set it to 0 in 20% of cases;
5) compute random initial joint acceleration constrained to the table manifold.

The dataset consists of 120 000 samples, from which 112 000 belong to the training set and the rest to the validation set.

4) Parameters of the Algorithms Used for Evaluation

We compared our proposed approach with several state-of-the-art motion planning algorithms, i.e., TrajOpt [25], MPC-MPNet [54], CBiRRT [11], and SST [41]. As we aim to increase the reproducibility of our research, none of these methods was implemented by ourselves, instead, we relied on the following implementations:

1) for TrajOpt, we employed the SLSQP minimization implemented in SciPy, with constraints and cost function implemented by us using pinochchio [57] library;
2) for MPC-MPNet and SST, we used the implementation provided by Li et al. [54] with some necessary modifications required to compile and run their code, and our C++ implementation of the heavy object manipulation and robotic air hockey systems, which also utilizes the Pinocchio library [57];
3) for CBiRRT, we used the OMPL [61] implementation and our own constraint and distance definitions.

To make our comparison fair we tried to adjust the parameters of these motion planning algorithms, to maximize their performance. Appropriate parameter tuning is especially important for

the sampling-based motion planning algorithms, therefore, we performed a random search of the parameters to find the regions of the parameter space, which give the highest success ratios and shortest planning times. In the case of the SST and MPC-MPNet, some of their parameters are common because they use the same algorithm under the hood, the difference is in the way of selecting the node to expand and in the expand procedure itself. Therefore, we set for both algorithms the same parameters for node search radius $\delta_{\text{BN}} = 0.2$ m, witness radius $\delta_s = 0.1$ m, and goal radius $r_g = 0.2$ m, for the task of heavy object manipulation, and $\delta_{\text{BN}} = 0.05$ m, witness radius $\delta_s = 0.02$ m and goal radius $r_g = 0.02$ m for robotic air hockey hitting. Regarding the extend procedure, for both tasks, we set the minimum and the maximum number of steps to 1 and 20, respectively, with an integration step equal 5 ms for SST, whereas for MPC-MPNet we set the parameters of the CEM MPC solver to 64 samples, 4 elite samples, the maximal number of iterations equal 30, and convergence radius equal 0.02, as they resulted in fast convergence. We have also set the values of the integration step to 10 ms, motion time Gaussian $(\mu_t, \sigma_t) = (0.05, 0.1)$, with maximal time $t_{\max} = 0.1$ s, control Gaussian $(\mu_c, \sigma_c) = (\mathbf{0}, 0.8\bar{\boldsymbol{\tau}})$, for the heavy object manipulation task, for MPC-MPNet, while for air hockey $(\mu_t, \sigma_t) = (0.1, 0.4), t_{\max} = 0.5\,s, (\mu_c, \sigma_c) = (\mathbf{0}, 0.5\bar{\boldsymbol{\tau}})$. Nevertheless, despite the tuning of the parameters, obtained motion planning times were relatively long. This may be caused by the innate difficulty of the considered problem and very tight constraints, which can be viewed as narrow passages in the configuration space. For both SST and MPC-MPNet, we used the Euclidean distance function in the task space. This simplifies the motion planning problem a lot, as it requires the planners to plan only for the position, disregarding the task of reaching the desired velocity. It is also possible to define a distance function that takes into account the distance in the space of the velocities weighted with the distance in the task space. However, this makes the exploration of the states-space inefficient, further slowing down the planning process. Another very important parameter of the MPC-MPNet is the training data. The MPC-MPNet relies on the behavior cloning, therefore, we trained it on the solutions generated by AQP in the air hockey task, and by our proposed method for lifting a heavy object. For CBiRRT, for both tasks, we set the goal radius to 1 cm, and the allowed tolerance of the constraint to 0.01, while for the range parameter, we used an automatically determined value by OMPL. In the case of the TrajOpt implementation in Scipy, there are no parameters that affect the performance of the method, so we only limited the maximum number of iterations to 100, to avoid prolonged optimization.

Our proposed method also has some important parameters that have to be chosen. For our experiments, we set the number of control points of the $\boldsymbol{p}(s)$ and $r(s)$ B-splines to 15 and 20, respectively, as it gives enough freedom to plan accurate trajectories that satisfy constraints. To ensure a high level of trajectory smoothness we set the degree of both B-splines to 7, and the number of neurons in each layer of the proposed neural network to 2048. Another group of parameters is the one related to the manifold metric optimization, i.e., $\boldsymbol{\alpha}^{(0)}$. For the heavy object manipulation $\boldsymbol{\alpha}^{(0)}$ was set to the zero vector, while for the air hockey task, we proposed a different initialization, which was meant to equalize the initial gradients of loss functions and was equal to $\boldsymbol{\alpha}^{(0)} = \begin{bmatrix}\alpha_{\mathcal{T}}^{(0)} & \alpha_{\dot{\boldsymbol{q}}}^{(0)} & \alpha_{\ddot{\boldsymbol{q}}}^{(0)} & \alpha_{\boldsymbol{\tau}}^{(0)}\end{bmatrix} = \begin{bmatrix}1 & 1 & 10^{-2} & 10^{-4}\end{bmatrix}$. For the heavy object manipulation task, we defined the following levels of the allowed constraint violations $\bar{C}_E = 10^{-6}$, $\bar{C}_O = 10^{-5}$, $\bar{C}_{\dot{\boldsymbol{q}}} = 6 \cdot 10^{-3}$, $\bar{C}_{\ddot{\boldsymbol{q}}} = \bar{C}_{\boldsymbol{\tau}} = 6 \cdot 10^{-2}$, while for the hitting task $\bar{C}_{\mathcal{T}} = 2 \cdot 10^{-6}$, $\bar{C}_{\dot{\boldsymbol{q}}} = 6 \cdot 10^{-3}$, $\bar{C}_{\ddot{\boldsymbol{q}}} = 6 \cdot 10^{-2}$, $\bar{C}_{\boldsymbol{\tau}} = 6 \cdot 10^{-1}$. The update step for the α parameters was set to 10^{-2} for both motion planning problems. The learning step for neural network parameters was set to $5 \cdot 10^{-5}$, while the batch size was set to 128.

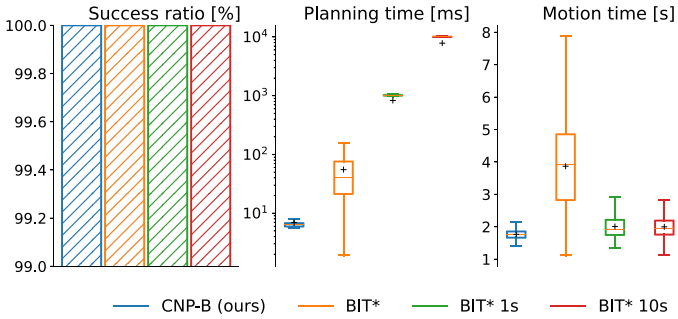


Fig. 15. Comparison of the proposed planner with BIT* in the task of obstacle avoidance. To make the comparison fair we compare with 3 versions of the BIT* algorithm: BIT*—returns the first solution found, BIT* 1 s—returns the best solution found within 1 s, BIT* 10 s—returns the best solution found within 10 s.

C. Obstacle Avoidance

To further evaluate the obstacle avoidance abilities of the proposed planner we prepared another experiment in which the KUKA Iiwa 14 arm was meant to plan a minimal-time trajectory between two random configurations while avoiding the collision with two randomly placed balls of random radiuses. Initial and desired configurations of the robot were drawn from the hyper-rectangles with the centers in $q_{0c} = \begin{bmatrix}-1 & 1.2 & 1 & 0 & 1 & 1\end{bmatrix}$ and $q_{\text{dc}} = \begin{bmatrix}1 & 1.2 & 1 & 1 & 1 & 1\end{bmatrix}$, and width of 0.4 rad in each axis. Obstacles were defined as 2 balls of the radius randomly drawn from the range [10; 20] cm and placed randomly within a hyper-rectangle located between the robot's initial and desired configuration sets, i.e., centers of obstacles located inside following set $(x_o, y_o, z_o) \in [0.25; 0.55] \times [-0.2; 0.2] \times [0.15; 1.0]$, expressed in meters. The body of the robot was also represented with balls, i.e., a sequence of 22 balls with 12 cm radius, located along the kinematic chain. To increase the difficulty of these problems, we enforced that the space between obstacles cannot be greater than the robot radius tripled and smaller than the robot diameter increased by 2 cm margin. From these ranges, we sampled all training, validation, and test problems, trained the proposed approach, and compared with SotA motion planner BIT* [23], as we neither have to take into account kinodynamic nor geometrical constraints except for the obstacles. The results of this comparison are presented in Fig. 15. Both CNP-B and BIT* correctly solve all planning problems, however, the BIT* version that works in a comparable time scale

TABLE III
STATISTICAL ANALYSIS OF THE RESULTS OF THE OBJECT MOVING TASK

Criterion	Test	Compared algorithm	Statistic	p-value
Success ratio (w/ constraint)	M	CBiRRT	602	$< 10^{-132}$
Success ratio (w/o constraint)	M	CBiRRT	324	$< 10^{-71}$
Planning time	W	CBiRRT	34047	0
	S	CBiRRT	11.76	$< 10^{-30}$
Motion time	W	TrajOpt	152474.5	$< 10^{-95}$
	S	MPC-MPNet	3.45	0.0003
Motion time	W	MPC-MPNet	532128	$< 10^{-134}$
	S	MPC-MPNet	26.34	$< 10^{-133}$
Vertical error	W	TrajOpt	219055	$< 10^{-57}$
	S	CBiRRT	2.83	0.0023

M — McNemar's test, W — Wilcoxon test, S — Student's t-test

TABLE IV
STATISTICAL ANALYSIS OF THE RESULTS OF THE SIMULATED AIR HOCKEY HITTING

Criterion	Test	Compared algorithm	Statistic	p-value
Scoring ratio	M	AQP	0.04	0.84
	M	TrajOpt	102.9	$< 10^{-23}$
Planning time	W	AQP	376	$< 10^{-275}$
	S	AQP	97.5	0
Puck velocity	W	AQP	1405227	$< 10^{-269}$
	S	AQP	78.4	0
Hitting time	W	AQP	75589	$< 10^{-220}$
	S	AQP	46.9	$< 10^{-307}$
Hitting time	W	MPC-MPNet	113075	$< 10^{-47}$
	S	MPC-MPNet	11.5	$< 10^{-27}$
Z-axis error	W	AQP	150074	$< 10^{-171}$
	S	AQP	1.32	0.09
Z-axis error	W	MPC-MPNet	888872	$< 10^{-19}$
	S	MPC-MPNet	5.81	$< 10^{-8}$

TABLE V
STATISTICAL ANALYSIS OF THE RESULTS OF THE SIMULATED AIR HOCKEY HITTING

Criterion	Test	Statistic	p-value
Scoring ratio	M	7.67	0.006
Puck velocity	W	1119	$< 10^{-7}$
	S	5.08	$< 10^{-6}$
Planned z-axis error	W	5460	$< 10^{-18}$
	S	74	$< 10^{-90}$
Z-axis error	W	5460	$< 10^{-18}$
	S	71.1	$< 10^{-88}$
Joint trajectory error	W	5460	$< 10^{-18}$
	S	28.4	$< 10^{-50}$

to CNP-B returns very long solutions. Increasing the time for computing the solution for BIT*, results in the reduction of the motion times, however, still CNP-B generates plans that are on average faster.

D. Statistical Significance of the Achieved Results

In the article, we have graphically presented the results of the experiments using boxplots showing the data distribution for several performance metrics. In this section, we would like to further analyze these results in terms of statistical significance. In all conducted experiments, we have performed tests on the same randomly chosen set of motion planning problems. Thus, the repeated measure experiment design applies [62]. In most of the cases, significant deviations from the normal distributions were observed, which are caused by the highly nonlinear mapping between the task descriptions, which were drawn randomly using a uniform distribution, and plans returned by the planning algorithms. Therefore, in our statistical analysis, we will rely on the nonparametric tests for the repeated measures design. We test our hypothesis against a 99% confidence level, such that the significance level $\alpha = 0.01$. For the hypothesis testing, we employed two python libraries *statsmodels* [63] for the McNemar's test [64], and *scipy.stats* [65] for the Wilcoxon matched-pairs signed rank test [66]. To complement our analysis, we also use the paired Student's t-test [67] (from *scipy.stats*) to compare not only medians, which are the object of the Wilcoxon test but also the mean values of the considered criteria obtained for different planners. We are aware that for most of the considered cases, we do not meet the t-test assumptions. However, this test is well known for its robustness to mild assumptions violations [68].

Although we compared many planning algorithms in this article, for the statistical analysis, we select only those planners that performed best in terms of the considered criterion, and we compare them with CNP-B. The results of this analysis can be found in Tables III, IV, and V. For the analysis of success ratios, we used McNemar's test (M) for the equality of the success probability [64]. The null hypothesis of this test is that both methods achieved the same ratio of success, while the alternative hypothesis is that their ratios of success are different. In turn, for the analysis of the remaining criteria, we used both Wilcoxon matched-pairs signed rank test (W) [66] and paired Student's t-test (S) [67]. For the Wilcoxon test, the null hypothesis is that the difference between the median values of the considered criterion achieved by both methods is zero, while the alternative hypothesis is that this difference favors our method, e.g., smaller planning time or smaller error. Analogous hypotheses can be formulated for the Student's t-test, but in this case, we compare means instead of medians. Only in the case of comparison between CNP-B and MPC-MPNet in terms of hitting time and z-axis errors in the task of simulated air hockey hitting, for which MPC-MPNet achieved better results, the alternative hypothesis is modified, such that the difference is in favor of MPC-MPNet.

The obtained results, in all the cases except two, allow us to reject the null hypothesis and accept the alternative one, typically with a great margin of the p-value. The only exceptions are obtained in the task of simulated air hockey. The first one is the lack of the statistically significant difference between the scoring ratio of CNP-B and AQP, while the second one is the result of the Student's t-test for the z-axis error, which we presume is caused by the outliers in the data, as the result of the Wilcoxon test for medians is confirming that CNP-B achieved significantly smaller z-axis errors than AQP. The remaining test confirms that the proposed approach outperforms all baselines in terms of almost all criteria, except the hitting time, and z-axis errors in the task of simulated air hockey hitting, which are smaller for

MPC-MPNet. However, with this approach, one cannot plan the movement that allows scoring the goal.

REFERENCES

[1] M. Kawato, F. Gandolfo, H. Gomi, and Y. Wada, "Teaching by showing in kendama based on optimization principle," in *Proc. Int. Conf. Artif. Neural Netw.*, 1994, pp. 601–606.

[2] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Adv. Neural Inf. Process. Syst.*, vol. 21, pp. 849–856, 2008.

[3] K. Mülling, J. Kober, and J. Peters, "A biomimetic approach to robot table tennis," *Adaptive Behav.*, vol. 19, no. 5, pp. 359–376, 2011.

[4] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters, "Learning to play table tennis from scratch using muscular robots," *IEEE Trans. Robot.*, vol. 38, no. 6, pp. 3850–3860, Dec. 2022.

[5] K. Ploeger, M. Lutter, and J. Peters, "High acceleration reinforcement learning for real-world juggling with binary rewards," in *Proc. Conf. Robot Learn.*, 2021, pp. 642–653.

[6] F. von Drigalski, D. Joshi, T. Murooka, K. Tanaka, M. Hamaya, and Y. Ijiri, "An analytical diabolo model for robotic learning and control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 4055–4061.

[7] A. Namiki, S. Matsushita, T. Ozeki, and K. Nonami, "Hierarchical processing architecture for an air-hockey robot system," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 1187–1192.

[8] P. Liu, D. Tateo, H. Bou-Ammar, and J. Peters, "Efficient and reactive planning for high speed robot air hockey," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 586–593.

[9] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 1, no. 1, pp. 159–185, 2018.

[10] Z. Kingston, M. Moll, and L. E. Kavraki, "Exploring implicit spaces for constrained sampling-based planning," *Int. J. Robot. Res.*, vol. 38, no. 10/11, pp. 1151–1178, 2019.

[11] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 625–632.

[12] R. Bordalba, L. Ros, and J. M. Porta, "A randomized kinodynamic planner for closed-chain robotic systems," *IEEE Trans. Robot.*, vol. 37, no. 1, pp. 99–115, Feb. 2021.

[13] M. Xie and F. Dellaert, "Batch and incremental kinodynamic motion planning using dynamic factor graphs," 2020. [Online]. Available: https://arxiv.org/abs/2005.12514

[14] P. Liu, D. Tateo, H. B. Ammar, and J. Peters, "Robot reinforcement learning on the constraint manifold," in *Proc. 5th Annu. Conf. Robot Learn.*, 2022, pp. 1357–1366. [Online]. Available: https://openreview.net/forum?id=zwo1-MdMl1P

[15] P. Kicki and P. Skrzypczyński, "Speeding up deep neural network-based planning of local car maneuvers via efficient B-spline path construction," in *Proc. Int. Conf. Robot. Autom.*, 2022, pp. 4422–4428.

[16] P. Kicki, T. Gawron, K. Ćwian, M. Ozay, and P. Skrzypczyński, "Learning from experience for rapid generation of local car maneuvers," *Eng. Appl. Artif. Intell.*, vol. 105, 2021, Art. no. 104399.

[17] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[18] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Comput. Sci. Dept.*, Iowa State Univ., Oct. 1998. [Online]. Available: http://janowiec.cs.iastate.edu/papers/rrt.ps

[19] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[20] J. A. Starek, J. V. Gomez, E. Schmerling, L. Janson, L. Moreno, and M. Pavone, "An asymptotically-optimal sampling-based algorithm for bi-directional motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 2072–2078.

[21] J. D. Gammell, S.S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 3067–3074.

[22] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5041–5047.

[23] J. D. Gammell, T. D. Barfoot, and S.S. Srinivasa, "Batch informed trees (BIT*): Informed asymptotically optimal anytime search," *Int. J. Robot. Res.*, vol. 39, no. 5, pp. 543–567, 2020.

[24] M. Zucker et al., "CHOMP: Covariant hamiltonian optimization for motion planning," *Int. J. Robot. Res.*, vol. 32, no. 9/10, pp. 1164–1193, 2013.

[25] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," *Robot.: Sci. Syst. IX*, 2013. [Online]. Available: https://www.roboticsproceedings.org/rss09/p31.html

[26] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time Gaussian process motion planning via probabilistic inference," *Int. J. Robot. Res.*, vol. 37, no. 11, pp. 1319–1340, Sep. 2018. [Online]. Available: http://dx.doi.org/10.1177/0278364918790369

[27] R. Bonalli, A. Cauligi, A. Bylard, T. Lew, and M. Pavone, "Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach," in *Proc. Robot.: Sci. Syst.*, Freiburg, Germany, 2019. [Online]. Available: https://www.roboticsproceedings.org/rss15/p78.html

[28] A. D. Dragan, N. D. Ratliff, and S.S. Srinivasa, "Manipulation planning with goal sets using constrained trajectory optimization," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 4582–4588.

[29] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A fast solver for constrained trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 7674–7679.

[30] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *Proc. 1st Int. Conf. Informat. Control, Autom. Robot.*, 2004, vol. 1, pp. 222–229.

[31] M. Bonilla, E. Farnioli, L. Pallottino, and A. Bicchi, "Sample-based motion planning for robot manipulators with closed kinematic chains," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 2522–2527.

[32] M. Bonilla, L. Pallottino, and A. Bicchi, "Noninteracting constrained motion planning and control for robot manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 4038–4043.

[33] J. Szkandera, I. Kolingerová, and M. Maňák, "Narrow passage problem solution for motion planning," in *Computational Science–ICCS 2020*, V. V. Krzhizhanovskaya, Eds. Cham, Switzerland: Springer, 2020, pp. 459–470.

[34] J. Wang, J. Lee, and J. Kim, "Constrained motion planning for robot manipulators using local geometric information," *Adv. Robot.*, vol. 29, no. 24, pp. 1611–1623, 2015.

[35] B. Kim, T. Um, C. Suh, and F. Park, "Tangent bundle RRT: A randomized algorithm for constrained motion planning," *Robotica*, vol. 34, pp. 202–225, 2016.

[36] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Trans. Robot.*, vol. 29, no. 1, pp. 105–117, Feb. 2013.

[37] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2537–2542.

[38] S. Stoneman and R. Lampariello, "Embedding nonlinear optimization in RRT for optimal kinodynamic planning," in *Proc. IEEE 53rd Conf. Decis. Control*, 2014, pp. 3737–3744.

[39] S. Primatesta, A. Osman, and A. Rizzo, "MP-RRT#: A model predictive sampling-based motion planning algorithm for unmanned aircraft systems," *J. Intell. Robot. Syst.*, vol. 103, no. 4, p. 59, Nov. 2021, doi: 10.1007/s10846-021-01501-3. [Online]. Available: https://link.springer.com/article/10.1007/s10846-021-01501-3#citeas

[40] D. Zheng and P. Tsiotras, "Accelerating kinodynamic RRT through dimensionality reduction," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 3674–3680.

[41] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *Int. J. Robot. Res.*, vol. 35, no. 5, pp. 528–564, 2016. [Online]. Available: https://doi.org/10.1177/0278364915614386

[42] M. Cefalo and G. Oriolo, "Dynamically feasible task-constrained motion planning with moving obstacles," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 2045–2050.

[43] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 3671–3678.

[44] C. Zhang, J. Huh, and D. D. Lee, "Learning implicit sampling distributions for motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 3654–3661.

[45] J. Huh and D. D. Lee, "Efficient sampling with Q-learning to guide rapidly exploring random trees," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3868–3875, Oct. 2018.

[46] D. Molina, K. Kumar, and S. Srivastava, "Learn and link: Learning critical regions for efficient planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 10605–10611.

[47] R. Cheng, K. Shankar, and J. W. Burdick, "Learning an optimal sampling distribution for efficient motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 7485–7492.
[48] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Trans. Robot.*, vol. 37, no. 1, pp. 48–66, Feb. 2021.
[49] T. S. Lembono, E. Pignat, J. Jankowski, and S. Calinon, "Learning constrained distributions of robot configurations with generative adversarial network," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 4233–4240, Apr. 2021.
[50] A. H. Qureshi, J. Dong, A. Baig, and M. C. Yip, "Constrained motion planning networks X," *IEEE Trans. Robot.*, vol. 38, no. 2, pp. 868–886, Apr. 2022.
[51] W. J. Wolfslag, M. Bharatheesha, T. M. Moerland, and M. Wisse, "RRT-CoLearn: Towards kinodynamic planning without numerical trajectory optimization," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1655–1662, Jul. 2018.
[52] P. Atreya and J. Biswa, "State supervised steering function for sampling-based kinodynamic planning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2022, pp. 35–43.
[53] M. Yavari, K. Gupta, and M. Mehrandezh, "Lazy steering RRT: An optimal constrained kinodynamic neural network based planner with no in-exploration steering," in *Proc. IEEE 19th Int. Conf. Adv. Robot.*, 2019, pp. 400–407.
[54] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "MPC-MPNet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4496–4503, Jul. 2021.
[55] D. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.
[56] A. Stooke, J. Achiam, and P. Abbeel, "Responsive safety in reinforcement learning by PiD Lagrangian methods," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9133–9143.
[57] J. Carpentier et al., "The Pinocchio C++ library—A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *Proc. IEEE Int. Symp. Syst. Integrations*, 2019, pp. 614–619.
[58] Y. R. Stürz, L. M. Affolter, and R. S. Smith, "Parameter identification of the KUKA LBR IIWA robot including constraints on physical feasibility," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6863–6868, 2017.
[59] R. Y. Rubinstein and D. P. Kroese, *The Cross Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation (Information Science and Statistics)*. Berlin, Germany: Springer-Verlag, 2004.
[60] S. Höfer et al., "Perspectives on sim2real transfer for robotics: A summary of the R:SS 2020 workshop," vol. 12, pp. 1–7, 2020.
[61] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.
[62] C. S. Davis, *Statistical Methods for the Analysis of Repeated Measurements*. Berlin, Germany: Springer, 2002.
[63] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with python," in *Proc. 9th Python Sci. Conf.*, 2010, pp. 92–96.
[64] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, Jun. 1947.
[65] P. Virtanen and et al., "SciPy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, pp. 261–272, 2020.
[66] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bull.*, vol. 1, no. 6, pp. 80–83, 1945.
[67] Student, "The probable error of a mean," *Biometrika*, vol. 6, no. 1, pp. 1–25, 1908.
[68] M. Bland, *An Introduction to Medical Statistics (Oxford Medical Publications)*. London, U.K.: Oxford Univ. Press, 2015.

Piotr Kicki (Member, IEEE) received the B.Eng. and M.Sc. degrees in automatic control and robotics in 2018 and 2019, respectively, from the Poznan University of Technology, Poznań, Poland, where he is currently working toward the Ph.D. degree in robotics with the Institute of Robotics and Machine Intelligence.

His main research interests include robot motion planning and applications of machine learning in robotics.

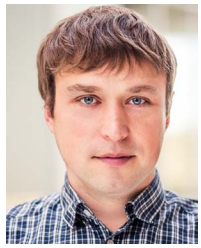
Puze Liu received the B.Sc. degree in engineering mechanics from Tongji University, Shanghai, China, and the M.Sc. degree in computational engineering science from Technical University Berlin, Berlin, Germany. He has been working toward the Ph.D. degree with Intelligent Autonomous Systems Group, Technical University Darmstadt, Darmstadt, Germany, since 2019.

His research interests include interdisciplinary field of robot learning that tries to integrate machine learning techniques into robotics. His prior work focuses on optimization, control, reinforcement learning, and safety in robotics.

Davide Tateo (Member, IEEE) received the M.Sc. degree in computer engineering and the Ph.D. degree in robot learning from Politecnico di Milano, Milan, Italy, in 2014 and 2019, respectively.

He is currently a Research Group Leader with the Intelligent Autonomous Systems Laboratory, Computer Science Department, Technical University of Darmstadt, Darmstadt, Germany. He worked in many areas of robotics and reinforcement learning, including deep reinforcement learning, planning, and perception. His main research interest is robot learning, focusing on high-speed motions, safety, and interpretability. He is one of the coauthors of the MushroomRL Reinforcement Learning library.

Haitham Bou-Ammar recieved the M.Sc. degree in mechatronics engineering from the University of Applied Sciences in Ravensburg-Weingarten, in 2011 and the Ph.D. degree in artificial intelligence from Maastricht University, in 2013. He leads the reinforcement learning team with Huawei Technologies Research and Development, Cambridge, U.K., and is currently an Honorary Lecturer with UCL. His primary research interests lie in the field of statistical machine learning and artificial intelligence, focusing on Bayesian optimization, probabilistic modeling, and reinforcement learning. He is also interested in learning using massive amounts of data over extended time horizons—a property common to "big-data" problems. His research also spans different areas of control theory, nonlinear dynamical systems, social networks, and distributed optimization.



Krzysztof Walas (Member, IEEE) received the M.Sc. degree in automatic control and robotics from the Poznan University of Technology (PUT), Poznań, Poland, in 2007, and the Ph.D. degree (Hons.) in robotics from Poznan University of Technology, in 2012. His Ph.D. dissertation was titled "legged robots' locomotion in structured environments."

He is currently an Assistant Professor with the Institute of Robotics and Machine Intelligence, PUT. His research interests are related to robotic perception for physical interaction applied both to walking and grasping tasks.

Piotr Skrzypczyński (Member, IEEE) received the Ph.D. and D.Sc. degrees in robotics from the Poznan University of Technology (PUT), Poznań, Poland, in 1997 and 2007, respectively.

He is currently a Full Professor with the Institute of Robotics and Machine Intelligence (IRIM), PUT and the Head of the IRIM Robotics Division. He authored or coauthored more than 160 technical papers in robotics and computer science. His current research interests include AI-based robotics, robot navigation and SLAM, computer vision, and machine learning.

Jan Peters (Fellow, IEEE) received the Diplom-Informatiker in computer science from Hagen University, Hagen, Germany, in 2000, the Diplom-Ingenieur in electrical engineering from the Munich University of Technology, Munich, Germany, in 2002, the M.Sc. degrees in computer science, and aerospace and mechanical engineering (dynamics and control), and the Ph.D. degree in computer science from the University of Southern California, Los Angeles, CA, USA, in 2002, 2005, and 2007, respectively.

Dr. Peters was the recipient of the Dick Volz Best 2007 U.S. Ph.D. Thesis Runner-Up Award, the Robotics: Science and Systems—Early Career Spotlight, the INNS Young Investigator Award, and the IEEE Robotics and Automation Society's Early Career Award as well as numerous best paper awards. In 2015, he was also the recipient of an ERC Starting Grant and he was appointed as an IEEE Fellow, in 2019.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2024, Yokohama, Japan. Cite as T-RO paper.