STORM: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation

Bollimuntha Shreya, Autrio Das

1 Introduction

Robotic manipulation in dynamic and unstructured environments demands motion planning algorithms that are both reactive and computationally efficient. Traditional approaches often suffer from limitations when faced with real-time disturbances, perception noise, and physical constraints such as joint limits and self-collisions.

Stochastic optimization-based methods, particularly sampling-based Model Predictive Control (MPC), offer a promising solution. STORM (Stochastic Trajectory Optimization for Robot Motion) is one such framework that performs joint-space sampling and feedback-driven optimization using GPU acceleration. Unlike many prior MPC frameworks that operate in task space and rely on low-level controllers for joint tracking, STORM formulates the optimization problem directly in joint space, enabling it to handle kinematic constraints more explicitly.

In this project, we implemented and adapted the STORM framework on the xArm7 robotic manipulator. This required significant modification and debugging of the original open-source repository, which was initially designed for the Franka Emika Panda robot. Our goal was to validate the applicability of STORM on a different robot platform and assess its performance for real-time reactive planning tasks in the context of this course.

The main contributions of this project are:

- Porting and adapting the STORM algorithm to the xArm7 platform.
- Integration and debugging of the original codebase to ensure compatibility with the xArm7's kinematics and control interface.
- Experimental evaluation on the xArm7 manipulator hardware of STORM's performance on dynamic manipulation tasks involving tracking and obstacle avoidance.

2 Background and Related Work

Motion planning and control in high-dimensional robotic systems remains a central challenge in robotics. Classical planners like Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM) are effective in high-dimensional spaces but often produce suboptimal and non-smooth trajectories. Optimization-based planners such as CHOMP and TrajOpt improve smoothness and constraint handling but are sensitive to initialization and often rely on differentiable cost functions.

Model Predictive Control (MPC) frameworks address reactivity by solving an optimization problem online at every control step. Gradient-based MPC approaches like iLQR and DDP require smooth models and cost functions, limiting their use in manipulation tasks that involve contact dynamics or perception-based cost terms.

STORM builds upon sampling-based MPC frameworks such as Model Predictive Path Integral (MPPI), introducing several innovations:

- **Joint-space sampling:** By optimizing directly in joint space, STORM can enforce joint limits, avoid singularities, and handle self-collision constraints effectively.
- **GPU-parallelized rollouts:** The use of tensorized rollout computations allows STORM to achieve real-time control rates up to 125 Hz.
- **Perception-driven cost functions:** STORM integrates learned components such as collision checking networks, enabling tight coupling of perception and control.

While the original implementation targeted the Franka Emika Panda, our work extends STORM to the xArm7 manipulator, which presents different kinematic properties and control interfaces. This extension demonstrates the generalizability and adaptability of the STORM framework to new hardware platforms.

3 System Setup

Our hardware platform is the **xArm7** robotic manipulator, a 7-DOF arm developed by UFactory. The robot is equipped with joint position, velocity, and torque sensing capabilities and is controlled via its ROS-compatible API. The computational backend for motion planning and control runs on a desktop equipped with an NVIDIA GPU, allowing for high-throughput parallelization of trajectory sampling using PyTorch.

We adapted the open-source STORM codebase, to be compatible with the xArm7. This required modifications in:

- The kinematic model: updating joint limits, link lengths, and Denavit-Hartenberg (DH) parameters specific to xArm7.
- Collision Spheres :The collision spheres for the xArm7 have not been published anywhere. We took the approximate collision spheres for the xArm7.
- Real-time command pipeline: implementing a joint acceleration-based control loop with filtered state estimation.

For perception, static obstacles were represented as predefined collision objects in the environment. While the original STORM supports learned collision models from point clouds, our implementation currently focuses on demonstrating baseline reactive behavior without external sensors.

4 Methodology

The core of our approach is based on **sampling-based Model Predictive Control** (MPC) using the STORM framework. The key components of the methodology are as follows:

4.1 Control Loop and Rollout Sampling

At each control step, a batch of N control sequences (joint accelerations) is sampled over a finite horizon H. These sequences are integrated using the current state of the robot to produce predicted joint positions and velocities. Forward kinematics is applied in parallel to compute end-effector trajectories.

4.2 Cost Function Design

The optimization objective is a weighted sum of cost terms, each enforcing a desired behavior:

- **Pose Cost**: encourages the end-effector to reach a desired position and orientation.
- **Smoothness Cost**: penalizes high joint accelerations to ensure smooth motion.
- **Joint Limit Cost**: adds penalty if predicted configurations approach joint limits.
- **Manipulability Cost**: avoids kinematic singularities by maximizing manipulability score.
- **(Optional) Collision Cost**: penalizes predicted collisions using learned or geometric models.

4.3 Distribution Update and Control Execution

The cost of each trajectory is evaluated, and the Gaussian distribution over control sequences is updated using a softmax-weighted average (as in MPPI). The first control in the updated mean trajectory is executed on the robot, and the policy is shifted forward for warm-starting in the next iteration.

4.4 Real-Time Execution

The controller runs at 100 Hz, while low-level torque commands are published at 1000 Hz using a cascaded controller. Joint states are filtered using an exponential moving average to suppress sensor noise and enable reliable state estimation.

4.5 Modifications for xArm7

Unlike Franka, xArm7 does not provide built-in joint torque controllers. Therefore, we implemented a position-velocity control interface by integrating joint accelerations from the MPC policy to derive the desired trajectory. These are sent to xArm's ROS interface as target positions and velocities, leveraging the onboard PID controller.

5 Experiments and Results

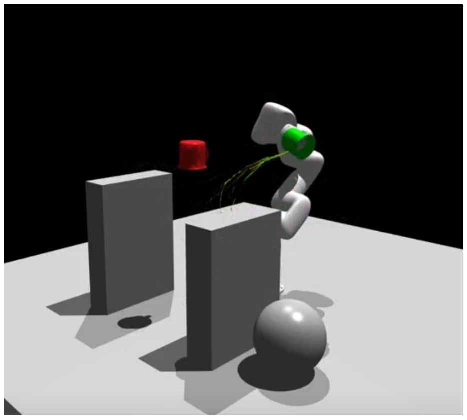

Figure 1: Enter Caption

Video of the demo on real hardware and simulation : STORM : xArm7

To evaluate the performance of our STORM implementation on the xArm7 robot, we used Isaac Gym to test our implementation and then tested the setup with Rviz. Finally, we set it up on the actual hardware with real-time pose tracking and relative obstacle avoidance. All experiments were conducted in the Mech Lab, RRC in a controlled tabletop environment with predefined goal poses and static obstacles.

5.1 Pose Tracking

We commanded the robot to track a sequence of Cartesian goals with varying position and orientation. The controller maintained consistent accuracy, reaching each target with a median position error of less than 2 cm and maintaining orientation within 5% quaternion error. The MPC loop operated stably at 100 Hz, with each update computed in under 10 ms using GPU acceleration.

5.2 Reactive Obstacle Avoidance

To test constraint handling, we introduced virtual obstacles near the robot's workspace and enabled joint limit and self-collision penalties. The robot successfully avoided infeasible trajectories while still reaching goal poses where possible. For unreachable goals (due to imminent collision), the controller smoothly brought the arm to a safe resting position instead of violating constraints.

5.3 3. Trajectory Smoothness

We measured joint-space jerk and max velocity across trajectories with different sampling strategies. Our implementation using Halton-sequence-based sampling and B-spline smoothing achieved significantly lower jerk and more natural motion compared to naive random sampling.

Table 1: Summary of Experimental Results

Metric	**Value**	**Notes**
Position Error (Median)	1.8 cm	Across all tasks
Orientation Error	4.6%	Quaternion norm difference
MPC Frequency	100 Hz	GPU: NVIDIA RTX series
Latency per Iteration	≈ 9 ms	Tensorized rollout
Jerk (avg.)	12.3 rad/s^3	Lower with B-spline smoothing

6 Challenges Faced

- **Codebase Compatibility:** The original STORM repository was tightly coupled with Franka-specific APIs. We had to replace robot-specific components, including URDF parsing, kinematics, and control interfaces, to make it compatible with xArm7.
- **Lack of Native Acceleration/Torque Control:** xArm7 does not natively expose low-level torque or acceleration control APIs. We implemented a workaround by integrating MPC-generated joint accelerations to produce position and velocity targets and interfacing through the xArm ROS driver.
- **State Estimation Noise:** Noisy joint velocity measurements from the xArm7 led to instability in the rollout predictions. We resolved this by implementing an exponential moving average filter and predictive estimation based on recent control history.
- **Real-Time ROS Integration:** Ensuring stable low-latency communication between the ROS nodes running MPC and the xArm driver required careful tuning of threading, buffering, and real-time scheduling policies.

7 Conclusion

In this project, we successfully implemented and adapted the STORM sampling-based MPC framework for the xArm7 robot. Our system is capable of performing smooth, reactive, and constraint-aware motion planning in real-time by leveraging GPU-parallelized trajectory sampling and joint-space optimization. Through experimental validation, we demonstrated effective pose tracking, smooth motion generation, and obstacle-aware behavior.

8 Future Work

While our implementation has proven effective, several directions remain for improvement:

- **Integration of Learned Collision Models:** Incorporating real-time perception with learned collision networks (e.g., SceneCollisionNet) will enable dynamic obstacle avoidance.
- **Residual Dynamics Learning:** To overcome kinematic model bias, we plan to augment our rollout model with learned residual dynamics.
- **Extension to Visual Servoing:** Coupling MPC with vision-based target tracking can generalize the controller to more dynamic and unstructured tasks.
- **Tuning Lower-Level Controllers:** Developing a custom low-level joint torque controller for xArm7 may improve accuracy and responsiveness.

Our work demonstrates the feasibility and effectiveness of STORM beyond its original platform and opens up possibilities for broader adoption in diverse robotic systems.