Robotic Path Planning Algorithms for Additive Manufacturing Using Advanced Simulation Tools

Marco Pereira[1,*], Hugo Costelha [1,2], Luís Conde Bento [1,3] and Carlos Neves [1,2]

[1] ESTG - Polytechnic University of Leiria, [2] INESC Coimbra, [3] ISR Coimbra, * Correspondence: luis.conde@ipleiria.pt

Abstract:

Robotics-based additive manufacturing (AM), or 3D printing, enables flexible printing systems. This paper analyses path planning algorithms for robotic manipulators aiming at dynamic AM environments. Using the cuRobo library, the study evaluates the path planning algorithm MotionGen and Model Predictive Control (MPC) using NVIDIA's Isaac Sim with ROS2 and MoveIt2. Docker provided a modular development environment, and an Intel RealSense camera was used to enable real-world and real-time obstacle detection.

Results show that MotionGen outperforms MPC in energy consumption and time efficiency, generating smoother and more efficient trajectories, more suitable for real-time AM contexts. The project shows the potential of advanced robotic control algorithms to optimize AM, using NVIDIA's Isaac platform. Future work will focus on applying this to real robots.

Keywords: Advanced Robotic Simulation; Dynamic Environments; Collaborative Environments; Trajectory Generation.

Acknowledgment: This work is carried out within the framework of the "Agendas para a Inovação Empresarial" (Project no 49, acronym "INOV.AM", with reference PRR/49/INOV.AM/EE, operation code 02/C05-i01.01/2022.PC644865234-00000004), supported by the RRP - Recovery and Resilience Plan and by the European Funds NextGeneration EU. This project was partially financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/00308/2020. This project was partially financed by National Funds through the Scientific Employment Stimulus - Institutional Call CEECINST/00051/2018.

I. INTRODUCTION:

Additive manufacturing (AM) is a transformative approach to industrial production allowing for the creation of parts in small batches or single unit productions. Manipulator robots, with their capability for complex and repetitive tasks with precision and reliability, are fundamental in many industries, from manufacturing to healthcare, including AM. Furthermore, the use of robotic manipulators in AM brings greater flexibility and non-planar multidirectional prints [1].

The motivation stems from the need for improvements in trajectory generation to get energy efficient, safe, and precise operations, crucial as industrial demands rise. The aim is to test algorithms for robotic manipulators to be used in future additive manufacturing processes in dynamic and collaborative environments. The key objective is to evaluate the precision and efficiency of robotic trajectory generation using advanced simulation platforms and algorithmic tools [2].

The objectives include training with new tools, contributing to the study of trajectory generation techniques for robotic manipulators, and evaluating the tools in various scenarios. This involved leveraging advanced simulation platforms such as NVIDIA's Isaac Sim [3], and algorithmic tools like cuRobo [4], to explore, test, and compare different approaches.

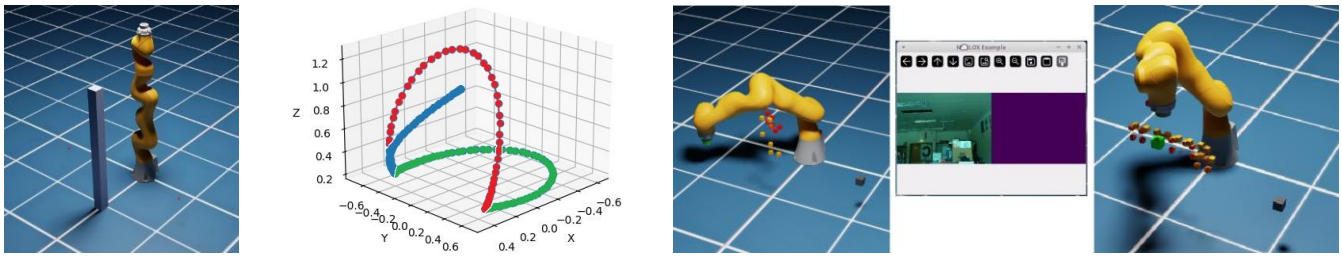
II. ROBOTIC PATH PLANNING ALGORITHMS:

This study explores and implements various trajectory generation and collision detection techniques in robotic systems, using tools available within the Isaac environment. The main goal was to generate smooth and efficient robot trajectories that adapt dynamically to environmental changes [5]. For that purpose, two libraries were studied: Lula and cuRobo.

A. Implementation of Trajectory Generation Algorithms

Lula is a high-performance motion generation library designed for robotic manipulation, providing algorithms and tools for tasks such as forward and inverse kinematics, global path planning, and smooth reactive motion generation:

- **Lula-RMPflow with Riemannian Motion Policy**, was configured within Isaac Sim to accommodate the Franka Emika Panda robot. This involved modifying the default settings of RMPflow [6], originally configured for a six-joint Cobotta Pro 900 robot, to support the Panda's seven-joint structure. RMPflow shown the ability to generate trajectories in static obstacle environments, where the Panda robot was able to dynamically follow a red cube while avoiding obstacles. Although it can generate trajectories in the presence of dynamic obstacles, only static environments were evaluated.
- **Lula-RRT, Rapidly Exploring Random Tree (RRT)** [7] algorithm, has limitations in dynamic and unstructured environments, where obstacles may be non-uniformly distributed. The main drawbacks include slow extension speeds for node exploration, which makes it inadequate for real-time systems, and the generation of non-smooth trajectories with numerous unneeded interruption points, resulting in abrupt curvatures.



(a) Static. **(b)** Path with static (S1 Green, S2 Red and S3 Blue). **(c)** Dynamic captured by camera.

Figure 1. Experiments scenarios with obstacles.

cuRobo offers a suite of tools for tasks such as forward and inverse kinematics, collision checking, trajectory optimization, and motion generation. By utilizing GPU acceleration, cuRobo achieves significant speed improvements over traditional CPUbased implementations, enabling real-time motion planning and control for robotic systems:

- cuRobo-MotionGen generates collision-free trajectories using batched inverse kinematics (IK) to find multiple joint configurations for a target pose. It applies trajectory optimization with linear interpolation from the start configuration to each IK solution. If it fails, a graph planner finds collision-free paths used later as seeds for further optimization.
- cuRobo-MPC MPC allows for continuous trajectory adjustments at each program cycle, using reactive behaviour as the target position changes. An example implementing MPC was adapted for trajectory generation. MPC's local optimization approach limits its ability to avoid larger or complex obstacles. Moreover, the algorithm sometimes struggles with real-time adaptation to rapid obstacle movements due to the lag in data acquisition and voxel creation.

The cuRobo library was selected specifically with MotionGen and the MPC algorithms. Initial experiments were done using basic motion generation examples. E.g., in a virtual setup, the Kuka iiwa robot tracked a target cube, generating trajectories as the cube moved. Unlike RMPflow, cuRobo was more flexible and adaptable to real-time scenarios.

B. Experimental Platform and Testing Scenarios

Virtual Platform: In Isaac Sim, a virtual Kuka iiwa LBR iiwa 7 R800 robot was tested in environments with and without obstacles. This allowed for extensive testing of both MotionGen and MPC algorithms under varying conditions: environments without obstacles, static obstacles, and dynamic obstacles detected via a depth camera.

Physical Platform: A real physical setup was also built, with a fixed RealSense D435i camera and an obstacle placed at a consistent distance from the robot. The depth camera generated real-time voxel maps through the nvblox tool, transforming depth images into Euclidean Signed Distance Fields (ESDFs). This enabled precise detection of static and dynamic obstacles (Figure 3), easing real-time adjustments to robot trajectories. This setup ensured controlled, repeatable testing scenarios, providing comparative data across different experimental runs.

The experiments were conducted to test trajectories under various conditions. A virtual and real obstacle, represented as a cuboid in the virtual world (see Figure 3(a)), was positioned to intersect the shortest path between target positions.

C. Data Collection and Analysis

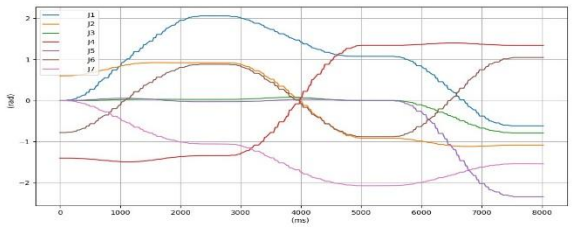
The robot's movements, velocity, and acceleration on each joint were logged for assessment [8]. During the experiments, data was systematically collected and stored in CSV files. Key metrics included joint positions (in Cartesian coordinates and radians), velocities, accelerations, and timestamps. Two separate tables of data were generated:

- Trajectory Planning Metrics: Recorded for each planned trajectory, providing insight into the planning effectiveness
- Execution Metrics: Monitored in real time, showing the actual path, enabling comparison with the planned trajectory.

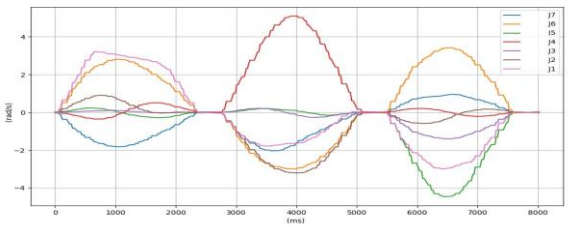
III. EXPERIMENTAL RESULTS:

The work's results analyse the performance of MotionGen and MPC algorithms in different scenarios, with various obstacle configurations. Metrics such as execution time, path planning duration, travelled distances, joint positions, and velocity, were recorded to evaluate algorithm's efficiency and accuracy. As depicted in Figure 3(b), the path was divided into three sections: section 1 (S1), in green colour; section 2 (S2), in red colour; and section 3 (S3), in blue colour.

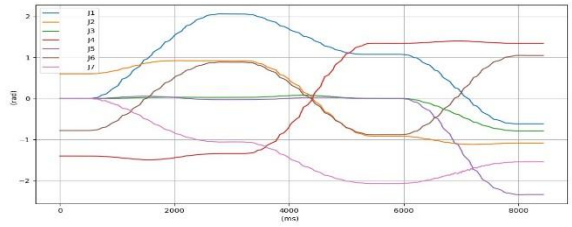
A. Scenario without Camera



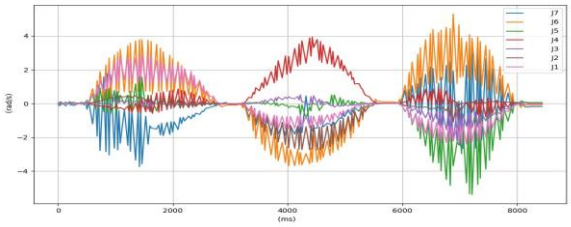
(a) Planned joint paths.



(b) Planned joint velocities.

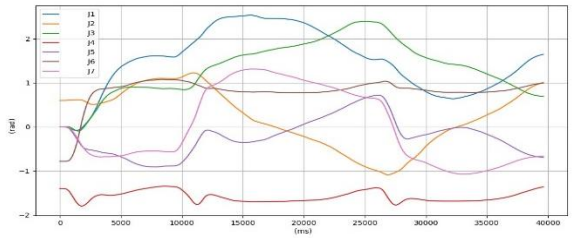


(c) Executed joint paths.

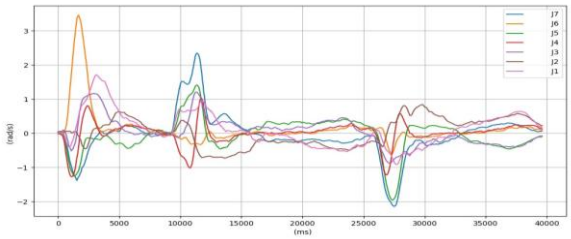


(d) Executed joint velocities.

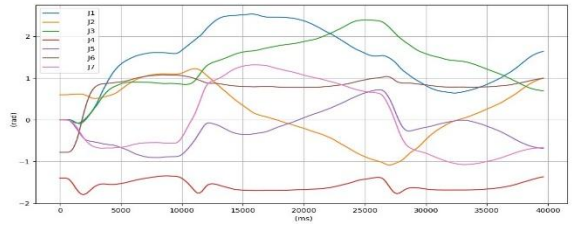
Figure 2. MotionGen - representation of the joint velocities along and path using RealSense to detect an obstacle.



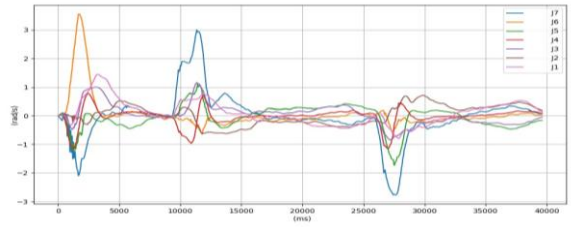
(a) Planned joint paths.



(b) Planned joint velocities.



(c) Executed joint paths.



(d) Executed joint velocities.

Figure 3. MPC - representation of the joint velocities along and path using RealSense to detect an obstacle.

Environment without Obstacles: In an obstacle-free environment (with no camera), the MotionGen algorithm had an average planning time of 61.46 ms and an average execution time of 1.6 s. The planned and executed trajectories shown small variations due to the probabilistic nature of the optimization, leading to different paths for identical conditions. S3 was more energy efficient in terms of cumulative joint travel distance (6.91 rad), compared to S2 (8.04 rad), while S2 covered a shorter end-effector distance (1.23 m compared to 1.47 m for S3). Joint positions varied as expected during the execution, the robot reached target positions within the expected time maintaining compliance with manufacturer-defined limits. Comparison of planned and executed trajectories indicated that speed limits were exceeded, suggesting possible configuration issues in the robot's setup or algorithmic limitations.

Using the MPC algorithm in the same environment yielded an average execution time of 7.6 seconds. S3 was again the most efficient, minimizing both end-effector distance and joint movement energy. The joint speed constraints were only respected during planning, with several joints exceeding speed limits during execution, particularly for Joints 6 and 7.

Environment with Fixed Obstacles: The fixed obstacle scenario tested the adaptability of MotionGen and MPC algorithms in a simulated environment with obstacles. MotionGen achieved successful navigation, with an average planning time of 62.15 ms and execution time of 1.65 s. S3, the most efficient, covered 2.6 m with 7.2 rad of joint travel, with S2 requiring additional joint rotations due to a vertical detour.

For MPC, trajectories were executed in an average of 7.3 s. S2, covering a greater end-effector distance (1.32 m), was less energy-efficient than S3, which minimized joint movements to 10.58 rad. The planned paths were strictly followed, and no joint position violations occurred. However, minor deviations in joint velocities led to slight speed limits breaches in Joints 6 and 7 during execution.

TABLE I: Experimental Results (S#W - With Obstacles; N/RS - No RealSense; W/RS - With RealSense).

Section #	Exec.Time (ms)		Dist.Covered (m)		ΣJoint Dist. (rad)	
	N/RS	W/RS	N/RS	W/RS	N/RS	W/RS
MotionGen						
S1	1.52	1.86	1.20	1.20	5.56	5.56
S2	1.78	2.11	1.23	1.23	8.04	8.01
S3	1.50	1.85	1.47	1.46	6.91	6.99
S1W	1.49	2.57	1.20	1.19	5.56	5.60
S2W	1.51	2.61	3.16	3.16	8.62	8.65
S3W	1.94	2.43	2.60	1.69	7.20	8.00
MPC						
S1	5.56	6.75	1.12	1.12	8.15	8.15
S2	10.44	12.84	1.43	1.43	11.58	11.58
S3	6.90	8.47	1.28	1.28	10.13	10.13
S1W	5.45	9.23	1.12	1.12	8.15	8.15
S2W	10.05	16.47	1.32	1.29	12.50	12.19
S3W	6.52	13.74	1.23	1.28	10.58	11.30

TABLE II: Average Experimental Results (Y - With Obstacles; N - No Obstacles; N/RS - No RealSense; W/RS – With RealSense).

Obst. N/Y	Exec.Time (ms)		Dist.Covered (m)		ΣJoint Dist. (rad)	
	N/RS	W/RS	N/RS	W/RS	N/RS	W/RS
MotionGen						
N	1.60	1.94	1.30	1.30	6.84	6.85
Y	1.65	2.53	2.32	2.01	7.13	7.42
MPC						
N	7.63	9.35	1.28	1.28	9.95	9.95
Y	7.34	13.14	1.22	1.23	10.41	10.55

B. Scenario with Camera

Environment without Obstacles: In experiments with the RealSense camera for obstacle detection, MotionGen showed consistent planning times (61.46 ms) and larger execution times (1.94 s). S3 was the most efficient, with a lower joint travel distance of 6.99 rad. The planned and executed trajectories agree, despite minor deviations in joint velocities (some joints exceeded limits).

The MPC algorithm, in this test, averaged 9.35 seconds for execution, with S3 demonstrating optimal efficiency. There was an accurate execution of planned paths, and the analysis of velocity profiles indicated minor amplitude differences, especially for Joints 5 and 7, though speed limits were largely maintained.

Environment with Fixed Obstacles: With a fixed obstacle present, MotionGen's performance saw average planning times of 64.9 ms and execution times of 2.5 s. S3 minimized joint movements (8 rad), while S2 shown lager end-effector distance (3.16 m) due to a vertical detour. The results confirmed the algorithm's ability to plan and follow obstacle-avoiding paths. Both planned and executed trajectories were consistent despite the presence of obstacles, as seen in Figure 2(a) and Figure 2(c), planned and executed joint paths respectively. As shown in Figure 2(b) and Figure 2(d), respectively the planned and joint velocities, the executed joint velocities differ from the planned ones. The executed joint velocities show high-frequency oscillations, suggesting that the algorithm might be compensating for disturbances or control inaccuracies.

For the MPC case, execution times averaged 13.14 seconds. S3 minimized energy expenditure with reduced cumulative joint travel (11.3 rad). The alignment between planned and executed paths was clear. Figure 3(a) and Figure 3(c) present the planned and executed joint paths, illustrating the similarity between planned and executed paths, confirming the accuracy of the MPC approach. The velocity analysis showed minor breaches in speed limits, particularly for Joints 6 and 7, though the planned trajectories adhered closely to the constraints. Figure 3(b) and (d) show that the planned and executed joint velocities, respectively, have minor differences exist between the planned and executed joint velocities, particularly in for Joints 5, 6 and 7.

Environment with Dynamic Obstacles: These tests were not completed due to limitations in the simulation software's ability to adjust obstacle positioning in real-time, preventing accurate performance assessment in this scenario.

C. Results Summary

For a comparative analysis of the main findings across all scenarios, table I summarizes the execution times, distances travelled by the end-effector, and cumulative joint distances for both algorithms, both with and without the RealSense camera and obstacles. This aids in clarifying the strengths and limitations of each approach, as well as their impact on system performance:

- **Execution Time:** Across all scenarios, the MotionGen algorithm sown shorter times than MPC. The inclusion of RealSense increased execution times for both algorithms, notably the impact in scenarios with obstacles.
- **End-Effector Travel Distance:** They were consistent within each algorithm across scenarios. As expected, the introduction of obstacles led to longer end-effector paths, particularly with the MotionGen algorithm.
- **Cumulative Joint Distance:** MotionGen presented smaller distances than MPC across the tests, showing better energy efficiency. As expected, the cumulative joint distance increased in scenarios with obstacles for both algorithms, due to additional manoeuvrers required to avoid collisions. Using the RealSense camera, however, did not impact cumulative joint distances.
- **Energy Efficiency:** The MotionGen algorithm was more energy-efficient across scenarios, as shown by its lower cumulative joint distances. Obstacle-free scenarios were more efficient in terms of energy for both algorithms since obstacle avoidance inherently demands more joint movements.

- **Mean Execution Time**: MotionGen shown faster execution times across all scenarios than MPC. As expected, the RealSense camera's inclusion increased execution time, especially with obstacles. The MPC algorithm took longer than MotionGen, with increased durations seen when RealSense and obstacles were used together.
- **Mean End-Effector Distance:** The end-effector travel distance remained stable for MotionGen at 1.3 m, regardless of the presence of obstacles or RealSense use. In contrast, the MPC algorithm showed greater end-effector distances, mainly in tests with obstacles and no RealSense.
- **Mean Cumulative Joint Distance:** MotionGen got always lower cumulative joint distances, showing better energy efficiency than MPC. MPC's joint distance was higher, increasing with obstacles and RealSense use.

Further analysis of joint position graphs from the MotionGen algorithm identified the precise moments when the robot reached the target, which was not observable in the MPC produced graphs. In the two experiments, joint position limits were consistently respected. However, velocity limits were exceeded in every scenario, with none of the algorithms fully adhering to these constraints. The trajectory optimization process, influenced by the Model Predictive Path Integral (MPPI) technique, also resulted in different trajectories even under identical scenarios, underscoring the stochastic nature of the trajectory generation process.

IV. CONCLUSIONS AND FUTURE WORK:

Concluding, this study highlights the comparative performance of the MotionGen and MPC algorithms for trajectory generation in robotic manipulators under various scenarios. The results show that MotionGen consistently outperforms MPC in terms of path execution speed and energy efficiency, as demonstrated by shorter execution times and lower cumulative joint distances. The presence of obstacles and the use of RealSense cameras slightly increased execution times and distances covered, as expected. Yet, MotionGen remained more efficient than MPC. These findings underscore the strengths of MotionGen for static environments, while MPC's dynamic obstacle handling is beneficial in dynamic conditions. Future work will explore the enhancement of MotionGen's adaptability by integrating trajectory re-planning capabilities to address dynamic obstacles more effectively, which would extend its applicability to a broader range of scenarios. Additionally, a more in-depth analysis of algorithmic parameters and limitations will offer insights into the stability and repeatability of each algorithm's performance under identical conditions. This study also emphasizes the promising role of platforms such as Isaac and the cuRobo library, together with the use of GPU-based computations, in robotic trajectory systems. Further development of cuRobo's features, especially in terms of depth-camera integration and environmental updates, could make it an even more powerful tool for robotics applications.

REFERENCES

[1] B. Ramos, D. Pinho, D. Martins, A. I. F. Vaz, and L. N. Vicente, "Optimal 3D printing of complex objects in a 5–axis printer, "Optimization and Engineering, vol. 23, no. 2, pp. 1085–1116, Jun. 2022.

[2] J. R. Benevides, "Planejamento de rota para manipulador espacial planar de base livre flutuante utilizando o algoritmo rrt," Masters' thesis, Escola de Engenharia de São Carlos, Universidad de São Paulo, São Carlos, Brazil, 2000.

[3] Nvidia isaac sim. Accessed May 2024. [Online]. Available: https://developer.nvidia.com/isaac/sim

[4] NVIDIA. curobo. Accessed May 2024. [Online]. Available: https://curobo.org/index.html

[5] K. Wei and B. Ren, "A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved rrt algorithm," Sensors, vol. 18, no. 2, 2018.

[6] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. D. Ratliff, "Rmpflow: A computational graph for automatic motion policy generation," ArXiv, vol. abs/1811.07049, 2018.

[7] K. Lavalle, "Rapidly-Exploring Random Trees: Progress and Prospects", New York, USA: A K Peters/CRC Press, 2000, pp. 293–308.

[8] A. Jain, H. Singh, R. A. Boby, S. K. Saha, S. Kumar, and S. D. Roy, "Repeatability measurement and kinematic identification of lbr iiwa 7 r800 using monocular camera," in Proc. 4th Int.Conf. on Advances in Robotics, ser. AIR '19. New York, NY, USA, 2020.