Exploring the Performance Implications of Collision Geometry Representations in Robot Motion Control

Fallon McBrien
fmcbrien@bu.edu
Boston University
Massachusetts, USA

Radhika Ghosal
rghosal@g.harvard.edu
Harvard University
Massachusetts, USA

Sabrina M. Neuman
sneuman@bu.edu
Boston University
Massachusetts, USA

Abstract

Robotics and other Physical AI applications rely on fast nonlinear optimization solvers to respond to changes in the physical environment in real time. However, state-of-the-art solvers do not currently meet the desired latency deadlines for responsive robot behavior. Thus, it is crucial to understand the relationship between robotics-specific application characteristics and their resulting compute behavior. To this end, this paper presents a characterization of the robot self-collision-free inverse kinematics problem using a state-of-the-art parallelized GPU based solver and a conventional serial CPU based solver. We study a range of sphere representations of the target robot, and quantify the three-way trade-off between representation fidelity, batch size, and solution accuracy. We find that GPU implementations provide substantially better scaling with increasing batch size and robot model fidelity, while CPU implementations can remain competitive for smaller batch sizes and produce more accurate solutions.

1 Introduction

Robot motion planning and control in high-dimensional environments is a fundamental challenge for real-time applications due to long run-times and computational complexity associated with collision checking between mesh-based models [5, 11]. The expectations for safe robot operation under strict real-time constraints continue to increase, with best practice control rates for articulated robots being on the order of 100s of Hz [6, 9]. This creates the need for more efficient motion planning and control methods.

One promising approach is the use of simplified geometric primitives (e.g., spheres, capsules, boxes, etc.) to approximate robot geometries. This reduces the cost of collision checking, the predominant bottleneck in motion planning, while preserving accuracy, given a sufficiently high fidelity approximation.

Recent work in sampling-based motion planning (SBMP) has demonstrated that sphere-based geometric primitive representation can provide order-of-magnitude improvements to performance by only leveraging vectorized instructions commonly available on commodity hardware[11]. These improvements are especially valuable for tasks involving high degree-of-freedom (DOF) robots, such as manipulator arms.

While these methods are explored in SBMP, the underlying benefits of geometric primitives can be expanded to broader cases of nonlinear optimization problems. In particular, optimization-based methods such as inverse kinematics, trajectory optimization, and model predictive control rely heavily on repeated calculations of collision constraints and distance gradients [10], which become increasingly expensive when applied to high DOF manipulators or humanoids. As such, efficient primitive-based representations of robotic systems may offer similar performance gains to those seen in SBMP.

Robotics applications require physically valid and collision-free solutions to ensure safety and hardware feasibility. A serial manipulator arm over-extending in a computer graphics-based application may seem visually feasible, but for a robotics application, may be physically unacceptable.

Thus, it is crucial to understand the relationship between robotics-specific application characteristics, such as geometric representations, and the resulting compute behavior, to safely accelerate motion planning. To this end, this paper profiles the use of sphere models with varying fidelities across a state-of-the-art hardware-accelerated nonlinear optimization solver, cuRobo [10], and a custom CPU-based solver to perform the self-collision-free inverse kinematics problem.

We analyze the tradeoffs between latency, convergence success, and geometric accuracy of model fidelities across both serial CPU and parallelized GPU applications, showing how spherized models can result in up to 150× speedup over the original mesh geometry, with tradeoffs in success rate and accuracy.

2 Background

2.1 Inverse Kinematics

Inverse kinematics (IK) is a fundamental problem in robotics that determines the joint configuration of a robot needed to achieve a desired end-effector pose while satisfying kinematic and environmental constraints [2, 4].

While IK has its origins in robotic manipulation, it has also been used in computer graphics for animating articulated subjects and humanoid characters [1]. However, the accuracy demands of robotics and graphics applications can differ. In graphics applications, occasional self- or environmental-collision is often tolerable without changing visuals, whereas

in physical robotics applications, self-collisions can create unsafe behaviors.

Optimization-based solvers commonly express self-collision-free IK as a constrained nonlinear optimization problem. These methods iteratively minimize pose error while enforcing collision avoidance constraints between robot links and the environment. As robot complexity increases, the computational cost of solving these optimization problems grows rapidly due to the number of collision constraints increasing with the square of the number of geometries, and the increasingly complex Jacobian computations required.

To solve these constrained nonlinear optimization problems, modern IK solvers employ different optimization algorithms such as Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) or Augmented Lagrangian (AugLag) optimization [7]. L-BFGS is well-suited for massively parallel GPU execution, such as cuRobo [10], because it relies on dense vectorized linear algebra operations to approximate a Hessian that can be batched across many IK problems simultaneously. In contrast, AugLag involves sequential constraint updates through iterative penalty adjustments, making it more suitable for serial CPU execution as seen in the custom CPU solver used in this paper.

2.2 Geometric Approximations

For geometric sphere primitive representation, the number of potential self-collision pairs scales with the number of spheres used to model the robot. Higher-fidelity models produce more accurate approximations of robot geometry but significantly increase the number of collision constraints. Since each collision constraint contributes to the additional rows of the constraint Jacobian, both memory usage and compute cost increase with model fidelity.

Despite this scaling, sphere-based collision representations are still more computationally efficient than mesh-based representations [3]. Distance calculations between two spheres have closed-form analytical solutions with inexpensive gradients, while mesh-based collision checking often requires iterative closest-point calculations, bounding volume traversal, or signed distance field queries.

3 Experimental Methodology

Experimental Setup. All experiments were run on a workstation equipped with an NVIDIA RTX 4070 GPU and an Intel Core i7-14700K CPU. We use the Kinova Gen3 7DoF manipulator arm [8] and consistent randomly-generated target end-effector pose inputs across both evaluated solvers. The success rate and accuracy measurements were taken as the averages of 20 iterations of 100 batch size (or 2000 attempts).

Geometry Approximations. To profile the impact of geometric representation fidelity on nonlinear optimization performance, we evaluate our solvers over a number of sphere approximations of the Kinova manipulator arm, hereon referred to as spherized models. We use the Foam [3] tool to generate spherized models of varying fidelities.

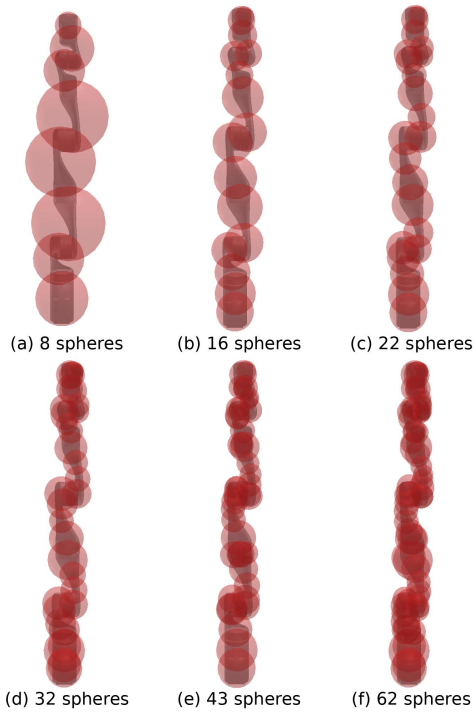

Figure 1. Spherized Foam Models of the Kinova Gen3 7DoF Arm. Our experiments sweep the visualized fidelities.

Foam takes as input mesh-based universal robot description format (URDF) files and outputs spherized URDF files. It converts complex mesh geometries into sphere representations through a multi-stage preprocessing pipeline for mesh smoothing, then an adaptive medial axis approximation to place spheres. The resulting sphere models preserve the overall morphology of the robot, possibly over-approximating geometry so there are no situations where the real mesh geometry of the robot is in collision, but the approximation is not. We generate spherized models containing 8, 16, 22, 32, 43, and 62 spheres (Fig. 1).

Solvers Evaluated. We analyze the performance of two solvers: **cuRobo**, a state-of-the-art GPU implementation of the L-BFGS method [7] for robotics [10], and a standard unoptimized CPU implementation in C++ of an Augmented Lagrangian-based method [7], hereon referred to as the **CPU solver**. Solution tolerances are set to 0.1mm for both.

cuRobo is specialized to solve the batched self-collision-free IK problem using sphere-based collision representations. The CPU solver solves for generic cost and constraint functions. For transcribing the self-collision-free IK problem for the CPU solver, we add a target pose error cost, and self-collision and joint limit constraints for the Kinova arm.

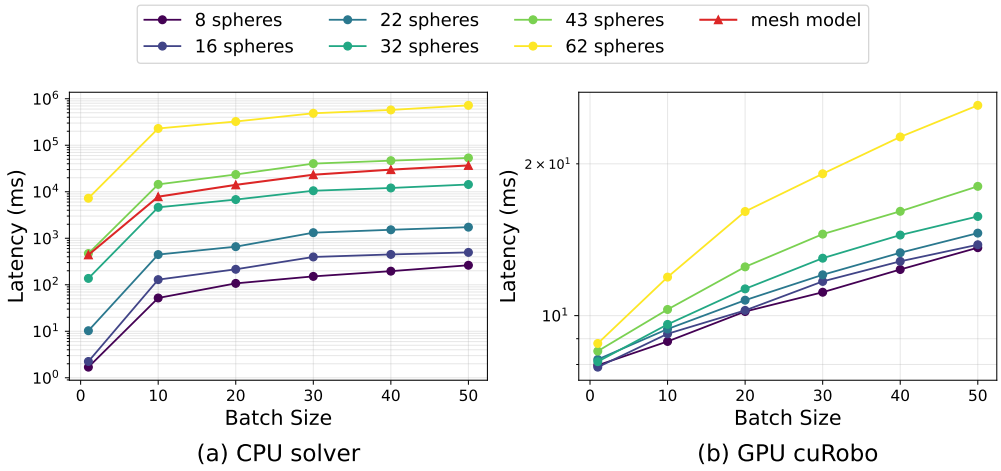

Figure 2. Latency vs. Batch Size for the CPU Solver and cuRobo. (a) After 32 spheres, solve time for the sphere models on the CPU becomes slower than the mesh model, (b) GPU speedup over CPU ranges from 5.8× faster for 8 spheres at batch size 10 to 30,100× for 62 spheres at batch size 50.

Experiment Knobs. Our experiments sweep sphere fidelities (ranging from 8–62 as mentioned above) and batch sizes (1–300). For each experiment, we execute 20 trials to account for variance in solver convergence and run-time.

Evaluation Metrics. We evaluate each solver and experiment knob combination using the following metrics: (i) *Latency*: the total time taken by each experiment, (ii) *Success Rate*: the percentage of optimization problems within each experiment that converged to a self-collision-free robot configuration within the solver tolerances, and, (iii) *Accuracy*: the Euclidean position error between the target end-effector pose input and the final achieved end-effector pose.

For the GPU solver experiments, cuRobo dispatches a single batch of `batch_size` solves simultaneously. The CPU solver performs the same number of solves serially to allow for direct comparison between parallel and serial solver methods on the GPU and CPU.

4 Evaluation

To understand the trade-offs between robot model fidelity and application metrics (latency, success rate, and accuracy) for self-collision-free IK on the CPU and GPU, we sweep batch sizes and spherized model fidelities of the Kinova Gen3 arm.

4.1 Impact of Model Fidelity on Application Metrics

Model fidelity, or the number of spheres representing the model, quadratically increases the number of collision pairs in self-collision-free IK, and thus increases solver run-time.

Impact on CPU Solve Latency. For low fidelities, e.g., the 8-sphere model, the spherized model heavily outperforms the mesh model in latency due to the simplicity of sphere-sphere collision checks. Spherized models can have up to 150× faster latency than the mesh model (Fig. 2). However, the additional collision checks induced by higher-fidelity models become more expensive than performing the mesh-mesh collision check. This results in a crossover point where solves using the higher-fidelity sphere models take longer than the mesh model, in this case, after the 32-sphere model.

Impact on Solver Success Rates.

Sphere models inherently approximate the robot geometry and overestimate occupied space to be conservative in collision checking. This improves safety, but can prevent valid configurations from converging.

However, as fidelity increases, the sphere models begin to match the behavior of the mesh model (Fig. 3). The 22-sphere model is the point of diminishing returns, after which success rates plateau while solve latencies continue to increase (Fig. 2 & Fig. 3). For the Kinova arm, the 22- and 32-sphere models provide the best balance between latency and convergence, having a success rate only slightly lower than that of the mesh model (94% vs. 99%) while benefiting from solve times up to 19× faster.

Impact on Solve Quality. Position errors between the target and the achieved end-effector poses for successful solves are comparable across all sphere fidelities (Fig. 4), with the average ground truth position error for the mesh model being 0.4μm.

These results suggest that intermediate-fidelity sphere models can maintain mesh-comparable accuracy while reducing solve times, by paying a small price in success rate.

4.2 GPU Comparison

We now evaluate the behavior of these spherized models within the context of massively-parallel GPU-based optimization pipelines that can take advantage of simplified collision

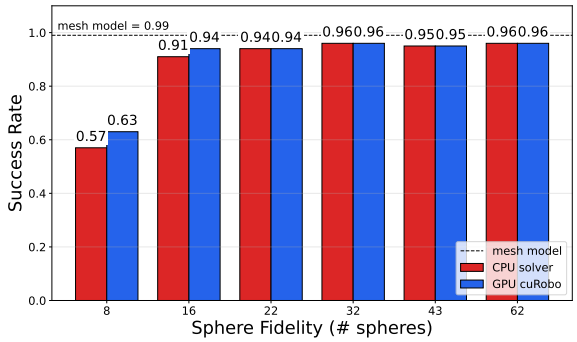

Figure 3. Success Rate vs. Fidelity for the CPU Solver and cuRobo. After 22 spheres, success rates plateau. Measured for batch size 100.

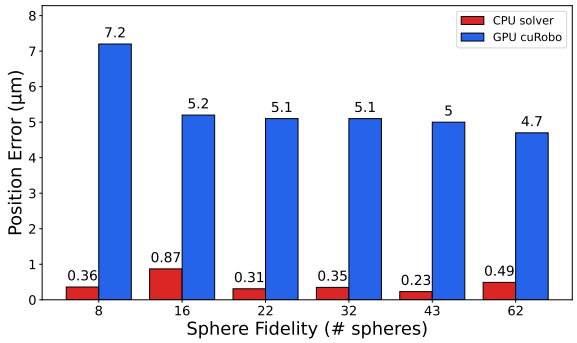

Figure 4. Average Position Error Between Target Pose and Achieved Pose. The CPU solver produces solutions up to 22× more accurate than the GPU. Measured for batch size 100.

checks. The state of the art GPU solver, cuRobo, operates only on spherized models.

Model Fidelity vs. Latency. Across all fidelities and batch sizes tested, cuRobo consistently achieves lower latency than the CPU solver (Fig. 2). GPU latency has superior scaling over the CPU as fidelity and batch sizes increase.

The weakened dependence of cuRobo latency on sphere count can be attributed to the massively parallel structure of the batched L-BFGS solve. Collision checks and constraint jacobian computations are distributed across thousands of GPU threads. In contrast, the CPU solver operates serially, causing latency to grow rapidly with sphere fidelity and batch size.

Model Fidelity vs. Success Rates and Accuracy. Success rates and accuracy seen across the GPU experiments are worse than the corresponding CPU ones, as seen in Figs. 3 and 4. These results suggest that while sphere approximations are especially well suited for GPU solvers latency-wise, applications demanding more precision may benefit from CPU implementations.

5 Future Work

This work scratches the surface of the tradeoffs of replacing mesh geometry with geometric primitives for nonlinear optimization. Future work will investigate alternative primitive representations, such as capsules and boxes, which may provide improved model representation while requiring fewer collision constraints than high-fidelity sphere models.

Another extension of this work is evaluating these experiments in more realistic environments containing world collision models for obstacles and constrained workspaces, where geometric over-approximation may have a more significant impact on latency, success, and accuracy.

Finally, future work will explore how sphere generation methods affect optimization performance. While increasing sphere count generally increases computational cost while improving result quality, results seen in Fig. 3, where success rate of the 43-sphere model is slightly lower than the 32-sphere model, suggest that sphere placement can also influence convergence. Understanding the relationship between primitive generation and fidelity will allow for more representative models using fewer primitives.

6 Conclusion

This paper evaluates the impact of sphere-model fidelity on self-collision-free inverse kinematics across CPU and GPU optimization solvers. The results demonstrate a clear tradeoff between increasing sphere fidelity and computational cost. Increasing fidelity generally improves convergence behavior, but also increases the latency due to the larger number of collision constraints associated with adding spheres. Our sweep studies reveal medium-fidelity sphere models that enable solver success rates comparable to mesh representations, while being up to 19× faster than meshes on CPUs. The GPU-based cuRobo solver achieves orders-of-magnitude lower latency than the CPU one for almost all problem sizes, while the CPU solver produces more accurate solutions.

These results suggest that geometric primitive representations are an effective approach for accelerating optimization-based robotics applications, especially as motion planning and control systems continue to push toward high-dimensional robots and faster re-planning rates. Ultimately, understanding how geometric fidelity impacts optimization performance is important not only for improving runtime, but also for balancing realism, safety, and computational cost across robotic applications.

References

[1] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir. 2018. Inverse Kinematics Techniques in Computer Graphics: A Survey. *Computer Graphics Forum* 37, 6 (2018), 35–58. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13310 doi:10.1111/cgf.13310

[2] Thomas Cohn, Seiji Shaw, Max Simchowitz, and Russ Tedrake. 2024. Constrained bimanual planning with analytic inverse kinematics. In

2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 6935–6942.

[3] Sai Coumar, Gilbert Chang, Nihar Kodkani, and Zachary Kingston. 2025. Foam: A tool for spherical approximation of robot geometry. *arXiv preprint arXiv:2503.13704* (2025).

[4] Serdar KuCuk and Zafer Bingul. 2004. The inverse kinematics solutions of industrial robot manipulators. In *Proceedings of the IEEE International Conference on Mechatronics, 2004. ICM'04.* IEEE, 274–279.

[5] Mustafa Mukadam, Xinyan Yan, and Byron Boots. 2016. Gaussian process motion planning. In *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 9–15.

[6] Sabrina M Neuman, Brian Plancher, Thomas Bourgeat, Thierry Tambe, Srinivas Devadas, and Vijay Janapa Reddi. 2021. Robomorphic computing: a design methodology for domain-specific accelerators parameterized by robot morphology. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 674–686.

[7] Jorge Nocedal and Stephen J Wright. 2006. *Numerical optimization*. Springer.

[8] Kinova Robotics. [n. d.]. Discover our Gen3 robotic arm — kinovarobotics.com. https://www.kinovarobotics.com/product/gen3-robots. [Accessed 29-05-2026].

[9] Jean-Pierre Sleiman, Farbod Farshidian, Maria Vittoria Minniti, and Marco Hutter. 2021. A unified mpc framework for whole-body dynamic locomotion and manipulation. *IEEE Robotics and Automation Letters* 6, 3 (2021), 4688–4695.

[10] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, et al. 2023. Curobo: Parallelized collision-free robot motion generation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 8112–8119.

[11] Wil Thomason, Zachary Kingston, and Lydia E Kavraki. 2024. Motions in microseconds via vectorized sampling-based planning. In *2024 IEEE international conference on robotics and automation (ICRA)*. IEEE, 8749–8756.