Robotics and Autonomous Systems 204 (2026) 105541

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

RobKiNet: Robotic Kinematics Informed Neural Network for optimal robot configuration prediction

Yanlong Peng [a], Zhigang Wang [b], Yisheng Zhang [a], Pengxu Chang [a], Ziwen He [a], Kai Gu [a], Hongshen Zhang [c], Ming Chen [a,*]

[a] School of Mechanical Engineering, Shanghai Jiao Tong University, No. 800 Dongchuan Road, Minhang District, Shanghai, 200240, China
[b] Intel Labs China, Raycom info tech, No. 2 Kexueyuan South of Road, Haidian District, Beijing, 100190, China
[c] Faculty of Mechanical and Electrical Engineering, Kunming University of Science and Technology, No. 727 Jingming South Road, Chenggong District, Kunming, 650500, Yunnan, China

ARTICLE INFO

Keywords:
AI-based methods
Kinematics
Differentiable programming

ABSTRACT

Task and Motion Planning (TAMP) is essential for robots to interact with the world and accomplish complex tasks. The TAMP problem involves a critical gap: exploring the robot's configuration parameters within continuous space to ensure that task-level global constraints are met while also enhancing the efficiency of subsequent motion planning. Existing methods still have significant room for improvement in terms of efficiency. Recognizing that robot kinematics is a key factor in motion planning, we propose a framework called the Robotic Kinematics Informed Neural Network (RobKiNet) as a bridge between task and motion layers. RobKiNet integrates kinematic knowledge into neural networks to directly learn a mapping from task space to feasible configuration space under global constraints, enabling efficient and constraint-consistent configuration prediction without iterative solving. We designed a Chassis Motion Predictor (CMP) and a Full Motion Predictor (FMP) using RobKiNet, which employed two entirely different sets of forward and inverse kinematics constraints to achieve decoupled control and whole-body control, respectively. RobKiNet significantly improves sampling efficiency and enhances prediction reliability under task-level constraints. Experiments demonstrate that CMP and FMP can predict configuration parameters with 96.67% and 98% accuracy, respectively. That means that the corresponding motion planning can achieve a speedup of 24.24x and 153x compared to random sampling. Furthermore, RobKiNet demonstrates remarkable data efficiency. CMP only requires 1/71 and FMP only requires 1/15052 of the training data for the same prediction accuracy compared to other deep learning methods. These results demonstrate the great potential of RobKiNet in robot applications.

1. Introduction

Robot manipulation in real-world environments requires seamless integration of high-level task reasoning and low-level motion execution [1,2]. Task and Motion Planning (TAMP) frameworks have emerged as a powerful paradigm for addressing this challenge [1,3,4], where task planners determine action sequences to achieve goals, while motion planners generate the corresponding trajectories.

Specifically, the search at the task layer focuses on the abstract subsequences that should be executed to achieve the target state, such as subtask planning in long-horizon tasks [5] or complex dynamic manipulation tasks [3,6]. Speed improvements and near-optimal heuristic searches can be achieved by designing corresponding guided planners that can be learned [7] or efficient neural planners [8,9]. On the other hand, the search at the motion layer is more focused on determining the specific parameter configurations of the entire robotic system when executing a particular action. The selected parameter configurations need to not only satisfy the current task's kinematic requirement but also ensure a feasible solution for the future task based on these parameter configurations [10,11].

However, the interface between these two planning layers remains a significant bottleneck in practical applications [6,12]. A key challenge in TAMP systems is determining viable robot configuration that satisfy both task-level objectives and motion-level constraints, especially for

* Corresponding author.
E-mail addresses: me-pengyanlong@sjtu.edu.cn (Y. Peng), zhi.gang.wang@intel.com (Z. Wang), zys_edward@sjtu.edu.cn (Y. Zhang), 19119175490@sjtu.edu.cn (P. Chang), heziwen@sjtu.edu.cn (Z. He), gukai0707@sjtu.edu.cn (K. Gu), hongshen@kust.edu.cn (H. Zhang), mingchen@sjtu.edu.cn (M. Chen).

https://doi.org/10.1016/j.robot.2026.105541
Received 8 April 2025; Received in revised form 20 April 2026; Accepted 17 May 2026
Available online 25 May 2026

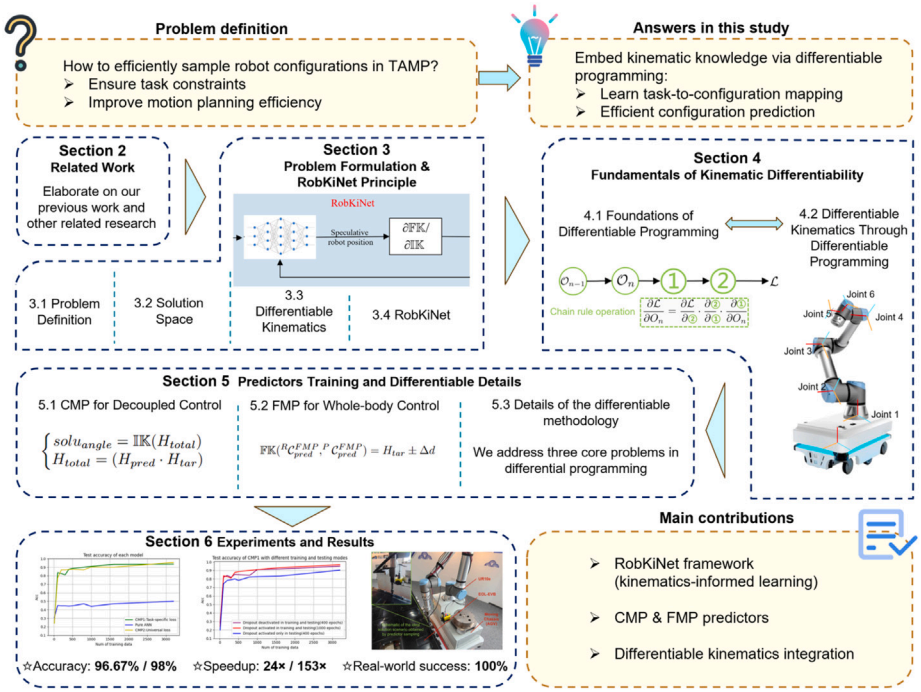


Fig. 1. Overview of this paper's issues, ideas, chapter relationships, and main contributions.

systems with redundant degrees of freedom, which is common in real-world scenarios [13][14][15]. It involves exploring and selecting from a potentially infinite solution space to ensure the correctness of the execution of related tasks.

In order to find viable robot configuration that are suitable for both the task level and the motion level, traditional approaches often rely on adding constraints or random sampling to find feasible configurations. A classic example is the inverse kinematics solution for a 7-degrees of freedom (DOF) robotic arm [16,17], where joint characteristics are constrained according to specific tasks to reduce the motion dimensions and narrow the continuous solution space. However, constraints are often difficult to determine when dealing with the aforementioned complex problems and inefficient. Random sampling [18,19] generate possible parameter configurations and check whether kinematic constraints satisfy all related tasks. However, this approach often becomes inefficient due to the large number of invalid constraint checks, despite many efforts to improve the success rate of sampling [20]. When a motion planner fails to find a trajectory that satisfies the task planner's requirements, the system must backtrack and explore alternative solutions, significantly increasing planning time and reducing overall system reliability.

Therefore, the challenge addressed in this study is how to achieve efficient end-to-end prediction of the robot's configuration parameters while strictly adhering to the constraints of task planning and significantly enhancing the efficiency of subsequent motion planning. This challenge becomes especially urgent when the robot has redundant degrees of freedom and an infinite number of potential solutions exist within a continuous solution space.

To address this critical challenge, recognizing the pivotal role of robot kinematics in TAMP, we have proposed a framework named Robotic Kinematics Informed Neural Network (RobKiNet), a neural network-based framework that serves as an intelligent bridge between task and motion planning. Our approach learns to predict kinematically feasible robot configuration directly from task goals, effectively reducing the gap between abstract task specifications and concrete motion constraints. By integrating kinematic constraints into the training of end-to-end networks, RobKiNet internalized and learned physical kinematic limitations. This significantly reduces the need for expensive iterative planning while maintaining high success rates in real-world manipulation tasks.

In this study, we explore RobKiNet in the field of mobile robots [21,22], which requires deciding high-quality robot configuration parameters based on the constraints of task planning in an infinite continuous solution space for subsequent motion planning. The paper is organized as follows (Fig. 1). After outlining the related work (Section 2), we formally give the notations, define the problem, and illustrate the RobKiNet principle in Section 3. Then, we provide some fundamental knowledge and explain differentiable kinematics in Section 4. Next, we elaborate on our two attempts based on the RobKiNet. We use inverse kinematics (IK) as a physical constraint criterion for the control decoupled from the mobile robot in the chassis motion predictor (CMP) and forward kinematics (FK) as a physical constraint criterion for the control of the whole body [13,23,24] in the full motion predictor (FMP). After that, we detail the core methods of using differentiable physical constraints to train end-to-end networks, which apply to all non-differentiable robotics constraints (Section 5). Finally, we present the experimental details and real-world results (Section 6).

The main contributions of this paper are as follows:

1. We propose RobKiNet, a kinematics-informed neural network that embeds forward and inverse kinematics into the learning process via differentiable programming, enabling direct mapping from task space to feasible configuration space under global constraints.
2. We develop two predictors for autonomous mobile manipulation robots (AMMRs) based on RobKiNet: the Chassis Motion Predictor (CMP) for decoupled control and the Full Motion Predictor (FMP) for whole-body control, both capable of generating valid configurations in an end-to-end manner.
3. A comprehensive technique for embedding robot forward and inverse kinematics constraints into neural networks was developed. This demonstrated the significant potential of differentiable programming techniques in the field of robot TAMP, providing valuable insights and references for the development and optimization of other differentiable computing frameworks.

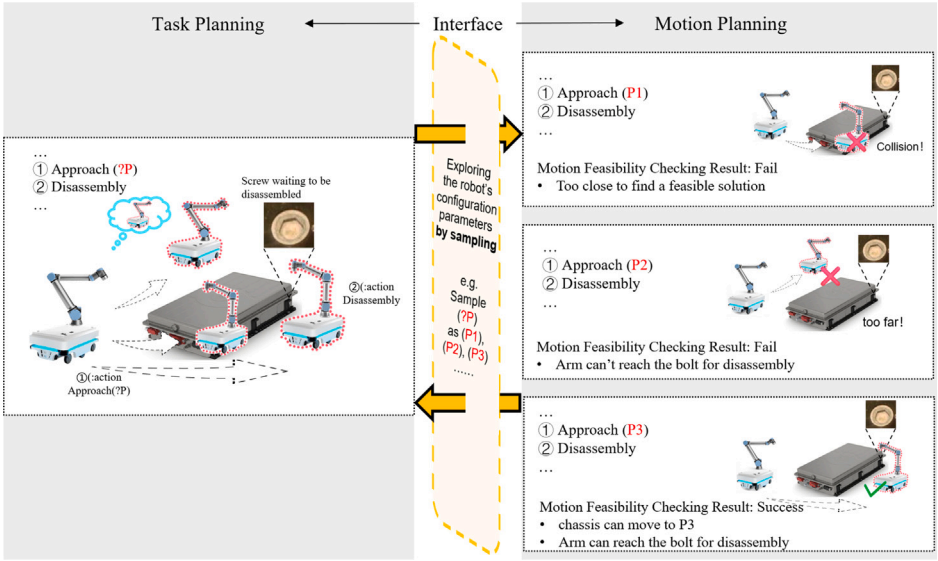


Fig. 2. Example—Simplified TAMP framework for disassembling a screw on an EOL-EVB. (Left) Task Planning Level: The task planner, based on operational requirements (e.g., disassembling a particular bolt), outputs a sequence of action primitives (such as Approach, Disassembly, etc.) that guide the robot from its current state to the goal state. (Middle) Interface Between Task Planning and Motion Planning: This essential interface uses sampling methods to search for the robot's configuration parameters and outputs them to the motion planning level. (Right) Motion Planning Level: The robot's configuration parameters predicted by the interface — such as chassis position and joint angles of the robotic arm — are used to ensure the feasibility of execution. In this study, the predicted parameters from the interface meet both the global constraints of task planning (e.g., ensuring the robot is positioned near the target screw during the Approach primitive) and significantly enhance the efficiency of subsequent motion planning (ensuring feasible kinematic solutions).

2. Related work

2.1. Our previous work

In our previous research, we built a NeuroSymbolic [25,26] Task and Motion Planning System (NeuroSymbolic TAMP [27–30]) to control an Autonomous Mobile Manipulation Robot (AMMR) to accomplish an unstructured industrial task, continuous disassembly of screws from end-of-life electric vehicle batteries (EOL-EVBs).

In such TAMP systems, task planning involves searching for the optimal sequence of subtasks in a high-level symbolic space to transition from the current state to the goal state. These subtasks typically require the robot to perform a series of motion primitives that each necessitate individual motion planning. However, there exists an essential interface between symbolic task planning and motion planning in Cartesian or configuration space (Fig. 2 (Middle)). This interface requires efficient sampling of robot configuration parameters to ensure compliance with task-level global constraints, while also considering constraints imposed by motion planning.

For instance, positioning the chassis too close to the battery may lead to kinematic singularities or an unsolvable kinematic configuration, whereas placing it too far could put the object outside the robot arm's reachable workspace (Fig. 2 (Right)). In this context, efficient sampling of the chassis position is critical, as it takes place after the "Approach" command at the task planning level and before path planning and robotic arm kinematic computations. Its selection takes into account the task constraints while preparing for motion planning.

In this study, we will use the RobKiNet to bridge this gap for optimal robot configuration prediction, which benefits not only our previous research but also other research especially for systems with redundant degrees of freedom.

2.2. Exploring robot configuration parameter

Since the solution space of the redundant degrees of freedom problems is continuous and lacks an explicit equation, the most common methods to explore the robot's configuration parameters involve either predefining relative nominal parameters or performing sampling.

Contemporary pre-programmed methods often rely on a fixed nominal parameter linked to the target to define the configuration parameter, such as the chassis position, a practice commonly observed in analogous tasks [31,32]. However, pre-programming fails in a highly dynamic environment with extensive workspace, diverse task types, and irregular work scenarios. Further, most mobile manipulators today have a loosely coupled chassis and robotic arm. The independent chassis controllers make it difficult to perform whole-body control from the navigation level. For these reasons, we cannot rely on predefined approaches to solve the parameter selection for the robotic arm and the chassis, as tasks are not fixed.

In the motion planning task about the chassis, we need to know both the current chassis position and the intended target position for chassis motion in advance to complete the path planning and robot kinematics simulation. It is common to employ random sampling (RS) [19] or efficient biased sampling (EBS) [33], but this can lead to uncertainty about whether the sampled position is truly effective for task completion. Hence, conducting forward and inverse kinematics simulations of the robot in the simulator becomes necessary, which consumes significant computational resources. The inefficiency of randomly sampling and simulating, together with the gap between simulation results and the real situation, are the key bottlenecks preventing robots from moving towards agility.

Furthermore, when considering the configuration parameters of the robotic arm, additional kinematic constraints must be accounted for Bouhsain et al. [14]. A common approach is to extend the DH matrix to include the chassis as a mobile joint within the overall configuration [13], followed by solving through kinematic analysis. Another method involves using the Jacobian matrix for iterative solving in specific robotic arm configurations [34]. However, these approaches often face challenges in solving complex analytical solutions or can only identify a single feasible solution within the solution space, limiting their ability to accurately and efficiently explore the potential solution space. The exploration of robot configuration parameters should incorporate more efficient methods, such as leveraging prior human knowledge.

2.3. Deep learning for TAMP

Considering the substantial benefits neural networks have demonstrated in learning from diverse data types, many researchers are trying to apply deep learning to TAMP. Regression in deep learning refers to a type of supervised learning task where the goal is to predict a continuous output variable based on input features. It is a natural candidate for TAMP. Its advantage lies in its ability to constrain and guide TAMP using known datasets [8–10]. However, this method faces the challenge of obtaining large amounts of real-world physical interaction data. In addition, the inherent characteristic of motion planning being a one-to-many mapping further increases the difficulty of convergence for regression methods.

Deep reinforcement learning (DRL) [35–37] is another widely used deep learning method for TAMP. It focuses on training agents to make a sequence of decisions by interacting with an environment to maximize cumulative rewards. While DRL has shown remarkable success in various applications [38,39], it also comes with efficiency challenges. DRL algorithms often require a large number of interactions with the environment to learn effective policies. This poses significant challenges in real-world applications where data collection is costly and time-consuming.

More recently, Vision-Language-Action (VLA) models have emerged as a promising direction, enabling robots to perform high-level reasoning and task execution based on multimodal inputs. For example, SayCan [40] and RT-2 [41] demonstrate strong generalization in task planning by leveraging large-scale pretrained models. However, these approaches typically rely on downstream motion planners or controllers, and do not explicitly address the problem of efficient and constraint-consistent configuration sampling in continuous spaces, which remains critical for TAMP.

Physics-informed neural networks (PINNs) can effectively incorporate physical equations as constraints into the neural network training process [42,43]. This provides a new approach to solving TAMP planning problems using deep learning. However, PINNs cannot be directly applied to the domain of robot kinematics because they require prior knowledge in the form of Partial Differential Equations (PDEs). The approach taken by PINNs is inspiring, and we aim to take it a step further in the field of robotics. By incorporating complex constraints as prior knowledge beyond just partial differential equations, we can enable efficient sampling within a narrowed solution space. In addition, hybrid inverse kinematics methods that combine analytical formulations with learning-based components have been proposed to improve robustness and efficiency. For instance, HybrIK [44] integrates analytical kinematic constraints with neural predictions for inverse kinematics solving. While such approaches enhance solution accuracy, they are typically designed for solving IK problems through structured formulations, rather than learning a direct mapping for efficient sampling in TAMP scenarios.

It is worth noting that our approach is also fundamentally different from existing differentiable robotics or differentiable kinematics frameworks [45,46]. These methods typically focus on making forward/inverse kinematics or dynamics differentiable as computational modules for gradient-based optimization or control, whereas they do not explicitly learn a mapping from task space to feasible configuration space under global constraints.

We aim to develop a framework that incorporates robot kinematic knowledge as constraints within a neural network to predict robot configuration parameters. This framework aims to provide an efficient solution for motion planning — specifically, the sampling of infinite solutions in continuous space — while satisfying the global constraints required by task planning.

2.4. Deep learning for TAMP

3. Notations, definitions and RobKiNet principle

This section introduces commonly used symbols, formal definitions, and key evaluation metrics relevant to this study. For ease of reading, we have provided a symbol table (Table 1) to clarify potentially confusing notations and definitions. Further, we explain the principles of the RobKiNet in general terms using a contrast with traditional methods.

3.1. Problem definition and generalization

In robotic systems with n degrees of freedom, the configuration space $\mathcal{C} \subseteq \mathbb{R}^n$ represents the set of all possible configurations the system can achieve. $\mathcal{C}$ is the most expansive space, encompassing every feasible configuration defined solely by the system's mechanical structure. However, when performing a specific task, the solution space is reduced to a subset $\mathcal{X} \subseteq \mathcal{C}$, constrained by the requirements of that particular task. The task-specific solution space $\mathcal{X}$ consists of an infinite number of valid configurations that all satisfy the given task's constraints, forming a continuous space.

Importantly, when additional global constraints are introduced, such as multi-tasks kinematic constraints or environmental restrictions, the solution space becomes further refined, reducing to $\mathcal{C}^* \in \mathcal{X}$. This represents the most restrictive subset, where each configuration satisfies not only the task-specific constraints but also the motion layers' constraints. From a topological perspective, the space $\mathcal{C}^*$ is ideally an open set within $\mathcal{X}$. This ensures robustness, as small perturbations around any point in $\mathcal{C}^*$ will remain within $\mathcal{X}$, meaning that slight deviations in configuration do not invalidate the solution. This hierarchical structure — from the overall configuration space $\mathcal{C}$ to the task-specific solution space $\mathcal{X}$ and finally to the globally constrained solution space $\mathcal{C}^*$ — allows for a rigorous and robust framework for understanding how robotic systems operate within complex environments.

The problem now shifts to sampling within the infinite, continuous space $\mathcal{X}$ to identify the optimal solution space $\mathcal{C}^* \in \mathcal{X}$, guided by a loss function $\mathcal{L} : \mathcal{C} \to \mathbb{R}$, where $\mathbb{R}$ denotes the real-valued space of loss metrics, distinct from the configuration space $\mathbb{R}^n$. Each configuration in $\mathcal{X}$ is evaluated by this loss function, representing how well the solution aligns with all global constraints. The challenge lies in efficiently exploring $\mathcal{X}$, selecting a set of candidate solutions, and identifying the optimal configuration in $\mathcal{C}^*$ that minimizes the loss $\mathcal{L}(C)$ while ensuring that all system constraints are satisfied.

In our example of the mobile robot (Fig. 2), consider the configuration space $\mathcal{C}_{base}$ for the mobile chassis, typically represented as a continuous space (e.g., $\mathbb{R}^2$ or $\mathbb{R}^3$). Let $\mathcal{C}_{arm}$ denote the joint configuration space of the manipulator's joints, described by the vector of joint angles $\boldsymbol{\theta} \in \mathcal{C}_{arm}$. For a manipulator with 6-DOF, the $\boldsymbol{\theta}$ can be represented as:

$$\boldsymbol{\theta} = \left(q_j^i\right)_{8\times 6}, \quad i = 1, 2, \ldots, 8 \text{ and } j = 1, 2, \ldots, 6 \tag{1}$$

This represents eight sets of inverse kinematics solutions, where each q^i denotes the ith set of solutions, consisting of the values for six joints. The task involves finding configurations $\mathcal{C}^* \in \mathcal{C}_{base}$ such that the target object is within the manipulator's workspace $\mathcal{W}$ and the manipulator's inverse kinematics solution exists.

The problem can be formalized as follows:

Chassis Position Validity Condition: The system configurations in $\mathcal{C}^*$ must allow the manipulator's end-effector to reach the target object's position and orientation $\mathcal{T}_{target}$. This requires finding a chassis position $\mathcal{C}^*_{base}$ and a corresponding arm configuration $\boldsymbol{\theta}^*$ such that the forward kinematics function $\mathcal{F}(\mathcal{T}^*_{base}, \boldsymbol{\theta}^*)$ maps to $\mathcal{T}_{target}$:

$$\mathcal{F}(\mathcal{T}^*_{base}, \boldsymbol{\theta}^*) = \mathcal{T}_{target} \tag{2}$$

where $\mathcal{F}$ represents the forward kinematics function mapping the chassis and arm configurations to the workspace.

Existence of Inverse Kinematics Solution: For a given chassis position $\mathcal{C}^*_{base}$, there must exist at least one arm configuration $\boldsymbol{\theta}^*$ such that

Table 1
Notations and Definitions.

Notations	Definitions
$\mathcal{C}$	Configuration space, representing all possible configurations of the robot system.
$\mathcal{C}_{base}$/ $\mathcal{C}_{arm}$	Configuration space of the mobile chassis/manipulator's joints.
$\mathcal{X}$	Task-specific solution space, a subset of $\mathcal{C}$ that satisfies a specific task's constraints.
$\mathcal{C}^*$	Globally constrained solution space, a subset of $\mathcal{X}$ that satisfies additional global constraints.
C^*_{base}	Optimal solutions/Ideal solution of the chassis.
$\boldsymbol{\theta}$	Manipulator joints' angle matrix.
$\boldsymbol{\theta}^*$	Optimal solutions/Ideal solution of the manipulator joints' angle.
C_{pred}	Estimated configuration through forward propagation of the end-to-end network.
C^{CMP}_{pred}/ C^{FMP}_{pred}	Output of the CMP or the FMP.
${}^{P}C^{FMP}_{pred}$/ ${}^{R}C^{FMP}_{pred}$	Sample for the chassis/Sample for the manipulator in FMP. Decoupled and independent in computation.
$\mathcal{T}_{target}$	The position and orientation of the target object.
$\mathcal{L}$	The real-valued space representing loss metrics, guiding the training process of the end-to-end network.
$\mathcal{W}, \mathcal{W}_1$	Manipulator's workspace (Cartesian space) at its current different C_{pred}.
(w, b)	Refers to the weights and biases in the neural network.
$\mathbb{IK}/\mathbb{FK}$	Differentiable kinematics computation engines obtained through differentiable programming.
$\mathcal{O}, \mathcal{O}_1, \ldots, \mathcal{O}_n$	Different operators in the computation graph that represent differentiable operations.
H_{pred}, H_{tar}	Homogeneous transformation matrix form of C_{pred}, $\mathcal{T}_{target}$
H_{total}	Represents the coordinate relationship between the end joint and the base (homogeneous transformation matrix).
Δd	The prediction error of the FMP.
(:action Approach	The PDDL [19,47] form of the action primitive "Approach" in our NeuroSymbolic TAMP Framework.

the end-effector can achieve the desired $\mathcal{T}_{target}$. The set of all valid arm configurations for a given chassis position is defined as:

$$S = \left\{ \theta^* \in C_{arm} \mid \mathcal{F}(C^*_{base}, \theta^*) = \mathcal{T}_{target} \right\} \tag{3}$$

The goal is to find $C^*_{base} \in C_{base}$ such that $S \neq \emptyset$.

Since C_{base} and C_{arm} are continuous spaces, the set of solutions C^* that satisfy the above conditions forms a continuous manifold, potentially with infinitely many solutions. This scenario illustrates the challenge of dealing with infinite solution sets in continuous spaces, where the exact enumeration of solutions is infeasible.

The motivation for this task is easily understood when we consider it from a human perspective. When a person needs to reach for a cup on a distant table, they instinctively move to an appropriate target position. Humans can achieve this effortlessly and without conscious thought because their brain inherently accounts for constraints such as the arm's workspace and singularities.

Furthermore, when performing TAMP on robots under prior constraints such as world models, the sampling problems in continuous solution spaces are abundant. Another common example is the grasping problem, where grasp points vary continuously across different objects but are subject to directional constraints [14].

3.2. Ideal solution and accuracy definition

We aim to solve this problem through end-to-end networks, so we need to define the ideal solution. The definition of the ideal solution does not rely on explicit dataset mappings but on known constraints, such as the CMP and the FMP defined under different kinematic constraints. The significance of defining the ideal solution lies in the need to guide and supervise backpropagation during network training. Our advantage is that network training can be guided by known constraints, which saves a significant amount of computational resources.

Suppose C_{pred} is the estimated target position obtained by the robot through forward propagation of the end-to-end network. The ideal solution space C^* is defined by satisfying the following conditions:

Existence Condition

The relationship between the target position and the workspace, as well as the decision on whether movement is required, is determined at the task planning level. However, it must always ensure that after motion planning:

$$\mathcal{T}_{target} \in \left\{ \mathcal{W}_1 \mid C^*_{base} \in C_{base} \right\} \tag{4}$$

Reachability Condition

By using the CMP for decoupled control, the $\boldsymbol{\theta}^*$ is implicit. By using the FMP for whole-body control, the $\boldsymbol{\theta}^*$ is an explicit result. Both cases satisfy the Chassis Position Validity Condition mentioned before and:

$$\exists \boldsymbol{\theta}^* = q^i_j, q^i_j \in [-\pi, \pi] \tag{5}$$

Then the accuracy of the network in this research ACC is defined as:

$$ACC = \frac{\sum_{i=1}^{N} \mathbb{I}(C_{pred}, C^*)}{N} \tag{6}$$

where $\mathbb{I}$ is an indicator function. The function takes the value of 1 when the predicted value C_{pred} matches the ideal space C^*, and 0 otherwise. N is the total number of samples using random ways or the methodologies proposed in this research.

This metric directly reflects a sampling method's ability to provide the ideal solution across N independent tasks, thereby indicating whether the end-to-end network guiding the robot has effectively learned the physical constraints. In the following experiments, we will compare random sampling/iteration methods, traditional supervised/reinforcement learning methods, and the two predictors we proposed by the RobKiNet, with ACC serving as the core evaluation metric.

3.3. Differentiable kinematics

It is important to clarify that all mentions of "Differentiable" in this study refer specifically to "Differentiable Kinematics through differentiable programming [48,49]". The differentiable kinematics here does not refer to the common practice of total differentiation for dynamics solving, but rather to differentiability at the computational graph level [48,50,51]. This is a key method in our proposed RobKiNet for incorporating human prior knowledge and physical constraints into end-to-end networks for robots. Consequently, backpropagation is defined as follows:

Differentiable Kinematics: differentiable programming $\{IK, FK\} \rightarrow \{\mathbb{IK}, \mathbb{FK}\}$

$$\frac{\partial \mathcal{L}}{\partial (w, b)} = \frac{\partial \mathcal{L}}{\partial \mathbb{IK}/\mathbb{FK}} \cdot \frac{\partial \mathbb{IK}/\mathbb{FK}}{\partial y} \cdot \frac{\partial y}{\partial (w, b)} \tag{7}$$

Here, we use $\{IK\}$, $\{FK\}$ to denote the processes of inverse kinematics and forward kinematics, respectively. $\mathbb{IK}$ and $\mathbb{FK}$ represent the differentiable kinematics computation engines obtained through differentiable programming methods discussed in this paper. (w, b) refers to the weights and biases in the neural network. y denotes the network prediction results.

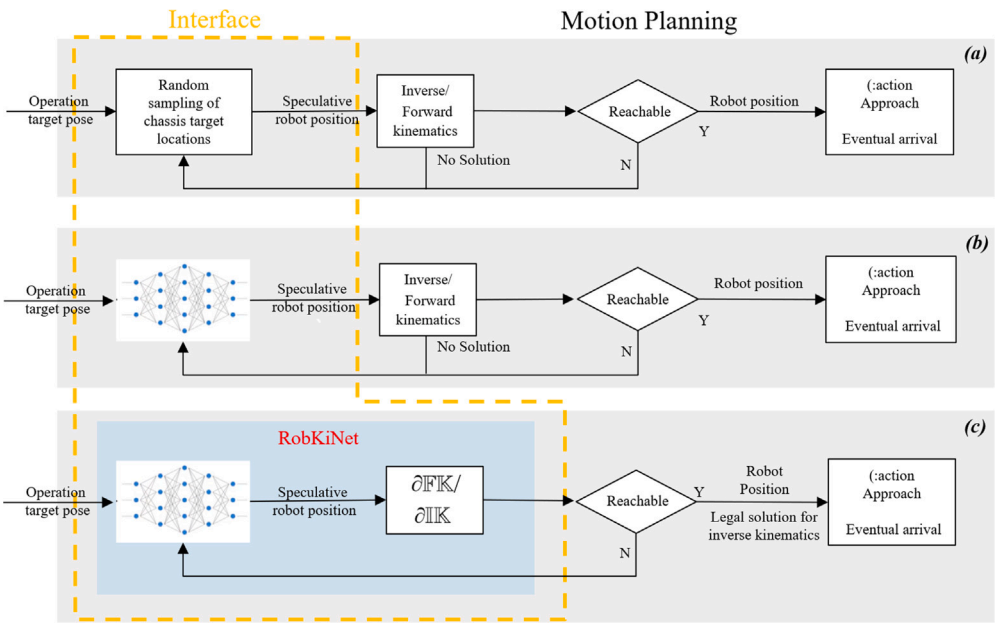


Fig. 3. Comparison of three robot configuration (such as the chassis position) sampling methods. (a) Randomized sampling. (b) NN Regression method. (c) RobKiNet proposed in this work. The effectiveness of sampling in a continuous solution space will greatly affect the efficiency of motion planning. RobKiNet eliminates the need to iterate several times to validate the kinematics in a simulation environment.

3.4. RobKiNet principle: Integrating neural networks with physical constraints

Based on the problem definition presented earlier, we will explain why RobKiNet is needed and outline its implementation principles by comparing it with traditional methods.

We often rely on prior knowledge to ensure that a robot's interaction with the environment is safe and reliable. Traditional methods involve setting up precise simulation models tailored to different scenarios and interaction environments. However, this is difficult to achieve in the real world's highly dynamic and unknown environments. The world model approach mentioned earlier typically uses data-driven networks to guide the robot, but data faces the problem of long-tail sparsity [52,53]. The constraints represented by datasets are limited, and in unfamiliar scenarios, it is challenging to achieve zero-shot learning or one-shot learning. This may lead to the robot violating physical laws.

To address these issues, our approach is to enable the end-to-end network to learn robotics physical constraints rather than the latent mappings of a dataset. This is the essence of the RobKiNet. Our RobKiNet is designed to inject the robot's kinematic constraints as physical constraints into the network to guide the sampling of the robot configuration in the interface between the task and motion planning layers. We will explain the RobKiNet by using the example that injects the differentiable kinematics to solve the sampling problem of infinite solutions in continuous space.

Fig. 3 shows the differences in determining the robot configuration using different pipelines. Fig. 3(a) is the common method of the PDDLSTREAM framework [19,54]. For the chassis target position, the RS method presents an assumed chassis position, which is then employed in the robot's inverse kinematics calculation to determine the feasibility of a valid solution and the robot's reachability. However, RS methods often necessitate closed-loop feedback and multiple sampling iterations to determine the correct final position. This prolonged process results in increased task duration and multiple recalculations for the robot's navigation. With the help of neural networks (Fig. 3(b)), this process can be optimized using a large amount of data training, through which we hope that the Artificial Neural Network (ANN) can learn the implicit mapping relationship between the current position and the target position, thus outputting a more credible result. Due to its supervised learning essence, we call it the NN Regression approach in our experiments.

Compared to the previous two methods, the most significant feature of our RobKiNet is the substantial expansion of the network's end-to-end scope. The network is designed to incorporate physical constraints that we want it to learn, such as complex kinematics. The greatest advantage of neural networks is that they can be expressed as a computational graph, where all forward and backward propagations are executed through the chain rule, making them a series of differentiable mathematical processes. Therefore, the physical constraints we incorporate must be differentiable.

With the introduction of differentiable kinematics, the training of the network undergoes a fundamental transformation (in Eq. (7)). During forward propagation, we impose penalties on computational graph nodes that do not satisfy the physical constraints. During backward propagation, the network can then produce optimal parameters through gradient optimization. The RobKiNet process can be exemplified by Fig. 3(c) which consists of two modules: the Position Prediction Network and the Differentiable Kinematics Calculation Engine. When the pose of the target object (such as the bolt to be disassembled) is provided as input, the predictors can directly output the position that the chassis should reach (in CMP and FMP) and provide the corresponding legal solutions for each joint (in FMP), which is approximate to the subconscious intuition of human beings. This method will greatly improve the efficiency of motion planning.

It is worth emphasizing that the fundamental difference between our RobKiNet and traditional neural networks lies in the fact that RobKiNet learns how to adhere to physical constraints during training, rather than learning a specific dataset mapping. For the predictors, the forward and inverse kinematics algorithms are added to the training process of the neural network in a differentiable way and are involved in learning to steer the output values of the network before backpropagation. The traditional neural network computational graph will be greatly changed due to the addition of the robotic differentiable computation as a computational node after the forward propagation. The addition of robotic differentiable computation as a computational node after forward propagation significantly alters the traditional neural network computational graph.

4. Fundamentals of kinematic differentiability

The implementation of Differentiable Kinematics (in Eq. (7)) is achieved by identifying and replacing non-differentiable operators in the kinematics computation with corresponding differentiable operators. For instance, redefining operators at the limits of function values or at discontinuities ensures the completeness of the computational graph. This section will provide a fundamental understanding of kinematics and differentiable programming, explaining the feasibility and operational logic behind their integration.

4.1. Foundations of differentiable programming

The primary objective of differentiable programming is to enable automatic differentiation (AD) [55], which is a critical tool in modern computational frameworks. AD operates by systematically applying the chain rule of calculus to compute derivatives efficiently, ensuring precise gradient propagation through complex systems. The computation graph plays a pivotal role in this process, as it must be designed to support the seamless flow of gradients across its nodes.

Assume that $\mathcal{O}_1, \ldots, \mathcal{O}_n$ are a total of n operators that together form a function $f : \mathbb{R}^s \to \mathbb{R}^t$, which completes the mapping from input to output in our system.

The nodes of the computational graph of the mapping represent operators and the edges represent dependencies. An operator $\mathcal{O}_i : \mathbb{R}^{s_i} \to \mathbb{R}^{t_i}$ in the graph is a differentiable operator only if it satisfies the condition that the gradient $\nabla\mathcal{O}_i$ exists everywhere. Then, the backpropagation can be symbolized as applying the chain rule to the composite function represented by the computational graph:

$$\nabla f(x) = \nabla\mathcal{O}_n(\mathcal{O}_{n-1}) \cdot \nabla\mathcal{O}_{n-1}(\mathcal{O}_{n-2}) \cdot \ldots \cdot \nabla\mathcal{O}_1(x) \tag{8}$$

Each operator $\mathcal{O}_i$ represents a fundamental building block of the computation graph, ranging from a simple linear operation to a complex function composed of multiple operations. Our goal is to express the kinematics process as a combination of such differentiable operators, thereby enabling full differentiability. This modular approach to designing differentiable programs allows us to control the granularity of automatic differentiation, facilitating an efficient differentiable kinematics process.

4.2. Differentiable kinematics through differentiable programming

In this paper, the kinematics algorithm follows the Denavit–Hartenberg (DH) convention (Fig. 4(a)) [56], and the homogeneous transformation matrix of any joint i concerning its predecessor joint $i-1$ can be expressed as:

$$H_i^{i-1} = A_i = Rot_{z,\theta_i}\,\mathrm{Trans}_{z,d_i}\,\mathrm{Trans}_{x,a_i}\,Rot_{x,\alpha_i} \tag{9}$$

θ is the joint angle, d is the linkage offset, a is the linkage length, and α is the linkage twist. For a manipulator with a known configuration, we can construct a DH table and extract the corresponding hyperparameters for each joint from it. Then the forward kinematics formula for a 6-DOF robotic arm is:

$$H_{total} = \prod_{i=0}^{5} H_i^{i+1} \tag{10}$$

H_{total} represents the coordinate relationship between the end joint and the base, consisting of orientation and relative position:

$$H_{total} = \begin{pmatrix} R & d \\ 0 & 1 \end{pmatrix}, \quad R \in SO(3), \quad d \in \mathbb{R}^3 \tag{11}$$

Fig. 4(b), (c), and (d) illustrates the analytical solution-solving procedure for the two kinematic constraints involved in our study, as well as the non-differentiable parts encountered in the solution of the program. When programming the computation for FK (Fig. 4(c)), we need to define each transformation matrix H_i^{i-1} and use matrix multiplication to obtain the final result. However, this definition and these multiplication operations are non-differentiable because they consist of independent scalar elements, and the partial derivatives within these operators cannot be propagated backward to the preceding operator.

The IK calculation works in the opposite way to FK (Fig. 4(b)). Given the transformation matrix T_n^0 between the end-effector and the base, we aim to find the inverse solution by seeking one or more solutions to the following equations:

$$\begin{aligned} T_n^0(q_1, \ldots, q_n) &= H_{total} \\ T_n^0(q_1, \ldots, q_n) &= A_1(q_1) \cdots A_n(q_n) \end{aligned} \tag{12}$$

The q_n denotes the value of the nth joint, which is exactly the quantity we need to solve for in inverse kinematics. Therefore, the process of obtaining analytical solutions for inverse kinematics (IK) is more complex, as the program needs to sequentially solve for the possible values of the six joint angles. This involves several operations that are difficult to differentiate, such as the use of the atan2 function or the definition of independent intermediate variables. These steps can cause the operators in the computational graph to lose their connected edges, resulting in the loss of dependencies.

We list some of the FK and IK processes in (Fig. 4(d)) as non-differentiable computational processes that are marked in red to represent their inability to participate in the computational graph. Thus, the core motivation of our research (Fig. 4(e)) is to seek a unified approach by using differentiable programming technology to differentiate programs in both machine learning and scientific domains.

Our core solution is to individually define each non-differentiable operation, transforming it into a differentiable operator. For instance, as illustrated in Fig. 4(f), in the chain rule operation process, $\mathcal{O}_n$ can refer to any differentiable operator in the calculation. Non-differentiable code processes (red circles 1 to 7) are encapsulated separately, becoming operators that perform the function of the operators(such as the atan2 function) with both forward and backward propagation capabilities (green circles 1 to 7). The detailed methodological implementation of this transformation process is described in 5.3, where we summarize several types of representative and generalizable methods. This allows the gradient to be propagated under the chain rule within the local computational graph(in Eq. (8)). Thus implements differentiable programming $\{IK, FK\} \to \{\mathbb{IK}, \mathbb{FK}\}$.

It is important to note that in subsequent computations, the neural network output $\mathcal{C}_{pred}$ typically consists of one-dimensional position and orientation information. This cannot be directly utilized by the differentiable $\mathbb{IK}$ and $\mathbb{FK}$ since kinematics require a homogeneous transformation matrix. We achieve this conversion through a transformation function, which must also be differentiable and become one of the operators.

$$H_{pred}, H_{tar} = \text{Shaping operator}(\mathcal{C}_{pred}, \mathcal{T}_{target}) \tag{13}$$

We use H_{pred} and H_{tar} to represent the homogeneous transformation matrix form of $\mathcal{C}_{pred}$ and $\mathcal{T}_{target}$, respectively.

5. Predictors training and differentiable details

In this section, we will discuss how we utilize RobKiNet to create two separate predictors to tackle the sampling issue of configuration in continuous solution spaces. We have chosen two fundamentally different kinematic constraints to execute distinct control strategies. Additionally, we will provide details on the implementation of differentiable programming.

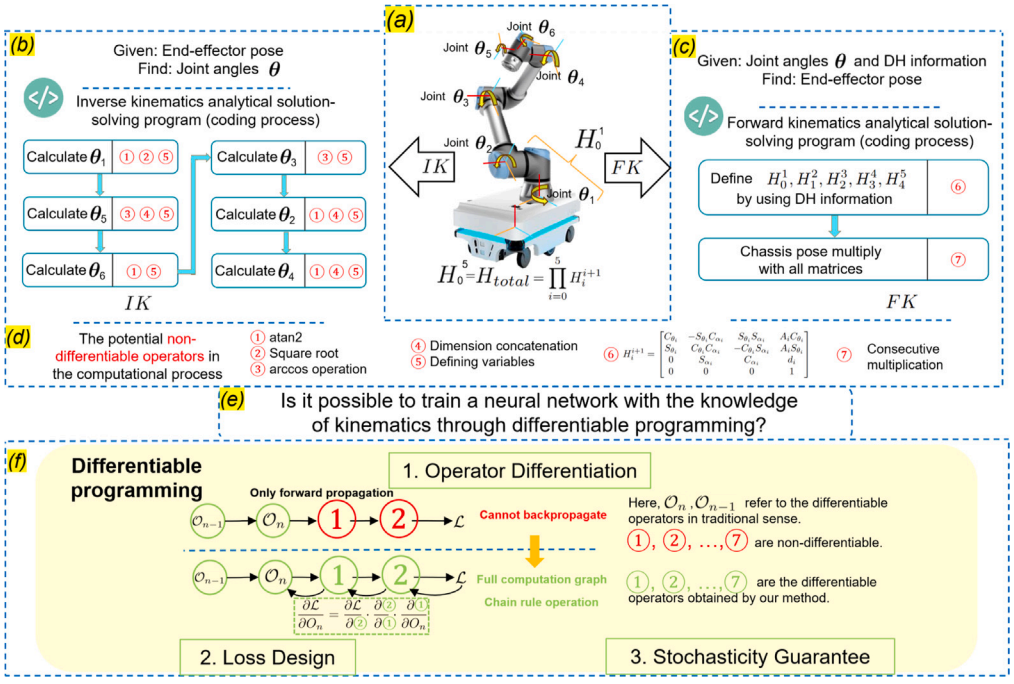


Fig. 4. An overview of differentiable kinematics and differentiable programming.
(a) Overview of the joint coordinate systems for the 6-DOF robotic arm kinematics. (b) The computational process for the analytical inverse kinematics solution. (c) The computational process for the analytical forward kinematics solution. (d) Non-differentiable operations and function modules within the program. (e) The motivation behind this study. (f) The basic concept of differentiable programming—ensuring the chain rule of the computational graph. This is achieved by transforming non-differentiable operators (red circles) into differentiable operators (green). There are other challenges to differential programming (2, 3), and solutions to all these problems will be detailed later. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

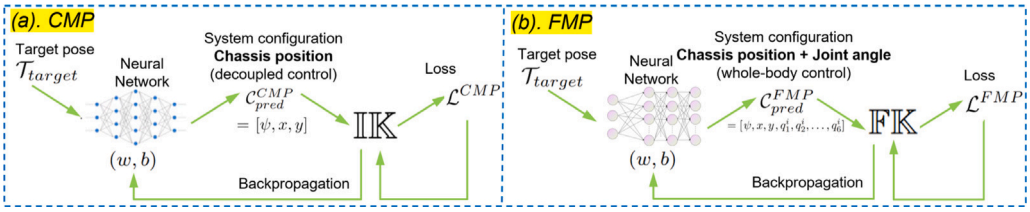


Fig. 5. An overview of network architectures(training stage).
(a) The integration of $\mathbb{IK}$ with neural networks for chassis position prediction (CMP). (b) The integration of $\mathbb{FK}$ with neural networks for predicting both chassis position and joint angle configuration (FMP).

5.1. Computational graph construction and the CMP training for decoupled control

We first define CMP using our RobKiNet in conjunction with more complex inverse kinematics (Fig. 5(a)). The structure of the CMP network requires that its robotic inverse kinematics computational engine be integrated into the training process in a differentiable form. This means neither existing computational functions nor third-party libraries can be added as nodes to the computational graph of the neural network backpropagation due to their discontinuities.

For the neural network input tensor, as in the previous example of the bolt pose to be disassembled, we give $\mathcal{T}_{target} = [\varphi, \theta, \psi, x, y, z]$, $\mathcal{T}_{target} \in \mathbb{R}^6$, where φ, θ, ψ represent the spatial orientation described using Euler angles. Considering that the robot's actual working process is unchanged compared to the chassis position, its pitch angle, lateral inclination, and longitudinal displacement should be zero. Under the action of the position prediction network, the robot target chassis position can be predicted as:

$$C_{pred}^{CMP} = \text{Neural Network1}(\mathcal{T}_{target}) = [\psi, x, y] \in C_{base} \tag{14}$$

Then the solution of its inverse kinematics under the differentiable $\mathbb{IK}$ computational engine proposed in this paper can be expressed as:

$$\begin{cases} solu_{angle} = \mathbb{IK}(H_{total}) \\ H_{total} = ((H_{pred})^{Transpose} \cdot H_{tar}) \end{cases} \tag{15}$$
$$solu_{angle} \in \mathbb{R}^{8\times 6}$$

Where H_{pred} and H_{tar} are defined in Eq. (13) by transforming the neural network's output C_{pred}^{CMP} into Eulerian coordinates, followed by conversion into a rotation matrix, and ultimately into a homogeneous transformation matrix. The $\mathbb{IK}$ is the differentiable computational engine for the inverse kinematics algorithm. The inverse kinematics solves eight sets of solutions, each with six joint values, corresponding to the six joints of the six-axis robotic arm, which can be expressed as Eq. (1).

During programming, to make this solution process differentiable, we redefine the traditional matrixed kinematics solution by Fig. 4(f). This ensures that the robot's kinematics solution can be added to the computational graph. The introduction of differentiable computational processes like $\mathbb{IK}$ into the computational graph significantly increases

its complexity. The computational graphs illustrating actual participation in CMP and the following FMP are available for download and inspection at the following website.[1]

5.2. FMP training for whole-body control

Most of the mobile manipulators currently in use employ a loosely coupled integration between the chassis and the robotic arm, with each being controlled independently by separate controllers. The design of the CMP specifically addresses this loosely coupled scenario, where the network outputs only the parameters for the chassis. However, a more advanced approach involves whole-body control [13,23,24], enabling coordinated control of the entire system. Consequently, we aim for the predictor to simultaneously provide both the sampled positions of the chassis and the six joint angles similar to the expanded DH method [13]. This prediction, made without any actual physical action, must still accurately adhere to the physical constraints of forward kinematics.

To achieve this, we adopted the RobKiNet and selected the second type of kinematic constraint, constructing the FMP specifically for whole-body control. FMP can directly output the chassis position along with the anticipated posture at that position, aligning more closely with human intuition and the subconscious behavior we exhibit when interacting with the world.

The principle of the FMP can be expressed by the following equation. With the input target pose $\mathcal{T}_{target}$, we predict the whole-body control system parameters:

$$C_{pred}^{FMP} = \text{Neural Network2}(\mathcal{T}_{target}) \tag{16}$$

$$C_{pred}^{FMP} = [\underbrace{\psi, x, y}_{^{P}C_{pred}^{FMP}}, \underbrace{q_1^i, q_2^i, \ldots, q_6^i}_{^{R}C_{pred}^{FMP}}]^T, C_{pred}^{FMP} \in \mathbb{R}^9 \tag{17}$$

According to Fig. 5(b), we need to use differentiable forward kinematics $\mathbb{FK}$ to calculate the prediction error:

$$\mathbb{FK}(^{R}C_{pred}^{FMP}, {}^{P}C_{pred}^{FMP}) = H_{tar} \pm \Delta d \tag{18}$$

Here, the forward kinematics formulae derived from Eq. (9) for a six-axis robotic arm are presented in Eq. (10). C_{pred}^{FMP} is the output of the FMP for whole-body control, which consists of the sample for the chassis $^{P}C_{pred}^{FMP} \in C_{base}$ and the corresponding joints $^{R}C_{pred}^{FMP} \in C_{arm}$ for the manipulator. $\mathbb{FK}$ is the differential calculus form of Eq. (10). Δd is the prediction error. Essentially, this involves forward-propagating the predicted parameters through the computational graph to assess accuracy before any action is taken. In FMP we specify that $\Delta d \leq 1$ mm. This means that no algebraic solvers are required, the positional error between the robotic arm and the actual target remains within a 1 mm range based on the network's predicted configuration.

5.3. Details of the differentiable methodology

In the process of differentiating the entire computation, we present the following three difficult problems (shown in Fig. 4(f)):

(i) Not all parts of forward and inverse kinematics can be easily differentiated. How can discontinuous operators be made differentiable and participate in the computational graph?
(ii) CMP and FMP should learn how to follow physical constraints during training. How can the loss function be defined to effectively guide the training process?
(iii) The parameters of the differentiable network are fixed after the training is completed. How to ensure its stochasticity to make the ideal solution space large enough?

[1] https://sites.google.com/view/sjtu-robkinet

5.3.1. Operator differentiation methods

For the first question, we have compiled a list of commonly used differential methods in differentiable programming (Fig. 6). These methods assist us in handling various non-differentiable operators.

Core Idea: First, identifying the inputs and outputs of the operator; Then, ensuring they originate from or point to other nodes within the computation graph; Finally, maintaining inner continuity without leaf nodes.

Methods C and D in the figure are relatively straightforward, as the PyTorch framework already provides several computational operators useful for algorithm development, including but not limited to torch.mm for matrix multiplication and torch.stack for dimensional reorganization. We simply need to follow the Core Idea in our programming. We will illustrate the general methodology with examples of the atan2 function (method type A) and the *arccos* function (method type B) during CMP training.

A widely adopted practice in robot inverse kinematics involves leveraging the atan2 function as an efficient solution for calculating inverse tangents. However, the standard form of the atan2 function is discontinuous, causing direct interruptions in gradient flow within the computational graph. Hence, it becomes imperative to redefine both the forward and backward propagation processes of the function itself, as illustrated in Algorithm 1.

Algorithm 1 Differentiable Four-Quadrant Arctangent

class Atan2Function: ▷ From torch. autograd.Function
function FORWARD(y, x)
  $result \leftarrow$ CALCULATE_ATAN2(y, x)
  $ctx \leftarrow$SAVE_FOR_BACKWARD(x, y)
  return TENSOR($result$, requires_grad = True)
end function
function BACKWARD(ctx, grad_output)
  $x, y \leftarrow$ SAVED_TENSORS(ctx)
  $grad\_y \leftarrow \frac{\mathrm{d}}{\mathrm{d}y}\text{atan2}(y, x) \leftarrow x/(x^2 + y^2)$
  $grad\_x \leftarrow \frac{\mathrm{d}}{\mathrm{d}x}\text{atan2}(y, x) \leftarrow -y/(x^2 + y^2)$
  return grad_output $\times$ $grad\_y$, grad_output $\times$ $grad\_x$
end function
function MAIN
  $y \leftarrow$ GET_INPUT_Y
  $x \leftarrow$ GET_INPUT_X
  $atan2 \leftarrow$ ATAN2FUNCTION.APPLY(y, x)
end function

Atan2Function defines its propagation process in the form of a class. While forward propagation computes the results and stores the values of the nodes, the derivation of backpropagation can be expressed as the gradient of the node equals the result of multiplying the partial derivatives of that node by the cumulative gradient passed from the previous node. The relationship can also be seen as a concrete application of the chain rule. In this way, this inherently non-differentiable function successfully participates in the computational graph. It turns out that by simply changing the function's body, this approach can be applied to address a range of formally discontinuous functions, ensuring computational differentiability.

Another frequent differentiable error in inverse kinematics occurs when an intermediate variable falls outside the domain of the mathematical function that operates on it. When confronted with this problem, the core idea is to continue to redefine the propagation operations of the function. Taking the *arccos* function as an example, when it appears that the value of the previous node in the computational graph is outside its domain of definition, we construct a new customized extended differentiable function $G(x)$, with the core arithmetic function *arccos* of the current node as a base function. Taking $\delta > 0$ and δ as

Method Type	Applicable Non-Differentiable Operators (Red Circles)	Operation Explanation
A. Propagation Redefinition	①	Redefines the forward and backward propagation processes by encapsulating new classes.
B. Limit and Discontinuity Handling	② ③	Redefines new functions to handle non-differentiable situations and ensure invariance of properties.
C. Torch Operator Composition	④ ⑦	Uses differentiable operators from the PyTorch ecosystem to handle multiple iterations step-by-step, allowing for matrix concatenation, dimensional upgrading, and differentiable multiplication.
D. Inheritance	⑤ ⑥	Inherits multiple upstream nodes on the computational graph, rather than redefining leaf nodes.

Fig. 6. Differential Method summary, corresponding to the treatment of non-differentiable operators in differential kinematics (red circle in Fig. 4). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

sufficiently small:

$$G(x)=\begin{cases}-\frac{(x-(-1+\delta))}{\sqrt{1-(-1+\delta)^2}}+\cos^{-1}(-1+\delta), \text{if } x\le -1+\delta,\\ \cos^{-1}(x), \qquad \text{if } -1+\delta<x<1-\delta,\\ -\frac{(x-(1-\delta))}{\sqrt{1-(1-\delta)^2}}+\cos^{-1}(1-\delta), \text{if } x\ge 1-\delta.\end{cases} \tag{19}$$

In our formula, we employ the concept of tangent lines for construction. This adjustment is made to ensure that within the domain of the *arccos* function definition, it operates normally and contributes to defining the loss function. At the same time, the unsatisfactory part outside the domain of definition can have such a large slope that it is penalized by the network preference without interrupting the computational graph. For this purpose, it is imperative to guarantee that the newly constructed functions exhibit identical values of the first-order derivatives at the points of discontinuity.

This ensures that even if the value passed in from the previous node is out of the definition domain, the whole computation engine will not be terminated due to error reporting. In practice, this change not only ensures the program's normal execution but also allows us to incorporate the out-of-domain region into the loss composition. This is beneficial because this region typically has a steeper slope, resulting in a larger contribution. During backpropagation, the out-of-definition domain is penalized preferentially, represented as L_k^{outdom} in Eq. (22).

5.3.2. Training core: Unique definition of loss

The solution to the second difficulty is inspired by the fact that, when the loss is not based on the difference between the output and the labels, the only pointers to training come from the various types of anomalies and their defined losses during the computation. It is crucial not only to inform the network whether its outputs comply with kinematic constraints but also to effectively communicate the degree of deviation from those constraints.

Core Idea: Penalize all differential operators that may yield abnormal results or fall outside the ideal solution space. Losses should reflect deviations or degrees of abnormality.

If we integrate these losses into the guidance of the neural network, it will generate outputs capable of traversing all computational pathways, thereby ultimately leading to a valid solution. Naturally, this is a qualitative analysis, whereas the quantitative calculation of the predictors' losses is provided by the following equations. For clarity, the various losses, their corresponding meanings, and the informed prior knowledge are summarized in Table 2.

For CMP:

$$\mathcal{L}^{CMP}=\frac{1}{N}\sum_{k=1}^{N}(\mathbb{U}\cdot\delta_{illroot}\mathcal{L}_k^{illroot}+\mathbb{U}\cdot\delta_{outdom}\mathcal{L}_k^{outdom} +\delta_{illsolu}\mathcal{L}_k^{illsolu}+\delta_{idesolu}\mathcal{L}_k^{idesolu}) \tag{20}$$

Where δ represents the weighting relationship between the different parts of the losses, which will determine which losses are preferentially penalized during backpropagation. The $\mathcal{L}_k^{illroot}$ as well as $\mathcal{L}_k^{outdom}$ denote exactly the penalties for the location prediction network data on the wrong branch during the differentiable computation process. They guarantee that the inverse operation has the result as shown in Eq. (5), even if the solved angle value is illegal. Denote certain intermediate variables in the computation of q_j^i as the symbol q^{pre}, the specific formula definitions are as follows:

To ensure that the $f(q^{pre})=\sqrt{q^{pre}}$ is legalized:

$$\mathcal{L}_k^{illroot}=\begin{cases}0, & \text{if } q^{pre}\geqslant 0\\ -q^{pre}, & \text{if } q^{pre}<0\end{cases} \tag{21}$$

To ensure that the $G(q^{pre})$ in Eq. (19) is legalized:

$$\mathcal{L}_k^{\text{outdom}}=\begin{cases}0, & \text{if } q^{pre}\in[-1,1]\\ |q^{pre}|-1, & \text{if } q^{pre}\notin[-1,1]\end{cases} \tag{22}$$

For numerically illegal angle values beyond $[-\pi,\pi]$, we use $\mathcal{L}_k^{illsolu}$ to penalize them. Obviously, if there exists any one of the eight sets of solutions where all the angular values of the angular solutions are within the legal range, the loss will be defined as 0, that is, $\mathcal{L}_k^{idesolu}$. The specific formula definitions are as follows:

$$\mathcal{L}_k^{\text{illsolu}}=\begin{cases}0, & \text{if } q_j^i\in[-\pi,\pi]\\ |q_j^i|-\pi, & \text{if } q_j^i\notin[-\pi,\pi]\end{cases} \tag{23}$$

$$\mathcal{L}_k^{\text{idesolu}}=\begin{cases}0, & \text{if } \exists i\in[1,8],\forall j\in[1,6],q_j^i\in[-\pi,\pi]\\ L_k^{\text{illsolu}}\end{cases} \tag{24}$$

In particular, for the inverse kinematics computation process included in the CMP, we specifically define $\mathbb{U}$ in Eq. (20). When it takes 1, the loss function takes into account the loss from all the errors in the inverse kinematics computation process that deviates from the correct data flow, which is the task-specific loss. When it takes 0, we only define the loss and guide the training for one or more sets of illegal values in the final angular solution. Apparently, the former is more relevant for the present application scenario and the loss is defined more strictly, while the latter is an attempt of ours regarding the definition of loss in general problems and generalized scenarios.

For FMP:

$$\mathcal{L}^{FMP}=\frac{1}{N}\sum_{k=1}^{N}(\delta_{pre\_error}\mathcal{L}_k^{pre\_error}+\delta_{distance}\mathcal{L}_k^{distance} +\delta_{orien}L_k^{orien}) \tag{25}$$

$$\mathcal{L}_k^{\text{pre\_error}}=\begin{cases}\text{MSELoss}(\mathbb{FK}({}^{R}C_{pred}^{FMP}, {}^{P}C_{pred}^{FMP})-H_{\text{tar}}), & \text{if } \Delta d\geq 1 \text{ mm}\\ 0, & \text{if } \Delta d<1 \text{ mm}\end{cases} \tag{26}$$

$$\mathcal{L}_k^{\text{distance}}=\begin{cases}\text{MSELoss}({}^{P}C_{pred}^{FMP},\mathcal{T}_{target}), & \text{if } \mathcal{T}_{target}\notin\mathcal{W}\\ 0, & \text{if } \mathcal{T}_{target}\in\mathcal{W}\end{cases} \tag{27}$$

$$\mathcal{L}_k^{\text{orien}}=\begin{cases}\sum_{i=x,y,z}\left|\arctan 2\left(\sin(\Delta\theta_i),\cos(\Delta\theta_i)\right)\right|, & \text{if } |\Delta\theta_i|\geq 0.1 \text{ rad}\\ 0, & \text{if } |\Delta\theta_i|<0.1 \text{ rad}\end{cases} \tag{28}$$

The loss function is defined by the weighting coefficients δ and three sub-losses. For FMP, we allow the nine output parameters for whole-body control to have a distance of less than Δd from the target position H_{tar} after being constrained by forward kinematics.

Table 2
Meaning of customized loss and their corresponding informed prior knowledge.

Loss components	Meaning and function	Informed prior knowledge
$\mathcal{L}_k^{illroot}$	Illegal Root Loss: Penalizes when a node in the computation graph results in a negative value under a square root.	Non-negative values under the square root.
$\mathcal{L}_k^{outdom}$	Out of Domain Loss: Penalizes when intermediate variables exceed the operator's defined domain.	Domain constraints on independent variables.
$\mathcal{L}_k^{illsolu}$	Illegal Solution Loss: Penalizes when the inverse kinematics has solutions, but the joint angles contain illegal values.	Joint angle limits.
$\mathcal{L}_k^{idesolu}$	Ideal Solution Loss: Do not penalizes when at least one valid solution for inverse kinematics exists.	At least one valid solution.
$\mathcal{L}_k^{pre\_error}$	Predicted Result Error Loss: Penalizes the prediction error after forward kinematics of the FMP predicted configuration.	High-precision fitting within 1mm.
$\mathcal{L}_k^{distance}$	Distance Error Loss: Penalizes errors in the appropriateness of the predicted chassis position range by the FMP.	Goal within the workspace.
$\mathcal{L}_k^{orien}$	Orientation Error Loss: Penalizes errors in the end-effector's orientation under the FMP predicted joint configuration.	Maximum allowable orientation angle of 0.1 rad.

Table 3
Training information for each model.

Model	Composition	Loss function	Hyperparameter optimization
NN Regression	input_layer = 6 hidden_layer = 50 Dropout hidden_layer = 50 output_layer = 6	MSE	Ray.tune
CMP1	input_layer = 6 hidden_layer = 50 Dropout hidden_layer = 50 output_layer = 3 $\mathbb{IK}$	$\mathcal{L}^{CMP}$ $\mathbb{U} = 1$	Ray.tune
CMP2	input_layer = 6 hidden_layer = 50 Dropout hidden_layer = 50 output_layer = 3 $\mathbb{IK}$	$\mathcal{L}^{CMP}$ $\mathbb{U} = 0$	Ray.tune
FMP	input_layer = 6 hidden_layer = 50 Dropout hidden_layer = 50 output_layer = 9 $\mathbb{IK}$	$\mathcal{L}^{FMP}$	Ray.tune

The degree of deviation is controlled by $\mathcal{L}_k^{pre\_error}$. $\mathcal{L}_k^{distance}$ and $\mathcal{L}_k^{orien}$ impose constraints on the chassis position and the end-effector's orientation, respectively. Here, $\Delta\theta_i$ represents the error between the end-effector's orientation, calculated through forward kinematics, and the target object's orientation in the x, y, and z directions.

5.3.3. Stochasticity guarantees

To tackle the third challenge, an additional Dropout layer was incorporated into the position prediction network of the RobKiNet, as illustrated in Table 3.

Core Idea: Introduce stochasticity into the forward propagation process without affecting the distribution of sampled results within the ideal solution space, C^*.

In conventional deep learning networks, Dropout layers prune neurons randomly during training to prevent overfitting. However, the Dropout layers introduced here inject minimal stochasticity during testing, and they do not significantly impact the network's accuracy, as shown in the next section. This approach is inspired by the principle of introducing stochasticity through the temperature parameter [57], commonly used in large language models.

During testing, the Dropout layer remains active. When the differentiable kinematics calculation engine detects unsatisfactory outputs from the trained model, the testing process is restarted, and the RobKiNet generates a fresh set of solutions. Since no loss is computed and no backpropagation is performed during this phase, the Dropout layer plays a crucial role in introducing stochasticity. Each regenerated result is obtained from the well-trained model, and any prior failures do not influence the current outcome. Stochasticity and accuracy are often inversely related. The level of stochasticity should be carefully controlled during both training and testing to avoid excessive randomness. It must ensure that the sampled solutions fluctuate within our ideal solution space C^*.

6. Experiments and results

6.1. Experiments setup

In order to validate the effectiveness of the proposed method in this work, we use the disassembly of EOL-EVBs bolts as a practical application scenario to explore the performance demonstrated by the CMP and the FMP in terms of training and testing.

We conducted comparative experiments to demonstrate the difference in training outcomes when incorporating the computational engine for differentiable kinematics into the computational graph compared to a single ANN (the NN Regression method). The specific

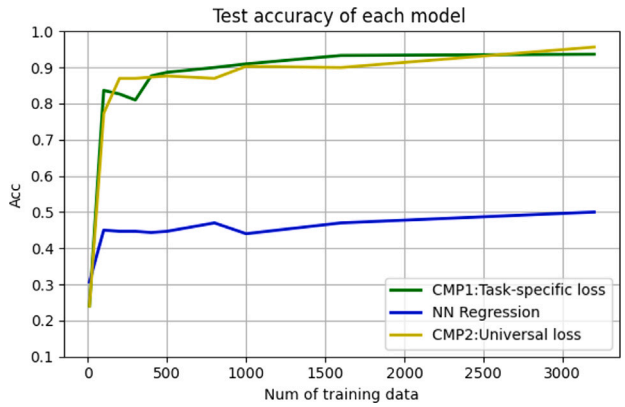


Fig. 7. For decoupled control: Test accuracy after training with different data amounts. The data volume ranges from 50 to 3200 groups.

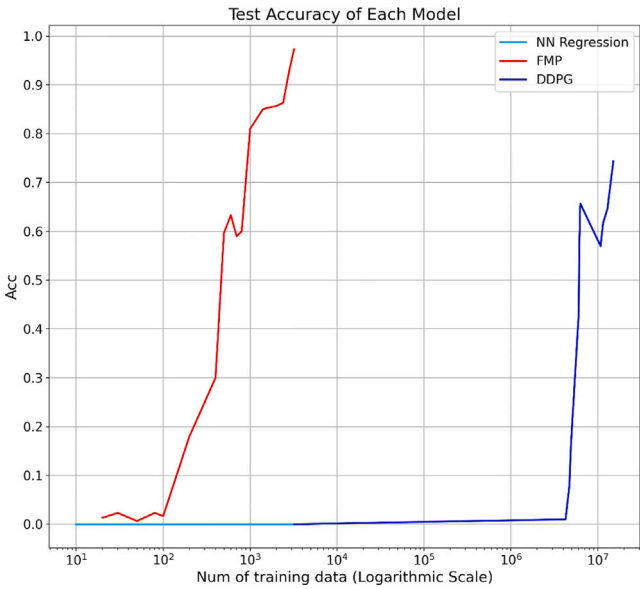


Fig. 8. For whole-body control: Test accuracy after training with different data amounts. Supervised learning does not apply to the configuration prediction tasks and cannot give joint angle predictions within the acceptable error Δd (Light Blue). We further implemented DDPG to achieve predictions for comparison (Dark Blue)[2]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

experimental training information is shown in Table 3. We built the network model based on PyTorch and provided the training of NN Regression with the end-effector position required for the disassembly task in the same coordinate system with a chassis position that can accomplish the disassembly task. The loss between the network output and the ground truth is defined by MSEloss. For the training of the predictors, the input data is a single end-effector pose required for the disassembly task, and its loss can be categorized into different cases according to the definition of Eqs. (20) and (25). CMP1 represents a more stringent task-specific loss, while CMP2 represents a more general loss based on the use of a differentiable method as described in the previous section.

To evaluate model performance under varying data volumes effectively, we designed twenty different data volume sets, ranging from a minimum of 20 groups to a maximum of 3200 groups, for training and testing the three models. To ensure that the training hyperparameters are optimal under different data volumes and model conditions, we use Ray-tune [58] to search for optimal hyperparameters. It is guaranteed that each network model is randomly selected at least 30 times for hyperparameters under each data volume, and the network models trained with optimal hyperparameters are recorded with their test results. In the training process, the weighting coefficients δ are not arbitrarily assigned but are designed to balance the magnitudes of different loss terms after normalization, ensuring that no single term dominates the optimization process. We also employ Ray-tune to search for suitable coefficients within a predefined range, where each coefficient is sampled from a bounded interval to maintain comparable scales among loss components. Empirically, we observe that the model is not highly sensitive to the exact values of δ within a reasonable range [0.1, 2]. Due to the incorporation of explicit kinematic constraints, the training process exhibits stable convergence behavior, and similar performance can be achieved across a range of coefficient settings.

6.2. Experiments details

We consider a sampling point to be valid when it satisfies the robot's kinematic constraints, is suitable for the workspace, and does not lead to robot singularities. The ideal solution and accuracy have already been clearly defined in Section 3.

Figs. 7 and 8 displays the test set results obtained from training different models with varying amounts of data, with the CMP achieving the highest test accuracy at 96.67% and the FMP achieving 98%. As we defined the accuracy metric earlier, supervised learning methods, such as NN Regression, do not perform well in terms of accuracy for the chassis motion prediction tasks. In the comparative experiment with the CMP, NN Regression needs larger data support in training its parameters in a more optimal direction. The existing volume of data falls short of meeting the parameter refinement requirements for NN Regression, resulting in a test accuracy of approximately 50%. In contrast, the CMP demonstrates an impressive accuracy exceeding 90% with only 1000 sets of training data.

Meanwhile, in experiments related to FMP (Fig. 8), we found that NN Regression is not suitable for such configuration prediction tasks. We prepared over 5000 sets of input values and corresponding labels, adhering to kinematic constraints for supervised training, hoping that NN Regression would learn the underlying implicit whole-body control mapping represented by the dataset. Rigorous experimental results showed that while NN Regression could learn simple position mapping relationships, it was unable to predict the nine valid parameter configurations required for whole-body control in a single instance (in Eq. (17)), particularly within the required error (in Eq. (18)). This further demonstrates that supervised learning is not well-suited to solving sampling problems, as there is no fixed functional mapping for valid solutions.[2]

We further attempted to use the Deep Deterministic Policy Gradient (DDPG) method [59] from DRL as a replacement for NN regression to address this prediction problem. The substantial data requirement is justified by our goal for the agent to learn the underlying real-world constraints. By training an Actor-Critic (AC) architecture, we aim to maximize the expected reward in the end. This approach is fundamental to how deep reinforcement learning addresses the challenges associated with sampling in continuous solution spaces. In scenarios characterized by non-fixed mapping relationships, the strategy involves converging the policy space to seek the optimal solution. We constructed an agent comprising two actors and two critics, totaling four networks, and utilized up to 15 million data samples, ultimately achieving an accuracy

[2] In Fig. 8, the light blue represents the NN regression method. We tested six different network architectures, three optimizers, and three kinematic representation relationships, each under 20 different data quantities.

Table 4
Experimental setup introducing stochasticity.

	Dropout (Training)	Dropout (Testing)	Epoches	Dropout rate
Case1	Deactivated	Deactivated	400	None
Case2	Activated	Activated	1000	2%-15% (use Ray-tune)
Case3	Deactivated	Activated	400	Consistent with case2 optimal result

Table 5
CMP performance: sampling times and time costs for each method.

	Average samples (times)	Single sampling time (ms)	Single kinematic simulation (ms)	Total time for one speculation (ms)	Time-reduction factor
RS	31.04	0.296		5425.05	1 × (base)
EBS	5.80	0.473		1014.73	5.34 ×
NN Regression	3.06	0.315	174.48	534.87	10.14 ×
CMP1	1.28	0.339		223.77	24.24 ×
CMP2	1.42	0.312		248.20	21.86 ×

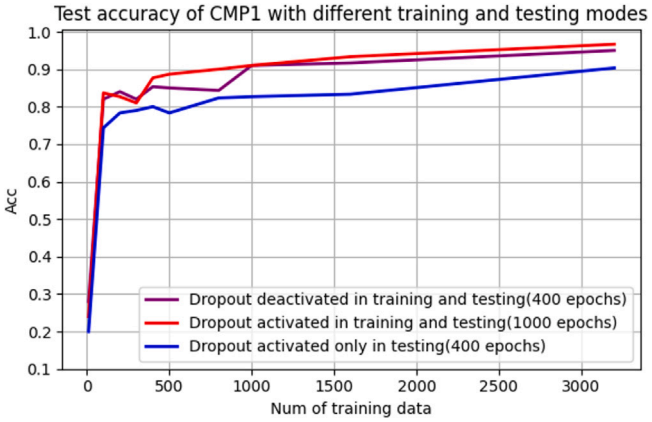


Fig. 9. Comparative experiments on CMP1 with the introduction of stochasticity methods.

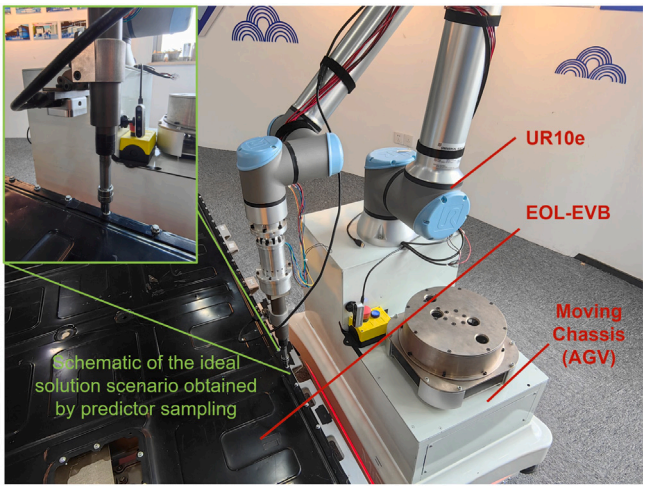


Fig. 10. Hardware platform.

of 74.33%.[3] In contrast, FMP once again demonstrated remarkable prediction efficiency, being able to output the entire system's parameter configuration in a single forward propagation while satisfying physical constraints.

Another conclusion is evident from Figs. 7 and 8: the predictors defined by RobKiNet demonstrate exceptional data efficiency. Compared to NN regression or deep reinforcement learning methods, CMP requires only 1/71 of the data, and FMP only 1/15052 of the data to achieve the same prediction accuracy. This remarkable data efficiency can be attributed to the integration of differentiable kinematics algorithms as prior knowledge within the framework.

It is important to emphasize that the experimental results depicted in Figs. 7 represent explorations into the fundamental performance of the predictors, undertaken without the introduction of stochasticity. Compared to RS methods, the CMP and the FMP demonstrate significant advantages in requiring less training data and fewer epochs. However, when undesirable failure cases occur, the fixed model requires multiple network models that have completed training to work together to ensure a 100% task completion rate. Thus, we introduce the network stochasticity methods described in Section 5.3 for a more in-depth comparative experimental exploration by controlling the degree of stochasticity during network training and testing. In this way, we aim to achieve a 100% success rate by relying solely on the combination of a single pre-trained predictor with the feedback mechanism. Table 4 shows the comparison experiments conducted with CMP1 as the base network. Fig. 9 shows the performances of the three methods with different datasets.

Through experiments, it is found that when the stochasticity method is introduced into the training and testing process, the accuracy has more obvious fluctuations in adjacent epochs. The reason for this is that the dropout's culling of hidden neurons is completely randomized, which prevents the network from overly relying on certain neurons to get the desired output. Therefore, when stochasticity is involved in training, the CMP needs to undergo more training iterations to achieve a similar accuracy as in case 1, and the introduction of stochasticity makes the individual hidden neurons more homogeneous. The experimental results show that after introducing the stochasticity method proposed in this paper, the CMP1 achieves a single prediction accuracy of 96.67%, but it takes a longer time and training epochs to reach convergence. The accuracy of this method is significantly higher than that of the method in case 3 where stochasticity is introduced only in testing. Therefore, despite the fact that the addition of stochasticity increases the number of training epoch, we still favor this approach for practical deployment.

To further validate the advantages of the RobKiNet, we provided identical disassembly bolt poses for different methods. Except for the networks, we choose basic sampling methods random sampling (RS) [19] and efficient biased sampling (EBS) [33] for clearer comparison with CMP (results in Table 5). We also combine these sampling methods with the traditional Jacobian iteration (Jacobian matrix

[3] In Fig. 8, the dark blue indicates DDPG, which is an effective offline deep reinforcement learning method designed for continuous control problems. we employed DDPG for predictions after 3200 data samples, to the maximum data amount of initial data (204800) and resampled replay buffer update data (max 29000 epoch × 512).

Type of task	Scenario& Requirements	Inputs	Number of RobKiNet-guided moves	Number of bolts dismantled in a single task	Success rate of single sampling that satisfies kinematic constraints	Success rate of total prediction through muti-sampling
Task 1	Approach the target bolt from any position at a distance to disassemble it.	Target bolt pose	1	4	95.0%	100%
Task 2	After dismantling the bolt in the safe working space at the current position, move to the appropriate position to continue dismantling the next bolt.	No inputs (The NSTAMP framework called RobKiNet to ensure its autonomy and continuity)	1	8	95.23%	100%
Task 3	Continuous extensive disassembly.	Only the starting pose of the first bolt is required	4 (on average)	20 (consecutive bolts on one side)	96.25%	100%
Task 4	Continuous disassembly with multiple scheduling over long distances.	The first bolt pose in the scheduling area	2	5	97.50%	100%

Fig. 11. Real-world experiments information and results.

Table 6
FMP performance: solving time and analysis.

	Total time for one speculation (ms)	Time-reduction factor	Analysis
RS	30000 (meet the maximum and got none result)	1 × (base)	The solution space has too many dimensions, leading to no solution within the given time limit.
RS + Jacobian	6732.77	4.46 ×	Gradient-based methods guide the search but take longer.
EBS + Jacobian	2455.06	12.22 ×	The iteration is more efficient but yields only a single solution.
DDPG	311.08	96.44 ×	Excessive data magnitude and high training costs (Fig. 8).
FMP	195.76	153 ×	Accurately and efficiently provides system configurations to achieve whole-body control.

method) [34] to generate solutions suitable for whole-body control and compare the performance with FMP (results in Table 6). We present 300 sets of data and compute the mean sampling times, the sampling time costs, and the time-reduction factor. Compared to random sampling, the CMP achieved a 24.24-fold time reduction, while the FMP achieved a 153-fold time reduction.

6.3. Real device experiments in disassembly scenarios

Fig. 10 is our experimental hardware platform, which consists of an AGV trolley with a mounted 6-axis collaborative robot UR10e working on a to-be-disassembled EOL-EVB.

We deployed the CMP and the FMP for experiments in real-world scenarios. The RobKiNet is utilized in the NeuroSymbolic TAMP framework [27] and acts as the primitive “Approach” in guiding the entire autonomous movement of the AGV chassis. Within this framework, the NeuroSymbolic task planning layer accounts for macro-level constraints. When “Approach” is invoked, its motion planning layer must perform planning under multi-level constraints to ensure that the disassembly posture is appropriate.

To comprehensively assess the performance of RobKiNet across various usage scenarios, we formulated four tasks. The corresponding task scenarios and requirements are illustrated in Fig. 11. Through over 80 times of real-machine experiments, we observed that the RobKiNet accurately samples positions in actual disassembly scenarios with a 100% success rate. It is worth noting that individual sampling attempts may occasionally fail to satisfy kinematic constraints due to prediction deviations or unmodeled factors such as potential collisions. In such cases, the TAMP framework performs rapid feasibility checking and triggers re-sampling, which typically converges within a small number of trials (on average less than 2), ensuring robust overall task execution. Demonstration videos are accessible on the paper’s website .

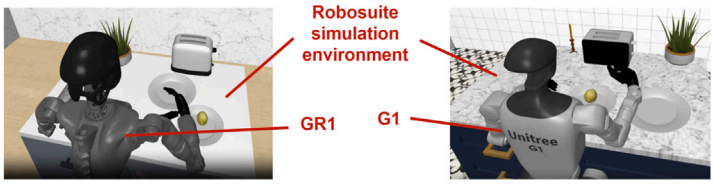
6.4. Simulation-based generalization study

To further evaluate the generalization capability of the proposed RobKiNet across different robotic configurations, we conducted additional simulation experiments on two humanoid robotic platforms, namely the Fourier GR1 and the Unitree G1, within the robosuite simulation environment. Unlike the industrial mobile manipulator used in the real-world experiments, these humanoid robots exhibit significantly different kinematic structures. In particular, the GR1 satisfies Pieper’s criterion, while the G1 does not, representing two fundamentally different classes of inverse kinematics characteristics. Each robot is equipped with dual 7-DoF arms, introducing higher redundancy and increased kinematic complexity. In the simulation, we constructed 20 randomized tabletop scenarios with varying object positions. For each scenario, 15 task-level target poses were defined. The RobKiNet was directly applied without structural modification, and tasked with predicting feasible joint configurations under task-level constraints. When transferring to different robotic platforms, only the corresponding kinematic parameters (e.g., DH parameters) are updated in the differentiable kinematic model, while the network structure remains unchanged.

As summarized in Fig. 12, the proposed method achieves high-precision configuration prediction across both platforms. For the GR1, the average position error is below 0.6 mm with an orientation error of approximately 0.43°, while for the G1, the position error remains below 0.8 mm with an orientation error around 0.51°. These results demonstrate that RobKiNet maintains strong performance even when applied to robots with substantially different kinematic properties. Overall, this study indicates that the proposed framework is not limited to a specific robot morphology, but can generalize effectively to diverse kinematic structures, supporting its applicability in broader manipulation scenarios.

7. Conclusions

This paper proposes RobKiNet, a kinematics-informed neural network that embeds differentiable forward and inverse kinematics into



Robot Platform	Kinematic Property	Random Scenarios / Objects	DoF	Task-Level Constraints	Position Error (avg over 900 trials)	Orientation Error (avg over 900 trials)
GR1	Satisfies Pieper's criterion	20	7DOF /arm	15 target poses	<0.6mm	~ 0.43°
G1	Does not satisfy Pieper's criterion				<0.8mm	~ 0.51°

Fig. 12. Simulation-based generalization evaluation on different humanoid robotic platforms.

the learning process for efficient robot configuration sampling in robot TAMP. By incorporating kinematic constraints into the network structure and training objective, the proposed method enables direct prediction of feasible configurations under global task constraints, significantly improving sampling efficiency and data efficiency compared to traditional methods. Experimental results in both real-world and simulation scenarios demonstrate that RobKiNet can achieve high-accuracy predictions and robust performance, while substantially reducing the number of required samples.

Despite these advantages, several limitations remain. First, the current framework does not explicitly incorporate collision constraints within the network, and instead relies on external feasibility checking and re-sampling. Second, the prediction accuracy may degrade under extremely strict tolerance requirements due to approximation errors. Third, although promising generalization results have been observed, broader validation across more diverse robotic platforms and tasks is still needed.

In future work, we plan to extend the framework by integrating additional forms of robotic knowledge, such as collision-aware constraints, dynamics, and control-related factors, directly into the learning process. We also aim to explore more scalable loss formulations and further improve generalization across heterogeneous robotic systems. These developments are expected to enhance the applicability and robustness of the proposed approach in more complex real-world scenarios.

CRediT authorship contribution statement

Yanlong Peng: Writing – original draft, Visualization, Validation, Methodology. **Zhigang Wang:** Writing – original draft, Methodology, Conceptualization. **Yisheng Zhang:** Visualization, Data curation. **Pengxu Chang:** Validation, Investigation. **Ziwen He:** Writing – review & editing, Visualization. **Kai Gu:** Writing – review & editing, Validation. **Hongshen Zhang:** Writing – review & editing. **Ming Chen:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors express their sincerest thanks to the Ministry of Industry and Information Technology of China for financing this research within the program "2021 High Quality Development Project (TC210H02C)" and support from Neuro-Symbolic Al HOME Satellite Laboratories.

Data availability

Data will be made available on request.

References

[1] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, Tomás Lozano-Pérez, Integrated task and motion planning, Annu. Rev. Control. Robot. Auton. Syst. 4 (1) (2021) 265–293.

[2] Asim Munawar, Giovanni De Magistris, Tu-Hoa Pham, Daiki Kimura, Michiaki Tatsubori, Takao Moriyama, Ryuki Tachibana, Grady Booch, Maestrob: A robotics framework for integrated orchestration of low-level control and high-level reasoning, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 527–534.

[3] Jeongmin Jeon, Hong-ryul Jung, Francisco Yumbla, Tuan Anh Luong, Hyungpil Moon, Primitive action based combined task and motion planning for the service robot, Front. Robot. AI 9 (2022) 713470.

[4] Caelan Reed Garrett, Tomas Lozano-Perez, Leslie Pack Kaelbling, Ffrob: Leveraging symbolic planning for efficient task and motion planning, Int. J. Robot. Res. 37 (1) (2018) 104–136.

[5] Zhutian Yang, Caelan Reed Garrett, Dieter Fox, Sequence-based plan feasibility prediction for efficient task and motion planning, 2022, arXiv preprint arXiv: 2211.01576.

[6] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, Tomás Lozano-Pérez, Learning compositional models of robot skills for task and motion planning, Int. J. Robot. Res. 40 (6–7) (2021) 866–894.

[7] Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, Tomás Lozano-Pérez, Learning to guide task and motion planning using score-space representation, Int. J. Robot. Res. 38 (7) (2019) 793–812.

[8] Sergio Cebollada, Luis Payá, María Flores, Adrián Peidró, Oscar Reinoso, A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data, Expert Syst. Appl. 167 (2021) 114195.

[9] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, Michael C Yip, Motion planning networks: Bridging the gap between learning-based and classical motion planners, IEEE Trans. Robot. 37 (1) (2020) 48–66.

[10] Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, Jun Morimoto, Deep learning, reinforcement learning, and world models, Neural Netw. 152 (2022) 267–275.

[11] Marc Hanheide, Moritz Göbelbecker, Graham S Horn, Andrzej Pronobis, Kristoffer Sjöö, Alper Aydemir, Patric Jensfelt, Charles Gretton, Richard Dearden, Miroslav Janicek, et al., Robot task planning and explanation in open and uncertain worlds, Artificial Intelligence 247 (2017) 119–150.

[12] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, Tomás Lozano-Pérez, Integrated task and motion planning, Annu. Rev. Control. Robot. Auton. Syst. 4 (2021) 265–293.

[13] Enjin Zhang, Yahui Gan, Xianzhong Dai, End-point steady control for whole-body control mobile manipulators by null space tunation, in: 2023 International Conference on Advanced Robotics and Mechatronics, ICARM, IEEE, 2023, pp. 1071–1076.

[14] Smail Ait Bouhsain, Rachid Alami, Thierry Simeon, Learning to predict action feasibility for task and motion planning in 3d environments, in: 2023 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2023, pp. 3736–3742.

[15] Haitao Zeng, Xinhang Song, Shuqiang Jiang, Multi-object navigation using potential target position policy function, IEEE Trans. Image Process. (2023).

[16] Rutong Dou, Shenbo Yu, Wenyang Li, Peng Chen, Pengpeng Xia, Fengchen Zhai, Hiroshi Yokoi, Yinlai Jiang, Inverse kinematics for a 7-DOF humanoid robotic arm with joint limit and end pose coupling, Mech. Mach. Theory 169 (2022) 104637.

[17] Xinyang Tian, Qinhuan Xu, Qiang Zhan, An analytical inverse kinematics solution with joint limits avoidance of 7-DOF anthropomorphic manipulators without offset, J. Franklin Inst. 358 (2) (2021) 1252–1272.

[18] Ioan A. Sucan, Mark Moll, Lydia E. Kavraki, The open motion planning library, IEEE Robot. Autom. Mag. 19 (4) (2012) 72–82.

[19] Caelan Reed Garrett, Tomás Lozano-Pérez, Leslie Pack Kaelbling, Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, 2020, pp. 440–448.

[20] Zachary Kingston, Mark Moll, Lydia E. Kavraki, Sampling-based methods for motion planning with constraints, Annu. Rev. Control. Robot. Auton. Syst. 1 (1) (2018) 159–185.

[21] Benjamin Burger, Phillip M Maffettone, Vladimir V Gusev, Catherine M Aitchison, Yang Bai, Xiaoyan Wang, Xiaobo Li, Ben M Alston, Buyi Li, Rob Clowes, et al., A mobile robotic chemist, Nature 583 (7815) (2020) 237–241.

[22] Emrah Akin Sisbot, Luis F Marin-Urias, Rachid Alami, Thierry Simeon, A human aware mobile robot motion planner, IEEE Trans. Robot. 23 (5) (2007) 874–883.

[23] Maria Vittoria Minniti, Farbod Farshidian, Ruben Grandia, Marco Hutter, Whole-body mpc for a dynamically stable mobile manipulator, IEEE Robot. Autom. Lett. 4 (4) (2019) 3687–3694.

[24] Mantian Li, Zeguo Yang, Fusheng Zha, Xin Wang, Pengfei Wang, Ping Li, Qinyuan Ren, Fei Chen, Design and analysis of a whole-body controller for a velocity controlled robot mobile manipulator, Sci. China Inf. Sci. 63 (2020) 1–15.

[25] Garcez Artur d'Avila, Raedt Luc De, Földiak Peter, Hitzler Pascal, Icard Thomas, Kühnberger Kai-Uwe, Miikkulainen Risto, et al., Neural-symbolic learning and reasoning: contributions and challenges, in: AAAI Spring Symposium Series, AAAI Press, Palo Alto, California, USA, 2015, pp. 20–23.

[26] Bo Zhang, Jun Zhu, Hang Su, Toward the third generation artificial intelligence, Sci. China Inf. Sci. 66 (2) (2023) 121101.

[27] Hengwei Zhang, Yisheng Zhang, Zhigang Wang, Shengmin Zhang, Huaicheng Li, Ming Chen, A novel knowledge-driven flexible human–robot hybrid disassembly line and its key technologies for electric vehicle batteries, J. Manuf. Syst. 68 (2023) 338–353.

[28] Yanlong Peng, Zhigang Wang, Yisheng Zhang, Shengmin Zhang, Nan Cai, Fan Wu, Ming Chen, Revolutionizing battery disassembly: The design and implementation of a battery disassembly autonomous mobile manipulator robot (BEAM-1), 2024, arXiv preprint arXiv:2407.06590.

[29] Hengwei Zhang, Hua Yang, Haitao Wang, Zhigang Wang, Shengmin Zhang, Ming Chen, Autonomous electric vehicle battery disassembly based on NeuroSymbolic computing, in: Proceedings of SAI Intelligent Systems Conference, Springer, 2022, pp. 443–457.

[30] Yisheng Zhang, Hengwei Zhang, Zhigang Wang, Shengmin Zhang, Huaicheng Li, Ming Chen, Development of an autonomous, explainable, robust robotic system for electric vehicle battery disassembly, in: 2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM, IEEE, 2023, pp. 409–414.

[31] Brad Hamner, Seth Koterba, Jane Shi, Reid Simmons, Sanjiv Singh, An autonomous mobile manipulator for assembly tasks, Auton. Robots 28 (2010) 131–149.

[32] Jun Huang, Duc Truong Pham, Ruiya Li, Mo Qu, Yongjing Wang, Mairi Kerin, Shizhong Su, Chunqian Ji, Omar Mahomed, Riham Khalil, et al., An experimental human-robot collaborative disassembly cell, Comput. Ind. Eng. 155 (2021) 107189.

[33] G. Kollios, D. Gunopulos, N. Koudas, S. Berchtold, Efficient biased sampling for approximate clustering and outlier detection in large data sets, IEEE Trans. Knowl. Data Eng. 15 (5) (2003) 1170–1187.

[34] Dechao Chen, Yunong Zhang, Shuai Li, Tracking control of robot manipulators with unknown models: A jacobian-matrix-adaption method, IEEE Trans. Ind. Inform. 14 (7) (2017) 3044–3053.

[35] Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y Siegel, Roland Hafner, et al., Learning agile soccer skills for a bipedal robot with deep reinforcement learning, Sci. Robot. 9 (89) (2024) eadi8022.

[36] Abdullah Al Redwan Newaz, Tauhidul Alam, Hierarchical task and motion planning through deep reinforcement learning, in: 2021 Fifth IEEE International Conference on Robotic Computing, IRC, 2021, pp. 100–105.

[37] Sixian Zhang, Xinhang Song, Yubing Bai, Weijie Li, Yakui Chu, Shuqiang Jiang, Hierarchical object-to-zone graph for object navigation, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 15130–15140.

[38] Ying Zhang, Guohui Tian, Jiaxing Lu, Mengyang Zhang, Senyan Zhang, Efficient dynamic object search in home environment by mobile robot: A priori knowledge-based approach, IEEE Trans. Veh. Technol. 68 (10) (2019) 9466–9477.

[39] Matteo Luperto, Michele Antonazzi, Francesco Amigoni, N Alberto Borghese, Robot exploration of indoor environments using incomplete and inaccurate prior knowledge, Robot. Auton. Syst. 133 (2020) 103622.

[40] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al., Do as i can, not as i say: Grounding language in robotic affordances, 2022, arXiv preprint arXiv:2204.01691.

[41] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al., Rt-2: Vision-language-action models transfer web knowledge to robotic control, in: Conference on Robot Learning, PMLR, 2023, pp. 2165–2183.

[42] Pao-Hsiung Chiu, Jian Cheng Wong, Chinchun Ooi, My Ha Dao, Yew-Soon Ong, CAN-PINN: A fast physics-informed neural network based on coupled-automatic–numerical differentiation method, Comput. Methods Appl. Mech. Engrg. 395 (2022) 114909.

[43] Zhiwei Fang, A high-efficient hybrid physics-informed neural networks based on convolutional neural network, IEEE Trans. Neural Netw. Learn. Syst. 33 (10) (2022) 5514–5526.

[44] Jiefeng Li, Chao Xu, Zhicun Chen, Siyuan Bian, Lixin Yang, Cewu Lu, Hybrik: A hybrid analytical-neural inverse kinematics solution for 3d human pose and shape estimation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 3383–3393.

[45] Jonas Degrave, Michiel Hermans, Joni Dambre, et al., A differentiable physics engine for deep learning in robotics, Front. Neurorobotics 13 (2019) 406386.

[46] Daniel Honerkamp, Tim Welschehold, Abhinav Valada, Learning kinematic feasibility for mobile manipulation through deep reinforcement learning, IEEE Robot. Autom. Lett. 6 (4) (2021) 6289–6296.

[47] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al., Pddl| the planning domain definition language, Tech. Rep. Tech. Rep. (1998).

[48] Sebastian Christodoulou, Uwe Naumann, Differentiable programming: Efficient smoothing of control-flow-induced discontinuities, 2023, arXiv preprint arXiv:2305.06692.

[49] Hai-Jun Liao, Jin-Guo Liu, Lei Wang, Tao Xiang, Differentiable programming tensor networks, Phys. Rev. X 9 (3) (2019) 031041.

[50] Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, Jonathan Ragan-Kelley, Differentiable programming for image processing and deep learning in halide, ACM Trans. Graph. (ToG) 37 (4) (2018) 1–13.

[51] Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckas, Elliot Saba, Viral B Shah, Will Tebbutt, A differentiable programming system to bridge machine learning and scientific computing, 2019, arXiv preprint arXiv:1907.07587.

[52] Weitao Zhou, Zhong Cao, Nanshan Deng, Xiaoyu Liu, Kun Jiang, Diange Yang, Dynamically conservative self-driving planner for long-tail cases, IEEE Trans. Intell. Transp. Syst. 24 (3) (2022) 3476–3488.

[53] Yucan Zhou, Qinghua Hu, Yu Wang, Deep super-class learning for long-tail distributed image classification, Pattern Recognit. 80 (2018) 118–128.

[54] Mohamed Khodeir, Ben Agro, Florian Shkurti, Learning to search in task and motion planning with streams, IEEE Robot. Autom. Lett. 8 (4) (2023) 1983–1990.

[55] Michael Bartholomew-Biggs, Steven Brown, Bruce Christianson, Laurence Dixon, Automatic differentiation of algorithms, J. Comput. Appl. Math. 124 (1–2) (2000) 171–190.

[56] Jacques Denavit, Richard S. Hartenberg, A kinematic notation for lower-pair mechanisms based on matrices, 1955.

[57] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al., A survey on evaluation of large language models, ACM Trans. Intell. Syst. Technol. 15 (3) (2024) 1–45.

[58] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, Ion Stoica, Tune: A research platform for distributed model selection and training, 2018, arXiv preprint arXiv:1807.05118.

[59] T.P. Lillicrap, Continuous control with deep reinforcement learning, 2015, arXiv preprint arXiv:1509.02971.