GPU-Accelerated Collision-Free Path Planning for Multi-Axis Robots in Construction Automation

Ilija Vukorep[1] Rolf Starke[1] Arastoo Khajehee[2] Nicolas Rogeau[2] Yasushi Ikeda[2]

[1]Faculty of Architecture, Civil Engineering and Urban Planning, BTU Cottbus-Senftenberg, Germany
[2]Department of Architecture, Graduate School of Engineering, The University of Tokyo, Japan
ilija.vukorep@b-tu.de, rolf.starke@b-tu.de, arastoo@arch1.t.u-tokyo.ac.jp, nicolas@arch1.t.u-tokyo.ac.jp, yasushi@arch1.t.u-tokyo.ac.jp

Abstract –

The Architecture, Engineering, and Construction (AEC) sector faces increasing pressure for higher production rates amidst a growing shortage of skilled labor, driving the demand for advanced robotic applications to enhance precision, efficiency, and adaptability in complex environments.

This paper introduces a software setup designed to ensure collision-free movements for multi-axis robots in AEC scenarios. Our approach leverages the NVIDIA cuRobo framework's robust capabilities, seamlessly integrated with Grasshopper for Rhino 3D software (GH), a tool widely recognized for its versatility in parametric design.

The integration of these technologies allows for the efficient online generation of optimal path movements, avoiding collisions even in highly intricate settings and changing environments. This is achieved in a remarkably short timeframe, enhancing productivity and reducing downtime. NVIDIAs framework's GPU-driven architecture paired with our GH parametric and controlling setup is a significant advancement, validated through a case study involving a complex, tree-like structure constructed from timber sticks. Using a six-axis robotic arm, the study demonstrates the system's capability to navigate and manipulate within congested spaces efficiently. With this enhanced automation workflow, new possibilities emerge for robotic applications, from industrial automation to sophisticated construction projects. Our GH software also allows visualization and exchange with URDF-models and better planning of collision logic, which was previously only possible with ROS and Nvidia Isaac technology.

Keywords –
collision-free path planning, multi-axis robots, AI robotic automation, parametric design

1 Introduction

Collision-free path planning is one of the most complex challenges in robotics, particularly in the AEC industry, where multi-axis robots must navigate dynamic, congested, and constrained environments [1]. We define collision-free in this study as the ability of the system to generate paths that avoid both self-collisions and external obstacles while optimizing for smooth motion and execution speed. Traditional path planning methods, such as manual programming or basic algorithms, are time-consuming and prone to errors, making them unsuitable for the intricate geometries and real-time adaptability required in construction scenarios. While advancements in sampling-based and grid-based algorithms have improved pathfinding capabilities, their integration into widely used design tools remains limited. Most existing solutions in platforms like Grasshopper for Rhino3D (GH) do not support real-time collision avoidance, leaving a critical gap in robotic automation workflows.

This paper addresses this gap between computational robotic planning and architectural workflows, making robotic automation more accessible to AEC professionals by introducing a novel framework that integrates NVIDIA cuRobo [2], a GPU-accelerated motion planning library, with GH. By leveraging cuRobo's advanced capabilities, the system enables real-time, collision-free path planning, even in dynamic environments. This streamlined approach eliminates many of the barriers associated with traditional methods, offering a robust solution for automating complex robotic tasks. The framework's effectiveness is validated through a case study of a robotic timber structure assembly, demonstrating its ability to navigate full cluttered environments with precision and reliability. This advancement represents a significant step toward solving one of the core challenges of robotic automation in the AEC industry.

2 Background & Related Work

The integration of multi-axis robots into the AEC industry has significantly advanced automated fabrication and assembly processes. A critical component of this integration is the development of efficient, collision-free path planning algorithms that can be seamlessly incorporated into architectural design software environments.

Path planning for multi-axis robots involves computing trajectories that avoid obstacles, adhere to the robot's kinematic constraints, and optimize performance metrics like speed and precision. Traditional methods often relied on manual programming, which is time-consuming and susceptible to errors. Recent advancements have focused on automating this process to enhance efficiency and adaptability in complex architectural tasks. One of the most common algorithms comes from the group of **Sampling-Based Algorithms**, such as Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM), that have been widely adopted for robotic path planning in software solutions [3]. These algorithms explore the robot's configuration space by randomly sampling points and connecting them to construct feasible paths. For instance, RRT incrementally builds a tree by randomly sampling the configuration space, effectively handling high-dimensional spaces typical of multi-axis robots. However, their direct application within design environments has been limited, necessitating the development of more accessible tools for architects and designers. In contrary to this, the **Grid-Based Algorithms**, such as the A* algorithm, operate on a discretized representation of the environment [4,5]. They evaluate possible paths based on cost functions to determine the most efficient route. In construction scenarios, an improved A* algorithm with a dynamic weight heuristic has been developed for global path planning, enhancing the robot's ability to navigate complex sites. Our paper describes the implementation of NVIDIA-cuRobo Algorithm that falls into the category of **Optimization-Based Algorithms.** In recent years, optimization-based algorithms have gained prominence in robotic motion planning, particularly for complex tasks requiring smooth and efficient trajectories. These algorithms frame motion planning as an optimization problem, aiming to find the best path according to specific criteria, such as minimizing time, energy, or/and avoiding obstacles.

To bridge the gap between advanced path planning algorithms and architectural design, several plugins have been developed to enable robot programming within design software. For instance, the Software RoboDK [6], which incorporates the PRM motion planner, integrates path planning software interfaces into computational design environments to automatically generate collision-free paths within the robot's workspace. Another example is MoveIt [7], which integrates seamlessly with various computational platforms to facilitate motion planning, collision checking, and trajectory generation. Unlike RoboDK, which focuses on predefined paths within industrial environments, MoveIt is often used for more dynamic and adaptive scenarios, making it suitable for robotics applications that require real-time decision-making or complex manipulations.

Those two software solutions are commonly used in industries such as automotive manufacturing and aerospace, where fabrication setups often rely on highly predefined and controlled processes. Althought RoboDK has a GH plugin and MoveIt can be integrated within Compas-Fab [8| framework the collision-free path planning is not trivial and robust. ROS and its simulation platforms, Gazebo and PyBullet, are excellent for developing robotic systems similar to those presented in this paper. Their integration with NVIDIA Omniverse also makes them potential candidates for automatic collision-free path planning. However, these systems require specialized robotics and programming expertise, posing a barrier to adoption in the AEC industry. While no definitive study exists on the most widely used software in AEC robotic fabrication, the prevalence of Rhino and Grasshopper (GH) plugins across major robotic and software companies suggests its dominant role. GH's visual programming capabilities lower the technical barrier, making it a crucial bridge between design and fabrication, particularly in parametric workflows with evolving design constraints.

For GH, several plugins are available for robotic control and path planning, including HAL, KUKA|prc, Robot Components, Robots (by Visose), FU-Robot, and Machina [9] that makes it a serious tool for robotic control. However, despite their utility, none of these plugins currently support collision-free path planning, primarily due to the complex nature of the task and the substantial computational power required. Real-time, collision-free path planning remains a significant challenge within these design environments. The intricacy of architectural geometries, parametric setups, the presence of dynamic elements such as moving objects and actors, and the high level of customization required, demand more advanced solutions. These solutions must be capable of managing complex design constraints while providing immediate feedback to designers.

3 cuRobo

A notable advancement in the area of automatic collision-free path planning for robots is NVIDIA's cuRobo, a GPU-accelerated library [2]. By leveraging the parallel processing capabilities of GPUs, cuRobo formulates motion planning as a global optimization problem, enabling the simultaneous evaluation of

multiple potential solutions. This parallelism allows cuRobo to solve complex motion generation tasks in approximately 50 milliseconds on average, achieving speeds up to 60 times faster than traditional trajectory optimization methods. It's efficiency stems from combining straightforward optimization techniques with numerous parallel starting points, or "seeds." It utilizes the L-BFGS method—a popular optimization algorithm for estimating step directions—alongside a novel parallel noisy line search strategy and a particle-based optimization solver. Additionally, the algorithm incorporates a parallel geometric planner capable of generating plans within 20 milliseconds and a collision-free inverse kinematics (IK) solver that can process over 7,000 queries per second. These capabilities make cuRobo a powerful tool for real-time robotic applications, including those in the AEC industry, where rapid and reliable motion planning is essential for tasks like automated fabrication and assembly.

The software requires installation on a robust system equipped with a CUDA-capable NVIDIA GPU of Volta or newer architecture, operating on Ubuntu 20.04 or 22.04, although it also works on Debian 11. A Python environment with version 3.10 is recommended, and the software depends on PyTorch version 1.15 or newer. These requirements, including the reliance on a Linux operating system and high-performance hardware, present significant barriers within the architectural and construction industries, where such setups are not commonly available. Additionally, the simulation environment, based on NVIDIA Isaac, a platform designed for developing and deploying robotics applications, further complicates testing different configurations due to its specialized requirements, such as loading Universal Robot Description Format (URDF) files, and the need for high-level technical expertise to adapt it to real-world scenarios. To address these challenges and use the paralleled speed and precision of this framework, we propose a novel setup that integrates high-performance computing with GH for AEC professionals.

4 Grasshopper – cuRobo Setup

The system setup integrates three interconnected software frameworks: cuRobo, Grasshopper and MQTT (Figure 1). **cuRobo's** AI-powered system is equipped with a Python API. It operates by loading a configuration file tailored to a specific robotic setup, such as the Universal Robots UR10e and others, to enable seamless integration and control. It operates headless by just responding with the calculated results (joint values, environment, error messages etc.). **Grasshopper in Rhino3D** (GH) serves for the visual and parametric interaction of the environment (world and robot) with the user as well for the control of the real robotic system. **MQTT**-communication framework stands between the two systems that assures low-level, easy to implement exchange of data. While the MQTT-broker Mosquitto [10] is installed on Debian Workstation, MQTT-clients were implemented inside cuRobo python code and on GH side in form custom build MQTT components, both based on Paho-clients.

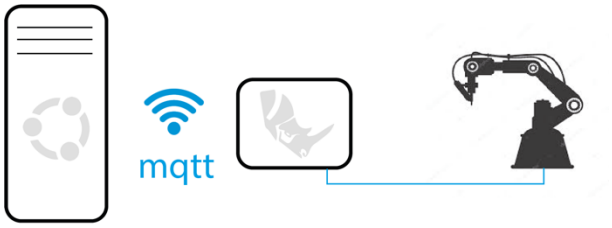

Figure 1. Scheme of the system setup: Left – Debian GPU workstation with cuRobo, connected to a GH user computer via an MQTT connection and to the robot via cable.

The backbone of the whole concept is cuRobo API that comprises interconnected modules designed for efficient and precise robotic motion planning. Key components include optimization solvers (curobo.opt) for refining trajectories, kinematics handling (curobo.cuda_robot_model) for forward and inverse computations, and collision detection (curobo.geom) for safe motion planning. Modules like curobo.graph enable geometric pathfinding, while curobo.rollout evaluates motion strategies by mapping actions to costs. High-level wrappers (curobo.wrap) simplify task programming, including collision-free reaching, and curobo.types standardizes data structures. The curobo.wrap.reacher module provides a high-level interface for programming tasks involving collision-free reaching. It simplifies the process of generating safe and efficient motion trajectories, enabling seamless implementation of reaching tasks within various robotics workflows.
Robots are represented in URDF files, along with their collision spheres in an additional YAML file. Another configuration YAML file defines the entire additional setup, as well as the dimensions of the optional end-effector. The world in cuRobo is also represented in this manner but can be programmatically reloaded after the system is initialized. This is important because the elements can change positions in a dynamic environment. Once initialized, the running program waits for a path creation command and returns either a successful path or a message explaining the reason for failure. Other commands update the world or read the collision spheres for improved visualization (Figure 2).

Figure 2. Rhino 3D / GH visualization of 1) Collision spheres of the robot; 2) Collision spheres of the attached stick; 3) World collision object of the cable connection of the robot.

GH handles the visualization and control of the setup. Inside GH, we read the shared robot URDF file and display the output of the resulting joint positions calculated by cuRobo. Additionally, GH is used for world creation, which is reflected back to cuRobo. Visualization and path rehearsal (Figure 4 and 5) are crucial for ensuring safe execution, especially in complex design environments. We enable designers to preview paths, identify potential issues, and refine movements before execution. This prevents costly errors and enhances user confidence. Additionally, we can plan attaching elements to the end-effector and send this configuration back to cuRobo, which incorporates the attached element into collision-free path planning. After detaching, the new world is updated with the element's new position. For this, a unique URDF reader for GH has been developed and incorporated (Figure 3). This makes it much easier to build novel robotic setups as well as collision sphere setups.

The third part is the communication system, whose purpose is to simply incorporate subscription and publisher capabilities on both sides. MQTT can be optimal for secure and rapid communication between subscribers and publishers in the form of topics. It requires an MQTT broker, which most platforms provide, and clients are widely available. For the case study, we incorporated four request topics and five response topics. Depending on the network quality and data size, it will deliver an almost immediate response (see Latency chapter).

The control inside GH consists in these three elements:

Exchange procedures GH with cuRobo. Here we can send the world setup, this means all objects in the real or simulated world, from GH to cuRobo in a form of a list of elements with their dimensions and poses. Vice versa, we can pull the world from cuRobo to cross-check how it sees the world. Furthermore, we can also pull information about the collision spheres for cross-checking the YAML file. This helps a lot also to identify the position of the robot from the cuRobo side.

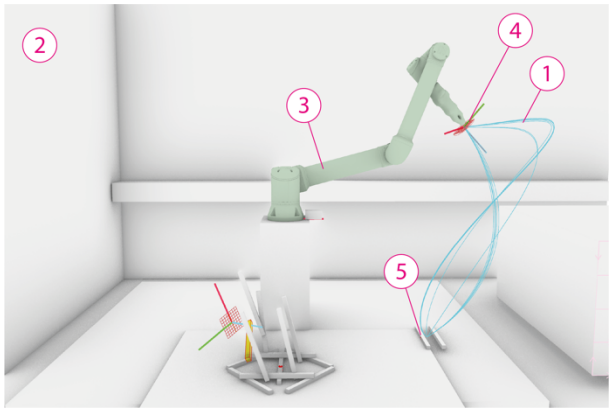

Figure 4. Representation of the real world translated in Curobo and visualized in Rhino3D / GH. 1) automated path options considering the world and self-collision; 2) world perception of Curobo; 3) Robot representation from URDF-file and cuRobo joint positions; 4) start position; 5) goal position

Figure 5. the original scenery

Path queries from GH to cuRobo in form of sending the initial start position (joints) and the desired goal position (plane, position, and pose). This fires back an array of joint positions that can be previewed. Every query will generate a unique new collision-free path. It considers the environment and prevents self-collision. In addition to the path query, we can attach or detach objects to or from the end-effector, in the case of gripping an object or similar. In this case of attaching, the object is removed from the world, and additional collision spheres will be created in the volume of the object as a part of the robot. All subsequent movements of the robot take these new collision spheres into account. In the case of not finding a valid solution for the target pose, start or goal collision states, or other failings, we can provide GH with feedback from cuRobo.

Robot control commands send the calculated new joints to the robot. Here, we can also add other commands, such as gripper control, depending on the robot being used.

5 Latency

The great advantage of cuRobo is its speed. In contrast to other motion planners described in the previous chapter, which often take several seconds to optimize a trajectory, cuRobo requires only 1/5–1/10 of a second. Black Coffee Robotics [11] compared RRTConnect-based MoveIt2 (MI) and cuRobo. In an obstacle-free environment, the results are similar (cuRobo/MI – 100%/96% success rate – 0.19 sec./0.17 sec.). In an environment with obstacles, cuRobo outperformed MI in terms of reliability and trajectory quality (cuRobo/MI – 90.32%/62.50% success rate – 0.69 sec./0.33 sec.).

The response in a system with multiple elements, as described in this paper, is dependent on several factors, such as the computers used and the connection between them. In our setup, cuRobo runs on a Linux workstation with an Nvidia RTX A4000 GPU with 16GB RAM. The computers are connected over a VPN network with a minimum 200 Mb/s download speed using the MQTT framework. GH is running on a MacBook Pro (2021). The average path calculation measured in GH, from activating the toggle until receiving the joint positions, is approximately 200–250 ms (Figure 4). With such speed, an almost continuous flow preview of possible paths for moving elements is possible.

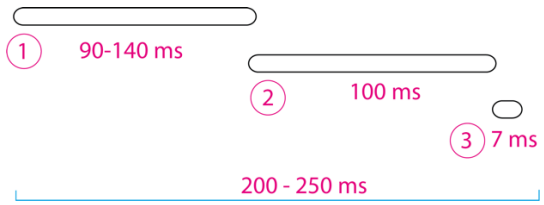

Figure 6. Approximated processing time 1) Path calculation (Python API time measure); 2) MQTT transfer (Mqtt-explorer timestamp); 3) GH path creation (GH profiler).

6 Case Study

To test the automatic collision-free path planning with conventional path planning, we required a sufficiently complex structure that went beyond the typical simple stacking examples. For this purpose, we utilized an ongoing collaborative project with the University of Tokyo. The task involved constructing a wooden structure composed of identical modular elements measuring 18x18x300 mm. In this "classical path planning scenario," two robotic setups were established in Japan and Germany to test remote collaboration scenarios for robotically assembled structures. In each facility, a collaborative robotic arm (cobot - respectively UR10E and UR10) was installed in front of a wooden base on which to perform the assembly. Both setups' gripper end effectors were the OnRobot RG6 finger gripper. The different tool mounting technology caused slight variations in the tool center points. Despite these small differences in the robots' end-effector positions relative to their frames of reference, both setups had very similar reachable workspaces.

In this modular construction system, the sticks are glued together along one of their long faces to form tree-like structures. The simplicity of the assembly rules, paired with the complex architectural results they create, makes it an excellent platform for experimenting with robotic assemblies [11]. The outcome of the original tests was a twin structure in both locations together with the design and fabrication data that was exchanged through a synchronized JSON file stored on a shared Dropbox folder. The JSON file included geometrical information

that was used to build another twin structure [12].

We replicated the experiment of the structure assembly of the first 12 sticks, and instead of manually or parametrically constructing the path, we relied solely on the cuRobo path-planning capabilities. The 3D model was built in GH, and all sticks were positioned at the pick position. The assembly consisted of several parts where all movements and attaching/detaching were made by cuRobo:

1. Movement from home to one of three offset gripping positions of the stick to be picked.
2. Opening the gripper and a movement towards the picking position.
3. Closing the gripper and a movement to the offset position.
4. Virtual attaching the stick to the robot.
5. Movement towards the offset placing position.
6. Movement to the placing position.
7. Opening the gripper and virtual detaching the stick from the robot.
8. Movement to the offset position.
9. Movement back to the home position.

All movements had their preview state in the form of a curve to ensure that the movements were possible and within the desired area. As soon as a preview was validated as valid, it was sent to the robot for execution. The gripper openings and closings were controlled manually through the same GH controller.

The movements were extremely accurate and immediately available. As we had the previews of the movements, we could execute the robot without fear of colliding with other objects, even with the stick in the gripper (Figure 5). Sometimes, it moved very near to objects, which made us nervous.

After the fifth stick, a collision still happened. We forgot to account for the cable connection to the robot that was facing towards the sticks. The joint position of the shoulder squeezed the connector and disconnected it from the power. We had to release the joints manually and rotate the shoulder away. After we integrated the connector into the world (Figure 3), this unusual position never repeated.

As we tested the cuRobo URDF file of a UR10e robot on a real UR10 robot, discrepancies in the calculated positions limited us to only 12 sticks. These discrepancies arose because the base of the UR10e is 15 mm higher than that of the UR10, along with some minor differences in arm length and will be improved in the next tests.

This specific case study demonstrated the technical capabilities of cuRobo in generating fast and precise collision-free paths, seamlessly integrating with Grasshopper for real-time visualization and robotic control. However, limitations emerged, such as discrepancies between simulated and real-world robot models, the need for manual adjustments to account for unmodeled elements like cables, and the lack of real-time dynamic obstacle handling.

7 Conclusion and Discussion

This study successfully integrates advanced AI-based software for collision-free path planning into an architectural workflow, offering significant potential for robotic automation in the AEC industry. By combining the NVIDIA cuRobo framework with Grasshopper in Rhino3D, the system eliminates the need for complex environments like NVIDIA SIM, streamlining the process and lowering adoption barriers. The case study demonstrates the framework's precision and efficiency in generating accurate collision-free paths, enabling complex assembly tasks with minimal intervention. Real-time visualization further empowers users to plan and execute robotic movements confidently.

The system excels in speed, precision, and accessibility. The GPU-accelerated cuRobo framework outperforms traditional motion planners, generating results in near real-time. Its integration into Grasshopper ensures AEC professionals can work within familiar environments, enhancing usability. Additionally, the system's ability to adapt to dynamic changes, such as object attachment and detachment, underscores its flexibility for construction applications.

Despite these strengths, challenges remain. The dual-hardware setup and the need for a CUDA-capable Linux system pose barriers for widespread adoption. Compatibility issues, as seen in the discrepancies between UR10e and UR10 hardware, highlight the need for precise configuration, and oversights in collision modelling, such as the robot's cable connection, demonstrate the importance of comprehensive planning.

Future research will focus on incorporating real-time scanning and dynamic feedback to handle variable environments, as well as developing intuitive control methods like voice or semantic commands. Expanding compatibility to include more robot models and releasing an open-source library is planned.

Acknowledgments

We gratefully acknowledge the funding of the project by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 519047184.

References

[1] Latombe, J.-C., Robot Motion Planning. Springer US, Boston, MA., 1991

[2] Sundaralingam, B., Hari, S.K.S., Fishman, A.,

Garrett, C., Wyk, K.V., Blukis, V., Millane, A., Oleynikova, H., Handa, A., Ramos, F., Ratliff, N., Fox, D., cuRobo: Parallelized Collision-Free Minimum-Jerk Robot Motion Generation, 2023

[3] Cui, C., Wang, Z., Sui, J., Zhang, Y., Guo, C., An improved RRT behavioral planning method for robots based on PTM algorithm. Sci Rep 14, 21776, 2024

[4] Ye, X., Guo, H., Guo, Z., Luo, Z., A construction robot path planning method based on safe space and worker trajectory prediction. ISARC Proceedings 2024 Proceedings of the 41st ISARC, Lille, France, pages 227–235, 2024

[5] Liu, L., Wang, X., Yang, X., Liu, H., Li, J., Wang, P., Path planning techniques for mobile robots: Review and prospect. Expert Systems with Applications 227, 120254, 2023

[6] Collision-Free Motion Planner - RoboDK Documentation [WWW Document], n.d. URL https://robodk.com/doc/en/Collision-Avoidance-Collision-Free-Motion-Planner.html (accessed 10.24.24).

[7] Motion Planning — MoveIt Documentation: Rolling documentation [WWW Document], n.d. URL https://moveit.picknik.ai/main/doc/concepts/motion_planning.html (accessed 12.25.24).

[8] Planning scene and collision objects — COMPAS FAB [WWW Document], n.d. URL https://compas.dev/compas_fab/latest/examples/03_backends_ros/05_collision_objects.html (accessed 12.25.24).

[9] Foo4Rhino [WWW Document], n.d. . Food4Rhino. URL https://www.food4rhino.com/en/browse (accessed 12.18.24).

[10] Eclipse Mosquitto [WWW Document], 2018. . Eclipse Mosquitto. URL https://mosquitto.org/ (accessed 12.25.24).

[11] Kaneko, T., Khajehee, A. & Ikeda, Y. Heuristic Fabrication: An Interactive Robotic Building System for Enhancing Human Participation in Timber Structures. (2024).

[12] Khajehee, A., Rogeau, N., Abdelaziz, M., Kotov, A., Starke, R. Vukorep, I. & Ikeda, Y. Collaborative Exploration of Complex Design Spaces: Integrating constraints from two remote robotic setups. Proceedings of CAADRIA, Tokyo, Japan (2025) (In Press)