PyRoki: A Modular Toolkit for Robot Kinematic Optimization

Chung Min Kim*

Brent Yi* Hongsuk Choi

hoi Yi Ma

Ken Goldberg

Angjoo Kanazawa

UC Berkeley



https://pyroki-toolkit.github.io

Fig. 1: **PyRoki is a modular, extensible, and cross-platform toolkit for kinematic optimization.** We unify problems like inverse kinematics, trajectory optimization, and motion retargeting using composable kinematic variables and costs. PyRoki aims to support a broad variety of robots and tasks, and runs on CPU, GPU, and TPU.

Abstract—Robot motion can have many goals. Depending on the task, we might optimize for pose error, speed, collision, or similarity to a human demonstration. Motivated by this, we present PyRoki: a modular, extensible, and crossplatform toolkit for solving kinematic optimization problems. PyRoki couples an interface for specifying kinematic variables and costs with an efficient nonlinear least squares optimizer. Unlike existing tools, it is also cross-platform: optimization runs natively on CPU, GPU, and TPU. In this paper, we present (i) the design and implementation of PyRoki, (ii) motion retargeting and planning case studies that highlight the advantages of PyRoki's modularity, and (iii) optimization benchmarking, where PyRoki can be 1.4-1.7x faster and converges to lower errors than cuRobo, an existing GPUaccelerated inverse kinematics library.

I. INTRODUCTION

Numerical optimization is the standard solution for many tasks in robot kinematics. Using objectives like pose error [8], smoothness [9], and similarity to a human demonstration [6, 10] the robotics community has built diverse optimization software for tasks such as inverse kinematics (IK) [1, 3, 11–13], trajectory optimization [4, 5, 14–19], and

motion retargeting [6, 7, 10]. These tools are fast, mature, and widely adopted in both research and production.

Despite the shared structure of kinematic optimization, existing tools are fragmented. Implementations typically rely on task-specific C++ routines [1], CUDA kernels [5], or task-specific analytical Jacobians [3]. While these features can improve efficiency, they create barriers for incorporating new objectives. As shown in Table I, different task and robot hardware variations can require different tools. This fragmentation prevents tools from transferring between related problems. Low-level specialization also constrains computation: optimizers tend to be restricted to either only CPU or only GPU operation.

We present PyRoki (**Python Robot Kinematics**), a modular, extensible, and cross-platform toolkit for kinematic optimization. The core idea behind PyRoki is that many tasks that currently require disjoint tools—such as IK, trajectory optimization, and motion retargeting—solve similar optimization problems. PyRoki provides (i) an interface for specifying these problems using modular variable and cost function abstractions, (ii) the ability to efficiently solve them using a Levenberg-Marquardt optimizer, and (iii) a web-based visualizer for interactively tuning cost weights.

	Hardware			Supported Robots			Tasks			Features	
Method	CPU	GPU	TPU	Arm	Hand	Humanoid	IK	Traj. Opt.	Retargeting	Collision	Custom Costs
TracIK [1]	1	X	X	1	1	1	1	X	×	1	×
pink [2], mink [3]	1	×	X	1	1	1	1	×	×	1	X
TrajOpt [4]	1	×	X	1	1	1	X	1	×	1	X
cuRobo [5]	×	1	X	1	X	×	1	1	×	1	1
Dex-Retargeting [6]	1	×	X	×	1	×	1	×	1	×	X
H2O [7]	1	X	X	×	×	1	1	X	1	×	×
PyRoki	~	1	~	1	1	1	1	1	1	1	1

TABLE I: Comparison between PyRoki and a selection of existing kinematic optimization tools. This table summarizes the documented features and supported functionalities of different frameworks, highlighting the broad capabilities of PyRoki.

Furthermore, PyRoki is designed for efficient execution on CPU, GPU and TPU. The goal of PyRoki is to streamline kinematic optimization for robotics, just as deep learning frameworks like PyTorch [20] have made it easier to define and experiment with deep learning models.

The contributions of this paper are as follows:

- Toolkit. We present PyRoki, a modular toolkit for kinematic optimization for robotics. PyRoki directly supports standards like URDF [21], and can run on CPU, GPU, and TPU. It also includes a real-time webbased visualizer to explore robot behavior, such as adjusting the impact of cost functions and adding new scene visualizations.
- Applications. We demonstrate the effectiveness of PyRoki's unified framework across multiple tasks, including inverse kinematics, trajectory optimization, and motion retargeting for robot hands and humanoids.
- 3) Benchmarking. PyRoki solves optimization problems using a Levenberg-Marquardt solver. We evaluate PyRoki using both automatic and analytic Jacobians. For batched IK, PyRoki can be 1.4-1.7x faster than previous GPU-accelerated methods [5].

All code is released with an open-source license.

II. RELATED WORK

A. Inverse Kinematics

The goal of inverse kinematics (IK) is to recover a joint configuration that achieves a desired end-effector pose. Classical approaches focus on recovering analytical, closed-form solutions for this problem [22–27]. These approaches are efficient, but require assumptions on kinematic configuration. Recent approaches have therefore focused more on iterative optimization [1, 3, 5, 9, 11, 28–32], which is more flexible. Optimization enables secondary objectives for controlling redundant degrees of freedoms [33], both geometric [5, 30, 34] and learned [29] collision constraints, Critically, iterative optimization also generalizes across robot embodiments. The goal of PyRoki is to take this flexibility a step further. Rather than specialize for the IK task alone, PyRoki provides a kinematic optimization toolkit where IK can be defined using a subset of possible cost functions.

B. Trajectory Optimization

While inverse kinematics focuses on solving for a single joint configuration, the goal of trajectory optimization is to output a continuous sequence of joint configurations. Methods include trajectory optimization with collision costs [4, 14–16, 35, 36], applications with multiple goal sets [37], and simultaneous grasp and trajectory optimization [38, 39]. An important insight made by [36] is that trajectory optimization can be accelerated by exploiting structure: each configuration in the trajectory interacts with its temporal neighbors, leading to sparse optimization problems. Trajectory optimization can therefore benefit from sparse linear algebra routines [40–44], which PyRoki supports.

C. Motion Retargeting

Motion retargeting is the problem of transferring motion from a source embodiment to a target embodiment [34, 45] (e.g., from human to robot). Retargeting is critical step for many systems for robot teleoperation [6, 46] and for learning from human demonstrations [7, 47-49], but is made challenging by kinematic differences between embodiments-it requires balancing both similarity and physical plausibility objectives [50, 51]. Different application contexts also impose distinct computational requirements: for teleoperation, motion retargeting might need to run in real-time on a single CPU to control a single robot's motion. In contrast, batch processing on GPUs can dramatically accelerate offline processing for learning from large-scale datasets [52-55]. PyRoki aims to simplify optimization-based retargeting by making it easier to compose and tune costs, while introducing efficient parallelization on GPUs.

D. Modular Optimization Tools

Modular optimization tools have transformed numerous scientific and engineering disciplines. Frameworks like TensorFlow [56] and PyTorch [20] have revolutionized machine learning with automatic differentiation and hardware flexibility. JuMP [57] provides a mathematical optimization framework with interchangeable solvers, while Drake [58], Ceres Solver [43], g20 [44], GTSAM [59], miniSAM [60], CasADi [61], and ACADO [62] offer specialized capabilities for robotics, computer vision, and control applications. Inspired by these tools, PyRoki aims to simplify kinematic optimization for diverse robots, tasks, and compute platforms. We summarize relevant features in Table I.

III. PYROKI: MODULAR KINEMATIC OPTIMIZATION

We propose PyRoki, a robot kinematics library that is designed around three goals:

1) **Modular**: PyRoki separates optimization variables from cost functions, creating reusable components that work across different tasks. This lets the same objectives (e.g., collision avoidance, pose matching) apply to multiple tasks (e.g., inverse kinematics or trajectory optimization) without reimplementation.

2) *Extensible*: PyRoki supports rapid prototyping through automatic differentiation [63], computing Jacobians for user-defined cost functions, as well as a real-time interface for tuning cost weights. This simplifies custom optimization objectives, while also supporting analytical Jacobians when performance is critical.

3) Cross-Platform: PyRoki runs natively on CPUs, GPUs, and TPUs, enabling seamless scaling from single-robot optimization to parallel batch processing. Cross-platform capabilities can accelerate demanding applications like processing for large motion datasets or sampling-based planning [5].

There are three components of PyRoki that make these goals possible: a quasi-Newton optimizer, variable abstractions, and composable cost functions.

A. Solver Backbone

Kinematic optimization benefits from quasi-Newton approaches, which accelerate convergence using cost curvature approximations. PyRoki uses a Levenberg-Marquardt (LM) optimizer [64–68]. Our optimizer builds on prior work [41, 69, 70] and uses JAX [63], a high-level array programming interface that enables parallelization on CPU, GPU, and TPU. For efficiency, it automatically computes block-sparse Jacobian matrices; this is particularly advantageous for temporally sparse motion planning problems [36]. PyRoki's optimizer does not directly handle hard constraints, but we outline in Section III-C how we represent joint limits and contact avoidance with differentiable penalties.

B. Variable Abstractions

For modularity, PyRoki defines abstractions for common kinematic optimization variables. The core abstraction is the joint configuration variable, representing robot articulation states. We represent joint configurations as \mathbf{q} , with timesteps denoted as \mathbf{q}_t . Forward kinematics maps these configurations to poses, with $\mathbf{T}_{base \leftarrow i} = FK(\mathbf{q})_i$ representing the transform between the robot base and joint *i*. We support fixed, revolute, and prismatic joints, along with mimic joints commonly found in robot hands.

Kinematic variables can also be composed with SE(3) and SO(3) Lie group variables for representing poses and orientations. Operations like interpolation, composition, and pose error computation respect geometric structure without requiring users to implement complex manifold operations.



Interactive Web Viewer



Visualizing Manipulability Ellipse

Fig. 2: Interactive Web-based Robot Viewer. Users can tune weights for the different costs in real-time using a web interface (top), built on viser [71]. The viewer can also display the robot's configuration, set a goal, or modify the environment. The user can also add additional visualization, e.g., manipulability ellipse (bottom).

C. Cost Functions

PyRoki's modular design is based on the idea of composable cost functions. Different objectives can be combined, weighted, and reused across different optimization tasks.

In addition to supporting custom cost functions, PyRoki also comes with pre-implemented costs for common kinematic optimization tasks. We survey these in their residual forms below; in practice, these are also weighted. The LM optimizer minimizes the sum of squared residuals.

Joint Pose Cost. The joint pose cost \hat{c}_{pose} penalizes the difference between current and target joint poses:

$$\hat{c}_{\text{pose}}(\mathbf{q}, i, \mathbf{T}_{\text{base}\leftarrow\text{target}}) = \log(\mathbf{T}_{\text{base}\leftarrow\text{target}}^{-1}\mathbf{T}_{\text{base}\leftarrow i})$$

This cost can be quickly modified—for example, to include the base pose of a mobile manipulator. We evaluate this in Section V-B.

Joint Limit Cost The joint limit cost \hat{c}_{limit} penalizes joint values that are outside of their mechanical bounds:

$$\hat{c}_{\text{limit}}(\mathbf{q}) = \max(0, \mathbf{q} - \mathbf{q}_{\text{upper}}) + \max(0, \mathbf{q}_{\text{lower}} - \mathbf{q})$$

where q_{upper} and q_{lower} are the upper and lower mechanical limits of the joint, respectively.

Velocity Limit Cost The velocity limit cost \hat{c}_{vel} penalizes joint velocities that are outside of their mechanical limits:

$$\hat{c}_{\text{vel}}(\mathbf{q}, \dot{\mathbf{q}}) = \max(0, |\dot{\mathbf{q}}| - \dot{\mathbf{q}}_{\text{limit}} \cdot dt)$$

where $\dot{\mathbf{q}}_{i,\text{limit}}$ is the velocity limit of the joint, and dt is the time step between configurations.

Joint Regularization Cost. The joint regularization cost \hat{c}_{reg} encourages the solution to be close to a user-defined default pose, as redundant robots (e.g., 7-DoF arms) can reach multiple configurations.

$$\hat{c}_{\text{reg}}(\mathbf{q}) = \mathbf{q} - \mathbf{q}_{\text{reg}}$$

Smoothness Cost. The smoothness cost \hat{c}_{smooth} encourages small changes in joint positions, and is useful for generating smooth trajectories.

$$\hat{c}_{\text{smooth}}(\mathbf{q},t) = \mathbf{q}_t - \mathbf{q}_{t-1}$$

Similar costs can be written for acceleration and jerk minimization by approximating from q_t with the five-point stensil method [5].

Manipulability Cost. The manipulability cost \hat{c}_{manip} penalizes configurations where the robot is close to a singularity, and maximizes the Yoshikawa's manipulability measure [72].

$$\hat{c}_{\text{manip}}(\mathbf{q},i) = \left(\sqrt{\det(\mathbf{J}_i(\mathbf{q})\mathbf{J}_i(\mathbf{q})^T)} + \epsilon\right)^{-1}$$

1

where $\mathbf{J}_i(\mathbf{q})$ is the manipulator Jacobian of the robot at configuration \mathbf{q} for the *i*-th robot joint, and ϵ is a small constant to prevent division by zero.

Collision Avoidance Costs. Self and world collision costs are critical for generating feasible robot motion. Following prior work [5, 11], we implement them as:

$$\begin{split} \hat{c}_{\text{self_coll}}(\mathbf{q}) &= \sum_{(i,j)\in\text{links}} f(d_{ij}(\mathbf{q}), \eta_{ij}) \\ \hat{c}_{\text{world_coll}}(\mathbf{q}) &= \sum_{(i,j)\in\text{links,obs}} f(d_{ij}(\mathbf{q}), \eta_{ij}) \end{split}$$

where d_{ij} is the signed distance between collision geometries for link pairs (i, j) at joint configuration \mathbf{q} $(d_{ij} < 0$ if in collision). The signed distance is then converted into a cost d_c using a smooth activation function $f(d, \eta)$ = that avoids a discontinuity at d = 0 [5, 14] and penalizes

$$d_{c} = \begin{cases} -d + 0.5\eta & \text{if } d < 0\\ \frac{0.5}{\eta} (-d + \eta)^{2} & \text{if } 0 < d < \eta\\ 0 & \text{otherwise} \end{cases}$$
(1)

where η is the buffer distance. In Section IV-A, we show how collision can also be considered across timesteps.

D. Interactive Robot Web Viewer

Good visualization is critical for 3D robotics tasks. PyRoki includes an interactive web-based viewer built on viser [71], which provides an intuitive way to explore robot behavior in real time. With the viewer, users can interactively adjust cost function weights and immediately see how they affect robot motion, helping to balance competing objectives like end-effector accuracy, joint limits, and collision avoidance. Beyond weight tuning, the viewer allows interactive scene modifications (moving obstacles) and custom elements like manipulability ellipses (Figure 2).



Scene Collision Body Representation

Fig. 3: **Trajectory Optimization.** PyRoki can be used to formulate trajectory optimization problems that find valid collision-free solutions from naive straight-line initializations (top), similar to CHOMP [14]. The arm is approximated as spheres (bottom), which are connected into capsules for collision checking between neighboring timesteps.

IV. QUALITATIVE RESULTS

We illustrate PyRoki's modularity and extensibility using four robot kinematic tasks, which use both pre-defined (Sec. III-C) and custom costs. We begin with a case study on IK and trajectory optimization, drawing on existing optimization-based motion planning approaches [4, 14]. We then examine PyRoki for robot motion retargeting, transferring human motions to humanoids and robot hands.

A. Inverse Kinematics and Trajectory Optimization

Writing inverse kinematics in PyRoki is straightforward; the user only needs to import their robot URDF and assemble cost terms defined in Section III-C, such as the joint pose cost, joint limit costs, and joint regularization cost. This formulation can be easily extended to find IK solutions close to the current state, or to find collision-free solutions. Moreover, these tasks can be executed seamlessly across a diverse range of robotic platforms, as shown in Figure 1.

Much of the logic for IK can be re-used for trajectory optimization. We show an example setting where a UR5 robot arm moves between specified start and goal poses while avoiding an obstacle in the middle, as shown in Figure 3. The trajectory is optimized with the same costs in IK, but also including collision avoidance and acceleration- and jerkminimization. We solve for the initial trajectory by first determining collision-free start and goal joint configurations using inverse kinematics. We then linearly interpolate between these configurations to create a trajectory, which may initially be in collision with the environment.

We additionally implement continuous collision costs through sweeping volumes [4, 14] to check collision between timesteps. The UR5 robot is modeled as a series of spheres



Fig. 4: **Robot Motion Retargeting.** We show motion retargeting for humanoids and robot hands using PyRoki, using the *same* motion transfer cost across robots and tasks. To handle differences in robot morphology, we optimize for robot joint configurations and per-link scaling factors between embodiments simultaneously. Contact costs ensure humanoids stay grounded through scene contact (left) and maintain fingertip-object contact when present in the source motion (right). Blue dotted lines indicate contact relationships between the robot hand and the object.

(see Fig. 3, bottom), and each sphere associated with the UR5 at timestep t is connected to the corresponding sphere at the next timestep to form a capsule. These capsules are then used to compute collision costs with the surrounding world obstacles. We note that this cost is quick to implement—this can be done with only a few lines of Python. An example output is shown in Figure 3; see the supplemental video for more results.

B. Motion Retargeting

PyRoki's flexibility is especially useful for motion retargeting, which often requires scaling or motion warping. Extensibility lets PyRoki tackle retargeting scenarios that require careful balance between motion fidelity and physical constraints. We demonstrate the versatility of PyRoki through several challenging retargeting scenarios, from fullbody humanoid motion to hand-object interactions. Each task requires domain-specific costs, which can be prototyped quickly using auto-differentiated Jacobians.

We implement sparse, keypoint-based retargeting inspired by [51]. By aligning keypoint positions and preserving the relative distances and angular relationships between joints, our cost functions treat both body and hand cases identically. This unified approach allows our method to seamlessly adapt across diverse skeletal structures with varying scales and degrees of freedom. We discuss the details for each experiment below:

1) Full-body Humanoid Retargeting: We transfer human motion onto the Unitree G1 [73] and H1 [74] humanoids, ensuring that it remains *physically plausible within the scene*. Our inputs include human keypoint trajectories, the scene

mesh, and information regarding whether each foot is in contact with the ground on a per-frame basis. Motion transfer costs are designed such that they focus on using joint relationships to preserve the motion [51], instead of keypoint positions which tend to bring the humanoid feet too close together [7]. For each pair of joints in the kinematic chain, we optimize their relative positions (scaled by learned per-link factors) and relative angles (using cosine similarity between joint vectors) to match input human keypoints. This allows the optimizer to handle the differences in limb proportions. We also add a cost to penalize the knee joints from becoming too close to each other. To ensure physical plausibility, we enforce floor contact constraints, foot orientation costs, selfcollision avoidance, and joint limits. Example results are shown in Fig. 4. The same cost ensures robust retargeting to robots with significantly different shapes (G1: 127cm, H1: 178cm); see the supplemental video for more details.

2) Hand-Object Interaction Retargeting: In this scenario, we use PyRoki to transfer human hand motions from DexYCB [75] to a robot hand. We use the same motion transfer cost used in full-body humanoid retargeting to align MANO [76] human hand motions to a robotic Shadow Hand [77]. In parallel, we add a contact cost that maintains consistent contact between the robot hand and object surfaces. This modular architecture simplifies domain-specific objectives (e.g., contact), which improve results for this task.

V. QUANTITATIVE RESULTS

We evaluate PyRoki on modularity, flexibility, and computational efficiency. First, we show it can be used to implement the vector retargeting approach of Dex-Retargeting [6]



Fig. 5: **Implementing vector-based hand retargeting.** We implement the keypoint vector loss described in Dex-Retargeting [6] using PyRoki. Our global IK approach achieves slightly lower final costs than the original (diff. IK).

to achieve comparable hand motion quality. Second, we show how we can extend an IK implementation to simultaneously optimize the base pose of a Fetch mobile manipulator with its arm configuration. Finally, we compare IK runtime performance, measuring how autodiff and analytical Jacobians scale against cuRobo [5] on CPU and GPU. All experiments were performed on a desktop PC with an AMD 5955WX CPU and NVIDIA RTX 4090 GPU.

A. Reproducing Dex-Retargeting

To evaluate PyRoki's modularity, we demonstrate its ability to replicate existing methods by re-implementing the vector retargeting cost in Dex-Retargeting [6], a CPUbased differential IK optimization library. We find that our implementation can produce visually similar motions that capture the same semantic hand poses, and reach very similar values for the keypoint cost (0.068 \pm 0.070) for a human hand trajectory of 621 timesteps, as shown in Fig. 5. This problem can also be extended with new costs (e.g., contact loss in Sec.IV-B.2) and batch processing on GPUs.

B. Inverse Kinematics with a Mobile Base

In mobile manipulation, solving inverse kinematics (IK) requires considering both the robot's arm and base positions. A common method is to first estimate a fixed base pose and then solve IK for the arm [1, 17]. However, this sequential approach can fail if the chosen base pose makes the target end-effector pose unreachable.

With PyRoki's modular design, we can add the SE(2) base pose as an optimization variable and adjust pose cost such that the end effector's transform includes the base transform ($\mathbf{T}_{world\leftarrow base}\mathbf{T}_{base\leftarrow i}$). To evaluate this method, we sample reachable end-effector poses from a fixed base position and apply random translations to create target poses that may require base movement. An IK solution is deemed successful if the position error is under 5mm and the rotation error is below 0.05 radians.

Mahila Daga	Error (mea	Success		
Mobile Base	Pos. (m)	Rot. (rad)	Rate (%)	
Static Optimized	0.616 ± 0.600 5.40e-6 ± 4.31e-5	0.470 ± 0.426 0.0 ± 0.0	24 100	

TABLE II: **Optimizing base pose for IK with mobile robot**: The inverse kinematics task in Sec. IV-A can be modified to incorporate the base pose of the Fetch robot as an optimizable variable, which significantly improves both position and rotation accuracy.

Batch	PyRoki	(CPU)	PyRoki (GPU)		
	Analytical	AutoDiff	Analytical	AutoDiff	
1	5.9	36.9	3.6	11.4	
10	48.2	332.4	3.6	13.1	
100	396.7	2173.0	4.7	26.6	
1000	3037.6	19949.0	15.5	253.1	
2000	5534.7	39284.8	32.8	624.4	

TABLE III: **IK-Beam runtime for analytical vs. autodiff Jacobians.** We compare inverse kinematics solve times (ms) on CPU and GPU. Analytical Jacobians provide significant speedup, especially at higher batch sizes. IK-Beam uses 64 initial seeds; a batch size of 1000 means that up to 64000 LM steps are computed in parallel.

As shown in Table II, optimizing the base pose dramatically improves performance. With a fixed base, the solver achieves only a 15% success rate with large position errors (>60cm). In contrast, simultaneous optimization of the base pose achieves a 100% success rate with mean position errors under 0.005mm. This improvement highlights the practical utility of PyRoki's extensibility.

C. Runtime Analysis

PyRoki supports both analytical and autodiff Jacobians, parallelized optimization, and different computation platforms (CPU, GPU, TPU). To explore the advantages of this flexibility, we explore IK for the Franka Panda robot using a beam search-inspired algorithm that we call *IK-Beam*. IK-Beam is designed to be both robust and simple to implement using PyRoki. Given a target pose, IK-Beam begins with 64 start seeds. We run 16 total Levenberg-Marquardt steps. The first 6 LM steps optimize all 64 initializations in parallel. We then discard all but the 4 seeds with lowest error, which are optimized for the 10 final LM steps. The returned solution is the best from these 4 seeds.

Table III presents IK-Beam solve times across varying batch sizes, Jacobian options, and CPU/GPU compute platforms. We find that analytical Jacobians provide speedups ranging from 3x-19x over autodiff. GPU parallelization helps runtime scale better with increasing batch sizes.

We additionally compare with cuRobo [5], a popular GPUaccelerated IK library (Table I). Depending on batch size, we find that IK-Beam is between 1.4x and 1.7x faster, while converging to lower errors. We believe that the accuracy improvement is mostly explained by the optimizer—cuRobo uses L-BFGS, while PyRoki uses LM (Sec. III-A).

Batch	cuRobo [5]				IK-Beam (PyRoki)				
	Time (ms)	Succ. (%)	Pos. Err	Ori. Err (rad)	Time (ms)	Succ. (%)	Pos. Err (mm)	Ori. Err (rad)	
1	5.03	100	7.40×10^{-4}	2.55×10^{-6}	3.58	100	8.50×10^{-5}	5.65×10^{-7}	
10	5.27	100	2.67×10^{-3}	5.58×10^{-6}	3.60	100	1.65×10^{-4}	2.72×10^{-7}	
100	6.76	100	3.59×10^{-3}	7.05×10^{-6}	4.70	100	2.99×10^{-4}	4.19×10^{-7}	
1000	26.37	100	4.70×10^{-3}	6.96×10^{-6}	15.54	100	2.55×10^{-4}	3.89×10^{-7}	
2000	50.32	99.95	4.31×10^{-3}	6.93×10^{-6}	32.81	100	3.04×10^{-4}	4.67×10^{-7}	

TABLE IV: Comparison between cuRobo and IK-Beam. IK-Beam, which is implemented using PyRoki, is faster and converges to lower errors. Experiments are run on the Franka Panda robot with a modified version of the official cuRobo [5] benchmarking script. Following cuRobo, reported errors are the 98th percentile of all solutions in the batch.

VI. CONCLUSION

We present PyRoki, a modular, extensible, and crossplatform toolkit for robot kinematic optimization. A variety of robot kinematic optimization problems can be implemented and extended simply by composing variables and cost functions, while running efficiently on CPUs, GPUs, and TPUs. For example, we can reuse the same collision costs throughout all the tasks presented in the paper—from inverse kinematics, trajectory optimization, to motion retargeting with humanoids. PyRoki is open-source and invites users to extend its capabilities.

VII. ACKNOWLEDGEMENT

This project was funded in part by NSF:CNS-2235013, IARPA DOI/IBC No. 140D0423C0035, and DARPA No. HR001123C0021; Chung Min Kim and Brent Yi are supported by the NSF Research Fellowship Program, Grant DGE 2146752. We thank Kevin Zakka and Justin Kerr for fruitful technical conversations.

REFERENCES

- P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), IEEE, 2015, pp. 928–935.
- [2] S. Caron, Y. De Mont-Marin, R. Budhiraja, S. H. Bang, I. Domrachev, and S. Nedelchev, *Pink: Python inverse kinematics based* on *Pinocchio*, version 3.2.0, 2025.
- [3] K. Zakka, Mink: Python inverse kinematics based on MuJoCo, version 0.0.4, Jul. 2024.
- [4] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal* of Robotics Research, vol. 33, no. 9, pp. 1251–1270, 2014.
- [5] B. Sundaralingam *et al.*, "Curobo: Parallelized collision-free robot motion generation," in 2023 IEEE International Conference on Robotics and Automation (ICRA), 2023, pp. 8112–8119.
- [6] Y. Qin et al., "Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system," in *Robotics: Science and Systems*, 2023.
- [7] T. He et al., "Learning human-to-humanoid real-time whole-body teleoperation," in 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2024, pp. 8944–8951.
- [8] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics," in 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 3191–3198.
- [9] D. Rakita, B. Mutlu, and M. Gleicher, "Relaxedik: Real-time synthesis of accurate and feasible robot arm motion.," in *Robotics: Science* and Systems, Pittsburgh, PA, vol. 14, 2018, pp. 26–30.
- [10] A. Handa et al., "Dexpilot: Vision-based teleoperation of dexterous robotic hand-arm system," in 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 9164–9170.

- [11] Y. Wang, P. Praveena, D. Rakita, and M. Gleicher, "Rangedik: An optimization-based robot motion generation method for ranged-goal tasks," in 2023 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2023, pp. 9700–9706.
- [12] S. Zhong, T. Power, A. Gupta, and P. Mitrano, *PyTorch Kinematics*, version v0.7.1, Feb. 2024.
- [13] F. Meier, A. Wang, G. Sutanto, Y. Lin, and P. Shah, "Differentiable and learnable robot models," *arXiv preprint arXiv:2202.11217*, 2022.
- [14] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in 2009 IEEE international conference on robotics and automation, IEEE, 2009, pp. 489–494.
- [15] M. Bhardwaj et al., "Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *Conference on Robot Learning*, PMLR, 2022, pp. 750–759.
- [16] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in 2011 IEEE international conference on robotics and automation, IEEE, 2011, pp. 4569–4574.
- [17] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE robotics & automation magazine*, vol. 19, no. 1, pp. 18–19, 2012.
- [18] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.
- [19] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," *Robotics: Science and Systems VIII*, pp. 1–8, 2012.
- [20] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," Advances in neural information processing systems, vol. 32, 2019.
- [21] M. Quigley et al., "Ros: An open-source robot operating system," in ICRA workshop on open source software, vol. 3, 2009, p. 5.
- [22] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, Aug. 2010.
- [23] B. E. Paden, "Kinematics and control of robot manipulators," *Ph. D. Thesis*, 1985.
- [24] D. L. Pieper, The kinematics of manipulators under computer control. Stanford University, 1969.
- [25] R. P. Paul, Robot manipulators: mathematics, programming, and control: the computer control of robot manipulators. Richard Paul, 1981.
- [26] M. Raghavan and B. Roth, "Kinematic analysis of the 6r manipulator of general geometry," in *International symposium on robotics research*, Citeseer, 1990, pp. 314–320.
- [27] M. L. Husty, "An algorithm for solving the direct kinematics of general stewart-gough platforms," *Mechanism and Machine Theory*, vol. 31, no. 4, pp. 365–379, 1996.
- [28] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *IEEE International Conference on Robotics and Automation*, 2003, pp. 2766–2771.
- [29] D. Rakita, H. Shi, B. Mutlu, and M. Gleicher, "Collisionik: A per-instant pose optimization method for generating robot motions with environment collision avoidance," in 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 9995–10001.
- [30] D. Rakita, B. Mutlu, and M. Gleicher, "RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion," in *Proceedings*

of Robotics: Science and Systems, Pittsburgh, Pennsylvania, Jun. 2018.

- [31] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Transactions on man-machine systems*, vol. 10, no. 2, pp. 47–53, 1969.
- [32] D. E. Whitney, "The mathematics of coordinated control of prosthetic arms and manipulators," 1972.
- [33] A. Liegeois *et al.*, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE transactions on* systems, man, and cybernetics, vol. 7, no. 12, pp. 868–871, 1977.
- [34] M. Gleicher, "Retargetting motion to new characters," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998, pp. 33–42.
- [35] M. Mukadam, X. Yan, and B. Boots, "Gaussian process motion planning," in 2016 IEEE international conference on robotics and automation (ICRA), IEEE, 2016, pp. 9–15.
- [36] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.
- [37] A. D. Dragan, N. D. Ratliff, and S. S. Srinivasa, "Manipulation planning with goal sets using constrained trajectory optimization," in 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 4582–4588.
- [38] L. Wang, Y. Xiang, and D. Fox, "Manipulation trajectory optimization with online grasp synthesis and selection," in *Robotics: Science* and Systems (RSS), 2020.
- [39] J. Ichnowski, M. Danielczuk, J. Xu, V. Satish, and K. Goldberg, "Gomp: Grasp-optimized motion planning for bin picking," in 2020 IEEE international conference on robotics and automation (ICRA), IEEE, 2020, pp. 5270–5277.
- [40] F. Dellaert, M. Kaess, et al., "Factor graphs for robot perception," Foundations and Trends[®] in Robotics, vol. 6, no. 1-2, pp. 1–139, 2017.
- [41] B. Yi, M. A. Lee, A. Kloss, R. Martín-Martín, and J. Bohg, "Differentiable factor graph optimization for learning smoothers," in 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2021, pp. 1339–1345.
- [42] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
- [43] S. Agarwal, K. Mierle, and T. C. S. Team, *Ceres Solver*, version 2.2, Oct. 2023.
- [44] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G 2 o: A general framework for graph optimization," in 2011 IEEE international conference on robotics and automation, IEEE, 2011, pp. 3607–3613.
- [45] E. S. Ho, T. Komura, and C.-L. Tai, "Spatial relationship preserving character motion adaptation," in ACM SIGGRAPH 2010 papers, 2010, pp. 1–8.
- [46] T. Schmidt, R. A. Newcombe, and D. Fox, "Dart: Dense articulated real-time tracking.," in *Robotics: Science and systems*, Berkeley, CA, vol. 2, 2014, pp. 1–9.
- [47] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, and S. Levine, "Sfv: Reinforcement learning of physical skills from videos," ACM Transactions On Graphics (TOG), vol. 37, no. 6, pp. 1–14, 2018.
- [48] Z. Fu, Q. Zhao, Q. Wu, G. Wetzstein, and C. Finn, "Humanplus: Humanoid shadowing and imitation from humans," in *Conference* on Robot Learning (CoRL), 2024.
- [49] H. Zhang *et al.*, "Kinematic motion retargeting via neural latent optimization for learning sign language," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4582–4589, 2022.
- [50] J. Zhang et al., "Skinned motion retargeting with residual perception of motion semantics & geometry," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 13 864–13 872.
- [51] T. Cheynel, T. Rossi, B. Bellot-Gurlet, D. Rohmer, and M.-P. Cani, "Sparse motion semantics for contact-aware retargeting," in ACM SIGGRAPH Conference on Motion, Interaction and Games (MIG), 2023.
- [52] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black, "AMASS: Archive of motion capture as surface shapes," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

- [53] Y. Li, T. Nagarajan, B. Xiong, and K. Grauman, "Ego-exo: Transferring visual representations from third-person to first-person videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision* and Pattern Recognition (CVPR), 2021.
- [54] J. Lin et al., "Motion-x: A large-scale 3d human motion dataset with x-reenactment," in Advances in Neural Information Processing Systems, 2023.
- [55] M. Bollo et al., "Nymeria: A multimodal indoor motion capture dataset with real and synthetic egocentric video," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024.
- [56] M. Abadi et al., "{Tensorflow}: A system for {large-scale} machine learning," in 12th USENIX symposium on operating systems design and implementation (OSDI 16), 2016, pp. 265–283.
- [57] M. Lubin, O. Dowson, J. Dias Garcia, J. Huchette, B. Legat, and J. P. Vielma, "JuMP 1.0: Recent improvements to a modeling language for mathematical optimization," *Mathematical Programming Computation*, 2023.
- [58] R. Tedrake and the Drake Development Team, Drake: Model-based design and verification for robotics, 2019.
- [59] F. Dellaert and G. Contributors, *Borglab/gtsam*, version 4.2a8, May 2022.
- [60] J. Dong and Z. Lv, "Minisam: A flexible factor graph non-linear least squares optimization framework," arXiv preprint arXiv:1909.00903, 2019.
- [61] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, pp. 1–36, 2019.
- [62] B. Houska, H. J. Ferreau, and M. Diehl, "Acado toolkit—an opensource framework for automatic control and dynamic optimization," *Optimal control applications and methods*, vol. 32, no. 3, pp. 298– 312, 2011.
- [63] J. Bradbury et al., JAX: Composable transformations of Python+NumPy programs, version 0.3.13, 2018.
- [64] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.
- [65] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.
- [66] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 93–101, 1986.
- [67] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," 1986.
- [68] S. R. Buss and J.-S. Kim, "Selectively damped least squares for inverse kinematics," *Journal of Graphics tools*, vol. 10, no. 3, pp. 37–49, 2005.
- [69] B. Yi *et al.*, "Estimating body and hand motion in an ego-sensed world," *arXiv preprint arXiv:2410.03665*, 2024.
- [70] N. Heppert, T. Migimatsu, B. Yi, C. Chen, and J. Bohg, "Categoryindependent articulated object tracking with factor graphs," in 2022 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 3800–3807.
- [71] B. Yi, Viser, https://viser.studio/main/, Accessed: 2025-02-17, 2025.
- [72] T. Yoshikawa, "Manipulability of robotic mechanisms," *The international journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.
- [73] Unitree g1 humanoid robot, Unitree Robotics, Available: https://www.unitree.com/g1, 2024.
- [74] Unitree h1 humanoid robot, Unitree Robotics, Available: https://www.unitree.com/h1, 2024.
- [75] Y.-W. Chao et al., "DexYCB: A benchmark for capturing hand grasping of objects," in *IEEE/CVF Conference on Computer Vision* and Pattern Recognition (CVPR), 2021.
- [76] J. Romero, D. Tzionas, and M. J. Black, "Embodied hand shape and pose estimation from a single depth image," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 4907–4916.
- [77] *The shadow dexterous hand*, Shadow Robot Company, Accessed: [Insert Access Date], Available: https://www.shadowrobot.com/dexterous-hand-series/.