Real-Time Optimization-Based Planning in Dynamic Environments Using GPUs

Chonhyon Park and **Jia Pan** and **Dinesh Manocha**
University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/ITOMP/

Abstract

We present a novel algorithm to compute collision-free trajectories in dynamic environments. Our approach is general and makes no assumption about the obstacles or their motion. We use a replanning framework that interleaves optimization-based planning with execution. Furthermore, we describe a parallel formulation that exploits high number of cores on commodity graphics processors (GPUs) to compute a high-quality path in a given time interval. Overall, we show that search in configuration spaces can be significantly accelerated by using GPU parallelism.

Introduction

The problem of computing the collision-free motion corresponds to searching for an appropriate path in the free configuration space. It is non-trivial to compute an explicit representation of the free space for high dimensional robots. As a result, most practical methods are based on sample-based planning, that search for a collision free path based on randomized sampling and connecting nearby samples. However, these techniques are mostly limited to static environments or scenes with a priori information about moving objects.

In this paper, we deal with the problem of computing collision-free motion for a high DOF amongst dynamic obstacles. Our approach is general and makes no assumptions about object motion. Rather, we update the position of obstacles in the scene using sensors. To deal with unpredictable environments, we use replanning algorithms that interleave planning with execution (Hsu et al. 2002). Instead of planning and executing the entire trajectory at once, these replanning methods interleave planning and execution threads within a small time interval Δ_t. This approach allows us to compute new estimates on the local trajectory of the obstacles based on latest sensor information. During each planning step, the planner computes an estimate of the position and velocity of dynamic obstacles based on sensor data.

In each planning step, we use an optimization-based planning algorithm (Park, Pan, and Manocha 2012) to plan a trajectory for a short interval. Optimization-based planning reduces trajectory computation to an optimization problem that minimizes the costs corresponding to collision-free, smoothness, and problem specific additional constraints such as kinematic or dynamic constraints. The goal of planning is to find a collision free trajectory from the start configuration to the goal configuration. The start configuration and the goal configuration are defined in the configuration space of a robot. We discretize the solution trajectory to represent it as a set of waypoints equally spaced in time. Similar to previous work (Ratliff et al. 2009), we define the objective function based on waypoint. The cost function includes terms for the obstacle cost, the smoothness cost, and other problem specific constraints.

We parallelize our optimization based search algorithm in two ways. First, we parallelize optimization of a single trajectory by parallelizing each step of optimization by using multiple threads on a GPU. Second, we parallelize optimization of multiple trajectories by using different initial seed values. The parallelism results in two benefits:

- The faster computation allows us to use shorter replanning time intervals which can improve the responsiveness and safety for robots working in fast changing environments.
- We compute multiple trajectories corresponding to different seed values, and thereby explore a broader configuration space to compute a better solution. In other words, we perform multiple searches in parallel to improve the path quality.

The trajectory optimization process and the number of threads used during each step are illustrated in Figure 1. The algorithm uses $(k \cdot m \cdot n \cdot d)$ threads in parallel according to these steps and exploits the computational power of GPUs.

The algorithm starts with the generation of k initial trajectories. Then the algorithm generates m random noise vectors (with dimension d) for all the n waypoints on the trajectory. These noise vectors are used to perform stochastic update of the trajectory. Adding these m noise vectors to the current trajectory results in m noise trajectories, which are slightly modified from the original trajectory. The cost for a waypoint, such as obstacle cost, is computed for each waypoint in the noise trajectories. Smoothness cost, which is computed for each joint, is also computed using parallel capabilities of a GPU. When costs of all noise trajectories are computed, the update of current trajectory is implemented by moving it towards the noise trajectory with low cost. At

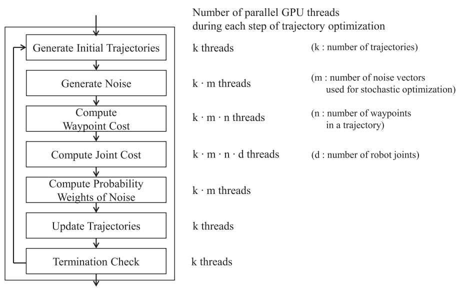
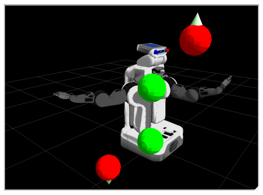
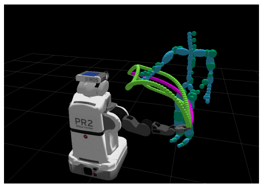

Number of parallel GPU threads
during each step of trajectory optimization
Generate Initial Trajectories
k threads
(k : number of trajectories)
Generate Noise
k · m threads
(m : number of noise vectors used for stochastic optimization)
Compute Waypoint Cost
k · m · n threads
(n : number of waypoints in a trajectory)
Compute Joint Cost
k · m · n · d threads
(d : number of robot joints)
Compute Probability Weights of Noise
k · m threads
Update Trajectories
k threads
Termination Check
k threads

Figure 1: The detailed breakdown of GPU trajectory optimization. It starts with the generation of k initial trajectories. From these initial trajectories, the algorithm iterates over stochastic optimization steps. First it generates random noise vectors which are used for stochastic optimization, then computes the cost for each waypoint on each noise trajectory. We also compute joint cost. The current trajectory cost is repeatedly improved using the cost of noise trajectories until the algorithm satisfies termination criteria.


(a) An environment with two static and two dynamic obstacles.



(b) An environment with a human obstacle.

Figure 2: Planning environments used to evaluate the performance of our planner. The planner computes a trajectory of high DOF PR2 robot arm, which avoids dynamic obstacles and moves horizontally from right to left. The green and red spheres correspond to static and dynamic obstacles, respectively.

the end of each iteration, the algorithm decides to stop the optimization or repeat the next iteration. If a trajectory is classified as being globally optimal, the optimization computation of all trajectories is interrupted and the globally optimal solution is returned. Or if the given time budget is expired, similarly, optimization of all trajectories is interrupted and the best solution is returned.

Results

Our first experiment is designed to estimate the responsiveness of the planner. We plan a trajectory of the 7 degree-of-freedom right arm of PR2 in a simulation environment. In the environment shown in Figure 2(a), there are two static (green) and two moving (red) obstacles. We measure the elapsed time to compute a collision-free solution with varying number of trajectories for both CPU and GPU-based planners. This experiment is performed to compute the appropriate time interval for a single planning time step during replanning. A shorter planning time makes the planner more responsive. The result is shown in Table 1. We observe

Scenario	Average planning time	Std. dev. planning time
CPU 1 core	810	0.339
CPU 2 core	663	0.284
CPU 4 core	622	0.180
GPU 1 trajectory	337	0.204
GPU 4 trajectory	203	0.326
GPU 10 trajectory	60	0.071

Table 1: Results obtained from our trajectory computation algorithm based on different levels of parallelization and number of trajectories (for the benchmarks shown in Figure 2(a)). The planning time decreases when the planner uses more trajectories.

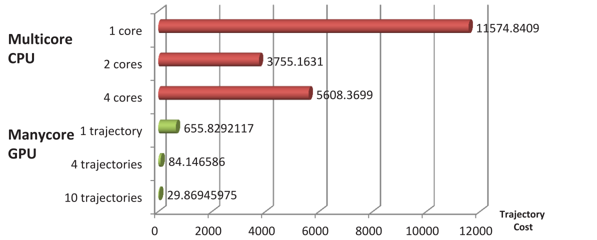


Figure 3: Results obtained from the replanning in dynamic environments on a multi-core CPU and a many-core GPU. The use of multiple trajectories in our replanning algorithm results in trajectories with lower costs and thereby, improved quality.

that the GPU-based planner demonstrates more than 10X speedup over the multi-core CPU-based planner. In both cases, it is shown that when more trajectories are optimized in parallel, the performance of the planner and the quality of resulting paths increases.

In the next experiment, we test our parallel replanning algorithm in dynamic environments with a high number of moving obstacles. The obstacles in the environments change their velocities periodically. However, this information about the obstacles is not known to the planner. The planner uses a replanning technique to reach the goal while avoiding collisions with the obstacles. We observe a different level of responsiveness between CPU and GPU-based planners. When the obstacles move at a high speed, the CPU-based planner may not be responsive. Moreover, we measure the cost of computing the entire solution trajectory, including robot execution. The cost used in this experiment consists of two costs, obstacle cost and smoothness cost. We measure the cost with varying number of optimized trajectories in order to measure the effect of parallelization. Figure 3 highlights the performance. As the number of optimized trajectories increases, the overall cost of entire trajectory computation decreases. This result validates that the multiple trajectory optimization improves the quality of the solution.

Conclusions

We present a novel, parallel algorithm for real-time replanning in dynamic environments. The underlying planner uses an optimization-based formulation and we obtain significant speed-ups and improved reliability by exploiting the parallelism on many-core GPUs.

Acknowledgments

This research is partially supported by NSF grants 1000579, 1117127, and 1142382.

References

Hsu, D.; Kindel, R.; Latombe, J.-C.; and Rock, S. 2002. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research* 21(3):233–255.

Park, C.; Pan, J.; and Manocha, D. 2012. ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Proceedings of the International Conference on Automated Planning and Scheduling, to appear*.

Ratliff, N.; Zucker, M.; Bagnell, J. A. D.; and Srinivasa, S. 2009. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proceedings of International Conference on Robotics and Automation*, 489–494.