2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 979-8-3503-9313-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCA57654.2024.00043

MOPED: Efficient Motion Planning Engine with Flexible Dimension Support

Lingyi Huang†, Yu Gong†, Yang Sui†, Xiao Zang†, and Bo Yuan†
Rutgers University, New Jersey, USA†
{lingyi.huang, yu.gong, yang.sui, xiao.zang}@rutgers.edu, bo.yuan@soe.rutgers.edu

***Abstract*—Motion planning aims to compute the high-quality and collision-free robotic trajectory. To solve the planning problems defined in varying dimensional sizes, motion planners, especially sampling-based, are typically computation intensive because of the costly kernel operations, and computation inefficient due to the inherent sequential processing scheme, hindering their efficient deployment.**

To address these challenges and enable real-time highly efficient motion planning, this paper proposes MOPED, an algorithm and hardware co-design for sampling-based motion planning engine with flexible dimension support. At the algorithm level, MOPED proposes a two-stage processing scheme to reduce the frequency and unit cost of collision check. It also fully leverages the spatial information and unique property of planning process to enable low-cost approximated neighbor search. At the hardware level, MOPED proposes a correctness-ensured speculative processing scheme to overcome the serialization problem. It also develop a multi-level caching strategy to reduce data movement and resolve resource conflict.

We demonstrate the effectiveness of MOPED via implementing a design example with CMOS 28nm technology via synthesizing. Compared with the baseline motion planning processors, MOPED brings significant improvement on throughput, energy efficiency and area efficiency.

I. Introduction

Motion planning aims to find an optimal/near-optimal trajectory from the start configuration to the goal configuration without colliding with any obstacles. As a fundamental and core task in the robotic system, motion planning plays a critical role to secure the successful missions of robots in various operational environments. In practice, many real-world scenarios, e.g., navigation in the complex indoor/outdoor environments and manipulation in robot-aided surgery, highly require real-time motion planning. However, due to its nature of intensive computations, generating high-quality collision-free path with short response time, especially for planning problems defined in the high-dimensional configuration space, is very challenging. For instance, many state-of-the-art CPU/GPU-based planners still need tens or even hundreds of seconds [3], [11], [31], [53], [67], [82] to find the movement trajectory for robotic arms with high degree-of-freedom (DoF), bringing challenging operational costs.

Motivated by this severe performance challenge, in this paper we study efficient hardware acceleration for RRT* [44], the seminal and fundamental sampling-based motion planner. Different from search-based motion planner (e.g., A*) that typically only works in the low-dimensional (2D/3D) configuration space, RRT* and its variants can solve both low and high-dimensional planning problems (e.g., 2-13 DoF) in an asymptotically optimal way. Because of this critical capability, to date RRT*-family are widely used in tremendous practical applications, such as arm manipulation, drone navigation, game development and bioinformatics [2], [6], [17], [89].

As an approach randomly building a space-filling tree, RRT* is inherently computation intensive and inefficient because of three reasons. First, it requires extensive collision checks. As indicated in many prior works [4], [60], collision check, which aims to examine the potential intersection between the robot body and environmental obstacles, consumes a large portion of the overall planning time. In particular, because RRT* has to check the possible collisions incurred by the planned movement at different processing stages after each new sampling, the computing demand from performing collision check is even more significant.

Second, RRT* also suffers costly neighbor search. Neighbor search is well known for its potential high computational cost with the presence of large search scope and high dimensionality. Unfortunately, because 1) RRT* always enlarges the search scope for the nearest neighbor as more sampled points are continuously added into the exploration tree; and 2) the distance calculation consumed in the neighbor search is performed in the configuration space, which is typically of high dimension, the cost incurred by searching the neighbor is very significant in RRT*. Moreover, the exploration scheme also requires multiple stages of neighbor search after each new sampling, further increasing computational burden.

Third, the computing procedure of RRT* is highly sequential. As a tree growing-based approach, RRT*, by its nature, exhibits serial processing pattern and strong data dependency between different computing stages. More specifically, every time after a new sampling in the configuration space occurs, the corresponding collision checks and neighbor searches must be finished first before the planner begins another round of sampling. Such inherent *inter-sampling* data dependency, consequently, hinders the potential sampling-level parallelism of the entire RRT* processing, thereby limiting the performance improvement brought by hardware acceleration.

To address these challenges and enable real-time highly efficient motion planning in different dimensional settings, this paper proposes MOPED, an algorithm and hardware co-design towards efficient motion planning engine with flexible dimension support. *At the algorithm level*, we first develop a two-stage processing scheme to reduce the frequency and unit cost of collision check, alleviating the corresponding computational burden. We then propose to fully leverage the spatial information and the unique property of RRT* process to perform only necessary and approximated neighbor search, significantly reducing the operational cost without compromising path quality. A low-cost ($O(1)$) tree insertion scheme is also developed to enable low-overhead and more balanced tree structure. *At the hardware level*, we first build the architectural engine to support the low-cost computing scheme. We then develop a correctness-ensured speculative processing scheme to enable sampling-level parallelism. We also propose to explore multi-level caching opportunities to reduce data movement and resolve resource conflict.

We synthesize an MOPED design example using CMOS 28nm technology. With 1000MHz operating frequency, the 0.62mm^2-area MOPED prototype consumes 137.5mW power. Compared with baseline motion planning processors, MOPED achieves significant improvement on throughput, energy efficiency and area efficiency. .

II. Background

A. Motion Planning

Sampling-based Motion Planning. Motion planning problem can be solved from different perspectives, such as graph searching [24], [30], [50], [51], [68], tree exploring [22], [23], [41], [44], [46], [64], [87], learning from data [11], [63], [82], [84], [85], optimal control [22], [44] and factor graph [16], [35], [57]. Among them, sampling-based motion planners, e.g., RRT* [44] and its variants, are popular and effective approaches that can solve the planning problems defined in the configuration space with various sizes of dimension. By randomly sampling the configuration space to build a space-filling tree, sampling-based planners aim to find the desired collision-free paths via grows the exploration tree towards connecting the start and goal configurations. To date, because 1) they work well when the dimensions of configuration space scale, i.e., avoiding "curse of dimensionality" problem that search-based planners (e.g., A*) suffer; and 2) they provide probabilistic completeness guarantees in an asymptotically optimal way, sampling-based planners have been widely in many real-world applications.

Collision Check and Bounding Box. A very important kernel operation in the planning process is collision check, which aims to examine that, given the planned movement, whether there exist potential intersections between the robot and obstacles. More specifically, consider in practice both the robot and obstacle can have complex shapes and varying orientations, the geometries of the robotic body and environmental obstacles are typically represented via using bounding boxes. As shown in Fig. 1, using such representation can transform the collision check between two potentially irregular shapes to between two regular rectangles, simplifying the examination process. To date the most commonly used bounding method is *Oriented Bounding Boxes (OBBs)* [18], which encodes both the shape and orientation information of the object in the workspace, bringing tight-fitting and more accurate representation of the geometries of the objects [72]. Such benefit leads to significant reduction in the potential false-positive collision detection, improving the path quality, i.e., shorter path cost.

B. Rapidly-Exploring Random Trees (RRT*)

RRT* is the seminal and fundamental sampling-based motion planning approach, serving as the basis for many of its variants [1], [7], [10], [13], [22], [39], [42], [47], [73], [83]. The key idea of RRT* is to expand the exploration tree and refine the growth of the tree in an alternating way. More specifically, as illustrated in Fig. 2, at the *Tree Extension* stage the RRT* planner first randomly samples the configuration space, and then finds the nearest neighbor $x_{nearest}$ of the sampled point x_{rand} (i.e., a robot state in the configuration space) among all the nodes in the current exploration tree. After that the RRT* performs steering operation to decide a point x_{new} located at the line connecting $x_{nearest}$ and x_{rand}. Here the steering operation represents the impact of kinematic/dynamic constraints, e.g., the maximum movement the robotic may execute each time, and hence x_{new} , instead of x_{rand}, is the potential next configuration state the robot may have. To verify the movement from $x_{nearest}$ to x_{new} is collision free, a collision check for such potential movement is performed. If the check is clear, x_{new} will be added to the exploration tree as a new node, and its nearest neighbors in the tree will also be searched. Then, at the *Tree Refinement* stage RRT* aims to find the best edge involved with x_{new} to minimize the path cost. To that end, it calculates all the path costs incurred by adding different x_{new}-involved edges to the exploration tree, and identifies the one candidate which is collision free with the lowest cost. After that, both the exploration tree and path cost are updated and the next round of sampling begins. As proved in [44], with sufficient iterations the exploration tree will connect the start and goal states with low path cost, exhibiting *probabilistic completeness*.

C. Performance Challenge of RRT*

The above processing mechanism of RRT*, though enabling the capability of solving motion planning problem with varying dimensional sizes, causes the entire planning procedure being very computation intensive and inefficient. More specifically, we argue that the following three challenges hinder the computing efficiency of RRT*. First, it requires extensive costly collision checks. This is because every time when RRT* finds a potential movement, the corresponding collision checks between the robot and all obstacles, *during the entire movement course*, have to be clear to verify the legality. Even worse, such exhaustive collision check process is required in both Tree Extension and Refinement stages of RRT*, further increasing computational burden. In addition, the unit cost for each single collision check query is high. According to Separating Axis Theorem (SAT) [25], the collision check between two OBB-represented objectives requires to verify the existence of a separating axis among 15 potential axes (for 3D) that are derived from the geometric information

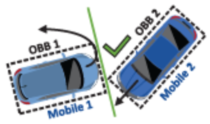

Fig. 1: Oriented bounding box (OBB) for check collision.

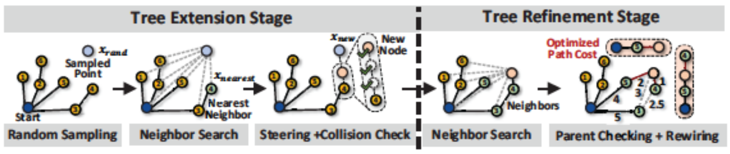

Fig. 2: Processing scheme of RRT*.

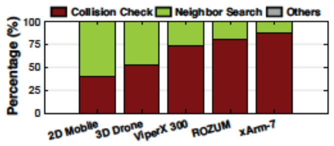

Fig. 3: Breakdown of computational costs for RRT*.

of the two OBBs; and the verification process for each axis consists of calculating dot products. Consequently, frequent query and pricey unit cost jointly make collision check become a very time-consuming operation (see Fig. 3).

Second, even with few obstacles that can ease the difficulty of collision check, RRT* is still computation intensive because of the costly neighbor search, an operation that suffers rapidly increasing complexity when the search scope or dimensionality scales. Unfortunately, for RRT* planning both of these two conditions occur. To be specific, because the neighborhood search is performed on all the nodes in the exploration tree, as the tree continues to grow, the search scope increases monotonically. Consider in RRT* the sampling operation can be repeated up to 500,000 times [44], the search cost in the later growing stage will become very significant. Meanwhile, because all the nodes are defined in the configuration space that can be of high dimension, the corresponding cost for distance calculation also linearly increases as the dimension scales – a typical working scenario for RRT*. Even worse, because neighbor search process is required twice, the incurred computational complexity has to be further doubled.

Third, RRT* exhibits inherent sequential processing pattern. As illustrated in Fig. 2, the exploration tree gradually grows as the sampling continues, and the collision check and neighbor search in next round of sampling cannot be activated until all the operations in the current round finishes. Essentially, such sampling-level serialization is the natural outcome of the primary mission of motion planner – finding a sequence of movement without collision. To achieve this goal, the decision on each movement must be made in a sequential manner, thereby hindering the potential performance improvement brought by applying parallel processing.

III. MOPED: Algorithm Design

To address the above analyzed challenges, we propose MOPED, an algorithm and hardware co-design to enable efficient motion planning with varying dimensional sizes. In this section we describe the algorithmic efforts first.

A. Two-Stage Processing for Low-Cost Collision Check

Recall that as profiled in Fig. 3, collision check, in most scenarios, contributes the largest portion of the computational costs consumed by RRT*. Therefore, the first step of our algorithmic optimization is to reduce the complexity of this bottleneck operation. To this end, we develop a two-stage processing mechanism to enable low-cost collision check, reducing both the frequency and unit cost of collision check query. As illustrated in Fig. 4, our key idea is to first use the pre-known hierarchical spatial information to avoid many unnecessary collision check and only perform low-cost coarse-grained collision check if needed; and then at the second stage more accurate collision check is performed in a fine-grained way to optimize path quality. Next, we describe this two-stage processing scheme in detail.

First-Stage Processing. The main purpose of first-stage processing is to filter the unnecessary collision check. Here the existence of "unnecessary collision check" is based on the observation that, because of robot's location and motion constraints (e.g., speed and scope), the potential collision between a robot and obstacle typically only occurs at certain region of workspace per each planned movement. For instance, when a robot is located in the north area of workspace, its collision with the obstacles located in the south region or out of its single movement range, obviously, is very unlikely. Therefore, many collision checks between the robot and obstacles can be potentially avoided, saving computational costs.

Based on this observation, a question then naturally arises: *How can we determine which collision checks are unnecessary?* To answer this question, we propose to use *R-tree* [27], a hierarchical tree data structure, to store the spatial information of obstacles. A key feature of R-tree is that it can group the nearby objects and represent the group at the lower level using minimum bounding box at the higher level (see Fig. 4). In other words, the spatial hierarchical information is naturally preserved in the R-tree data structure, thereby significantly facilitating identifying unnecessary collision check. More specifically, as illustrated in Fig. 4, when the collision between robot and node D in R-tree is free, it is evident that the robot cannot collide with any obstacles in the rectangle region that node D represents. Therefore, the corresponding collision checks between the robot and all the obstacles in that region (child nodes of node D), are unnecessary and can be skipped, improving processing efficiency.[1] In general, the R-tree search process starts from the root node. Then each time the possible intersections, which are between AABB-format bounding boxes of non-leaf nodes at level-i and OBB-format bounding box of robot, are examined via using Separating Axis Theorem (SAT). If there is no intersection for one level-i node, meaning collision free, its child nodes at level-(i+1) will not be further visited; otherwise the exploration continues for the corresponding child nodes. This search process continues till

[1]Building R-tree is based on bulk-loading method, e.g., sort-tile-recursive (STR) algorithm [48]. Here tree construction for collision check is an offline process before planning activities occur, not affecting runtime performance; while constructing R-tree for neighbor search is online and consumes costs. See Section III-C for details.

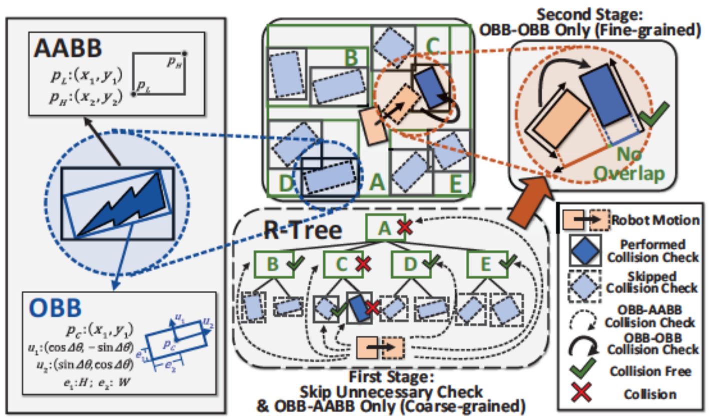

Fig. 4: Two-stage processing scheme. R-tree-based first-stage processing skips unnecessary collision checks, and all first-stage checks are OBB-AABB. Only second-stage processing performs OBB-OBB check.

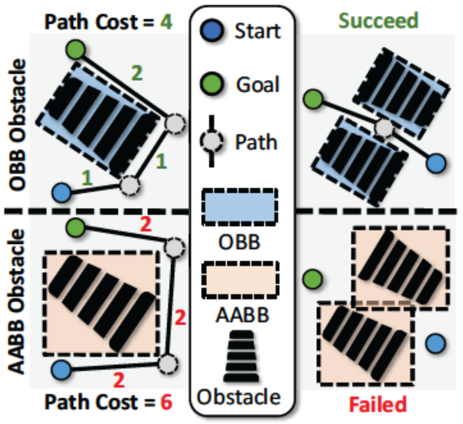

Fig. 5: Using OBB obstacle representation can bring lower path cost and higher success rate.

all the leaf nodes have been either skipped or visited.

In addition to reducing the frequency of collision check, our proposed first-stage processing also lowers the unit cost per query. More specifically, because R-tree data structure requires that the bounding method for its node must be Axis-Aligned Bounding Boxes (AABB), the collision check between the tree node and robot, when needed, is in the format of AABB-OBB (obstacle-robot), which is much more computational efficient than OBB-OBB type (details referred to Fig. 11). Consider all the nodes in R-tree are AABB-bounded, the unit cost per collision check occurred at the first stage is very low, further reducing the computational cost.

Second-Stage Processing. Compared with OBB, AABB bounding method suffers loss-fitting and less accurate representation of the geometry of object. Therefore, when AABB-OBB collision check is clear, it indeed means no collision; but if the check is not clear, it does not necessarily mean the collision occurs. Though the robot can still aggressively mark this condition as collision and adjust the planned movement, such *false-positive problem* will affect the path quality, e.g., causing higher path cost, or even fail the entire planning task in the small/narrow passage scenario. Notice that *path cost* is a very important performance metric for motion planning. In general, higher path cost means the robot has to consume much more energy and time to move and act. Because the actuation part of robot typically needs much higher power than the computing part, e.g., 86% (propellers) vs 5% (processors) of the total allocated power budget for small-size UAVs [76], reducing path cost, even in a small amount, can potentially bring huge energy saving for the overall robot system.

To address this challenge and achieve high path quality, we propose to perform fine-grained collision check as the second-stage processing. As shown in Fig. 4, for the identified collision between AABB-format obstacle (leaf node of R-tree) and OBB-format robot, we will then represent obstacle using more accurate OBB format to further perform OBB-OBB collision check. In general, OBB bounding method provides more compact encapsulation of an object than AABB. This closing-fitting representation of obstacles minimizes overestimation in the spatial extent of obstacles, enabling more precise computation of potential paths. Therefore, compared with planning method using OBB-AABB checker, using OBB-OBB checker can 1) find paths with lower path costs; and 2) find actually collision-free paths that OBB-AABB marks as collision (false-positive) (see Fig. 5).

Notice that though OBB-OBB check is costly per query, the required amount has been significantly reduced because of the filtering effect brought first-stage processing. Therefore the overall computational costs incurred by second-stage processing is still very low. Fig. 6 (experimental setting refers to Section V) shows compares the required computation incurred by collision check before and after using the two-stage processing scheme. It is seen that our proposed approach enables more than 20 times saving, significantly alleviating the computational burden for this bottleneck operation.

B. Steering-Informed Neighbor Search

With the order-of-magnitude cost reduction for collision check, neighbor search now becomes the new performance bottleneck. Therefore our next step of algorithmic effort is to optimize neighbor search operation in motion planning. To that end, because the success of efficient neighbor search is based on the proper extraction and use of spatial information, which can be efficiently encoded by tree structure in an organized way, we propose a novel data structure, steering-informed minimal-bounding-rectangle tree (***SI-MBR-Tree***) for motion planning-oriented neighbor search.

Skipping Neighbor Search using SI-MBR-Tree More specifically, as illustrated in Fig. 7, SI-MBR-Tree serves to store the spatial hierarchical information of nodes (robot state) in the RRT* exploration tree (referred as ***EXP-tree***). Here each leaf node of SI-MBR-Tree corresponds to one high-dimensional node of EXP-tree, and each non-leaf node of SI-MBR-Tree represents a minimum bounding box that contains all of its child nodes. Because such hierarchical representation implicitly encodes the information of spatial distance, e.g.,

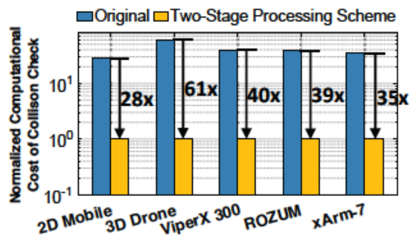

Fig. 6: Reduced computational cost of collision check brought by two-stage processing scheme.

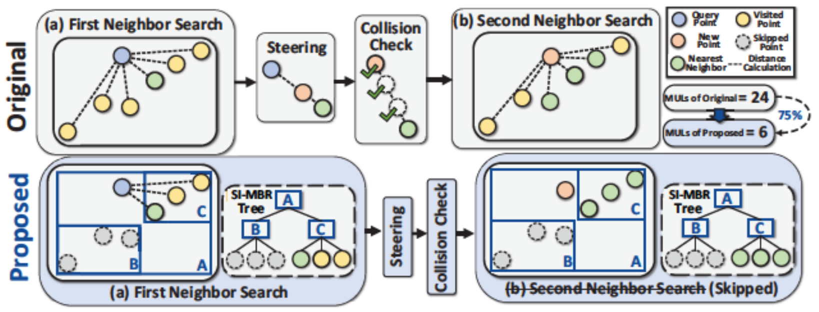

Fig. 7: Using SI-MBR-Tree and approximated neighborhood to reduce computational cost of neighbor search.

the leaf nodes of node C in Fig. 7 are closer to each other than to the leaf nodes of node B, using SI-MBR-Tree enables efficient neighbor search. For instance, when the distance of the query blue point to the non-leaf node B of SI-MBR-Tree (as a minimum bounding rectangle (MBR)), measured as the minimum distance (MINDIST) [14] between a point and rectangle (see Fig. 11 for detailed calculation), is greater than the distance to the current nearest neighbor, all the distance calculation between the query node and the leaf nodes of node B can be skipped. This is because according to its definition, MINDIST between the query and an MBR gives the shortest possible distance to any leaf node within that MBR, and hence MINDIST effectively serves as a representative minimum distance for all enclosed leaf nodes to the query point. Therefore, if the MINDIST to a non-leaf node's MBR is larger than the distance to a previously identified nearest neighbor, the entire sub-tree under that node can be skipped, as any leaf node within that sub-tree will inherently be more distant than the current nearest neighbor.

In general, the search process over SI-MBR-Tree starts at its root node and traverses the entire tree. When the descendants of the node being visited are leaf nodes, the Euclidean distances between these leaf nodes and the query point are calculated. If any of these distances is shorter than that of the current identified nearest neighbor, the nearest neighbor is updated accordingly. On the other hand, in the case that the node being visited does not have child leaf nodes, the MINDIST between the query point and the MBR of its child nodes is calculated. If the MINDIST for any child node is shorter than the distance to the current identified nearest neighbor, that node will be further explored as it might contain a nearer leaf node. These qualifying child nodes have their MINDISTs sorted, and the node with the shortest distance is visited first. Once this node and all its descendants have been fully explored, the child node with the next shortest MINDIST will be visited. However, if during this process the MINDIST for any unexplored child node exceeds the distance to the current nearest neighbor, that child and all its descendants will be skipped. The entire search process finishes once all leaf nodes have either been visited or skipped.

Steering-Informed Approximated Search. A key rationale of using SI-MBR-Tree-based neighbor search for motion planning is that it can fully leverage the unique characteristic of RRT* processing scheme, and hence open up the opportunities of ***using approximated but performance-preserved neighbor search to bring more computational saving***. To be specific, recall that two times of neighbor search are required in each round of sampling: the first is to identify the nearest node in EXP-tree ($x_{nearest}$) for the newly sampled point x_{rand}, and the second is to find the neighborhood of the node steered from $x_{nearest}$ towards x_{rand}. In such scenario, because 1) the newly determined point x_{new} (pink node in Fig. 7) is steered from the nearby $x_{nearest}$ (one green node in Fig. 7); and 2) the neighborhood of $x_{nearest}$ has been explicitly represented as non-leaf node C, we can simply approximate the nearby points of x_{new} as the neighbors of $x_{nearest}$, *further eliminating the need of the neighbor search*. Notice that this approximation is essentially enabled by joint use of unique characteristics of SI-MBR-Tree and RRT*. To be specific, the building procedure of SI-MBR-Tree (details refer to III-C) can inherently group the geometrically nearby leaf nodes within the same non-leaf node. Meanwhile, the steering operation in RRT* typically only brings short distance between $x_{nearest}$ and x_{new}, making it very likely that $x_{nearest}$ shares the same neighbors of x_{new}.

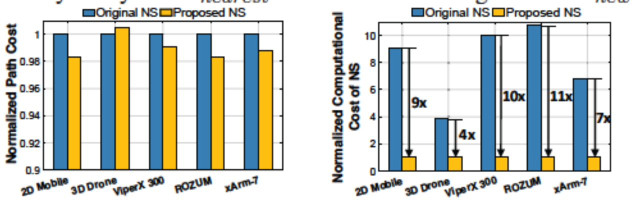

Fig. 8: (Left) Approximated neighbor search (NS) does not affect path quality. (Right) It reduces computational cost.

Two Remaining Concerns. Although this SI-MBR-Tree-based approximation can significantly reduce the computational cost of neighbor search in motion planning, it also brings two new concerns to be addressed. First, it is not clear that whether using the neighborhood of $x_{nearest}$, the approximation of the neighborhood of x_{new} , may affect the overall path cost or not, a critical performance metric to evaluate a motion planner. Ideally, such approximation should push the Pareto front instead of making trade-off between computational and path costs. Second, building SI-MBR-Tree in neighbor search is online and dynamic – all the nodes in the EXP-tree are incrementally obtained via sequential sampling and gradually inserted to SI-MBR-Tree. The computational implications of continually updating SI-MBR-Tree may cause non-negligible overhead incurred by tree growing.

Approximation Error Tolerance. In this subsection we analyze the potential impact on path cost incurred by the approximated neighbor search, and address the second concern in Section III-C. As shown in Fig. 8, (experimental setting refers to Section V) after applying the approximated neighbor search, the computational saving (at least 4 times) is not traded with significant increase in path cost. Instead, for several workloads the approximation-based solution even brings better path quality (lower path cost). We believe the main reason for such seemingly impossible "free lunch" phenomenon is that, since the Tree Refinement stage of RRT* will optimize the final path cost, more tolerance is granted for performing low-cost approximated neighbor search – any potential penalty incurred by the approximation can be further fixed via refining the tree later. In other words, the approximated neighbor search essentially leverages the inherent error-tolerance of RRT* with respect to path cost, reducing the computational cost without sacrificing path quality.

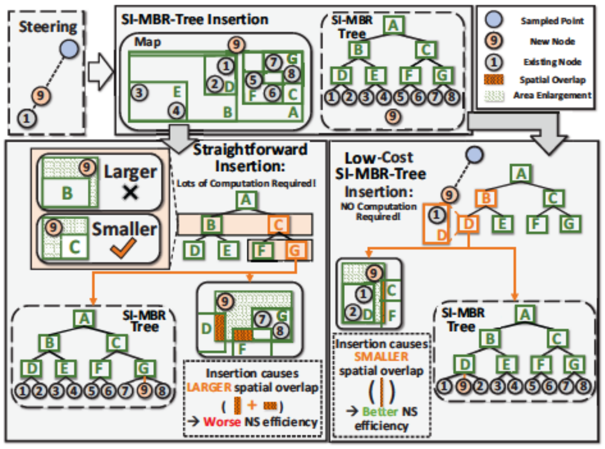

Fig. 9: The proposed low-cost SI-MBR-Tree insertion. SI means steering-informed. MBR means minimal-bounding-box.

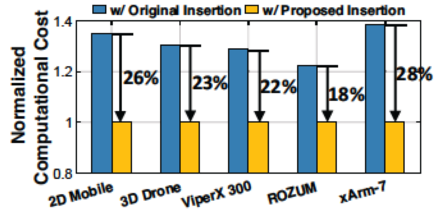

Fig. 10: Computational cost reduction brought by the proposed low-cost insertion method.

C. SI-MBR-Tree with Low-Cost Insertion

With the assurance that steering-informed approximated neighbor search does not affect the path cost, we next address the second issue – the computational implications of dynamically updating the SI-MBR-Tree. The primary ambition in developing this tree structure specific for neighbor search of RRT* lies in maximizing spatial efficiency, which encapsulates the structure's ability for logically and compactly organizing spatial data, facilitating rapid neighbor searches and accurate steering-informed approximations.

To ensure this spatial efficiency, the straightforward way is to hinge on a costly online construction process especially in high-dimensional scenarios where the motion planners always operate. More specifically, to insert the newly sampled point to the tree, the planner has to traverse the entire tree from the root to a leaf node with selecting the child node with *minimum area enlargement* at each level. As illustrated in Fig. 9, the area enlargement is defined by the increase of bounding box when inserting the sampled point to the bounding box that is represented by the tree node. In general, to improve the efficiency of neighbor search served by the tree, the area enlargement incurred by each node insertion should be minimized to reduce the spatial overlap between the sibling non-leaf nodes and maintain a balanced tree. This is because less area enlargement leads to more compact spatial organization of the data, bringing fewer number of overlapping regions and making it more likely that the nodes are well-distributed [54].

As described above, the straightforward tree insertion traverse the entire tree and check the minimum area enlargement at each level. Consider calculating the area enlargement is costly especially in high-dimensional setting, such level-by-level procedure, when repeats for each sampled point, brings high computational cost. In addition, this level-wise check strategy is only locally optimal, since sequentially minimizing the area enlargement for each level does not guarantee that the final choice at the leaf node exhibit the globally minimum area enlargement. Consequently, the tree constructed using this traversal check may still suffer unbalanced structure and excessive spatial overlap between the non-leaf nodes at the same level, hindering search efficiency.

SI-MBR-Tree with Low-Cost Insertion. To overcome these limitations, we propose the steering-informed minimal-bounding-box tree (***SI-MBR-Tree***) with low-cost ($O(1)$) insertion strategy that is specifically designed for neighbor search operation in RRT*. As illustrated in Fig. 9, our key idea is to directly set x_{new} as the sibling leaf node of $x_{nearest}$ in the tree, i.e., they share the same parent non-leaf node. This simple strategy is motivated by the observation that x_{new} is steered from $x_{nearest}$ with a pre-set step size. In other words, it is very likely that x_{new} and $x_{nearest}$ are very closed to each other, implying that when we add x_{new} as the sibling leaf node of $x_{nearest}$, the incurred minimum area enlargement should be substantially small, potentially leading to smaller spatial overlap and more balanced tree structure. Another attractive benefit brought by this ultra-simple direct insertion is the low operational cost. Compared to repeatedly calculating the level-wise minimum area enlargement during the entire course of tree traversal, computation-free simple insertion is evidently much more efficient. As shown in Fig. 10 (experimental setting refers to Section V), this spatial information-aware insertion approach brings additional more than 20% lower computational cost over the design using conventional tree insertion, resolving previous concern on insertion overhead.

Why SI-MBR-Tree-based Neighbor Search? Readers who are familiar with efficient K-Nearest Neighbor (KNN) search may question that why SI-MBR-Tree, instead of KD-tree and other Bounding Volume Hierarchy (BVH) tree that have been efficiently accelerated in both hardware [12], [19], [62], [79] and software implementations [32], [33], [54]–[56], [88], is adopted here. We believe SI-MBR-Tree is a more suitable solution in our target robotic scenarios because of three reasons. 1) Many existing efficient KNN implementations [12], [19], [32], [33], [62], [79], [88] are primarily designed for low-dimensional (2D/3D) neighbor search used in computer graphics, vision and fluid dynamics; while motion planning typically requires neighbor search operated in high-dimensional configuration space (e.g., 5-13 DoF). As indicated in [26], [39], the *curse of dimensionality* causes the low-dimension-oriented solutions (e.g., KD-Tree) need to visit substantially more branches in the high-dimension contexts, thereby substantially reducing acceleration efficiency. On the other hand, MOPED's SI-MBR-Tree is specifically designed for high-dimensional search, mitigating this challenge effectively. 2) Most of existing efficient KNN implementations, even for FLANN library [55], [56] that can work in high-dimensional space, are primarily designed to handle static datasets. However, the neighbor search required in RRT* is usually operated in the dataset that is dynamically updated – the planner samples configuration space continuously. As indicated in [21], [52], [55], incorporating these static data-oriented methods (e.g., KD-tree) into sequential data acquisition-based robotic applications requires to rebuild the entire tree from the scratch for multiple times, obviously causing substantial computational overhead. Instead, as analyzed in Section III-C, SI-MBR-Tree insertion enjoys very low cost as tree grows, bringing non-negligible overhead. 3) Fig. 7 shows that the computing procedure of steering operation-based RRT* requires two times of neighbor search after each round of sampling. Applying other efficient KNN methods cannot reduce such repeated requests because it is the algorithmic requirement of RRT*. On the other hand, as described before, because SI-MBR-Tree data structure naturally groups the geometrically nearby leaf nodes belonging to the same non-leaf node, we can leverage this unique property and the characteristic of steering operation in RRT* to perform steering-informed approximated search, eliminating the need of second neighbor search and saving computational costs.

IV. MOPED: Hardware Architecture

A. Overall Architecture

Based on the proposed algorithmic optimization, in this section we further develop and optimize the corresponding hardware accelerator. Fig. 11 shows the overall architecture of MOPED hardware, which consists of a *Tree Extension Module* and a *Tree Refinement Module*. Here Tree Extension Module is in charge of sampling configuration space, performing neighbor search, steering and collision check. To that end, a group of LFSRs is used to perform random sampling in the high-dimensional configuration space. The sampled point x_{rand}, together with the information of the nodes in the EXP-tree (stored in the EXP Node SRAM), are sent to the neighbor search component to identify the neighborhood. In the EXP Node SRAM, the spatial information of each node is stored using d values, where each value takes 16 bit and d is the number of DoF in the current task. As described in Section III-B, SI-MBR-Tree is used to simplify neighbor search operation, hence the information of this tree is stored in Bottom NS SRAM with cached top tree (see details in Section IV-C). The spatial information of each Minimum Bounding Rectangle (MBR) in the tree is represented by a set of 16-bit values, consisting of $2d$ components. The first d components denote the minimum coordinates, and the remaining d components signify the maximum coordinates. The results of neighbor search is then sent to S&R unit, which is designed for increasing sampling-level parallelism (see Section IV-B for details) with speculative and repaired neighbor search. After x_{new} is steered from $x_{nearest}$, together with OBB/AABB-format obstacle information stored in SRAM, it is then sent to the SAT-based collision checker to determine the clearance of the requested check. The spatial information of AABB and OBB- format obstacles are represented by sets of 16-bit values, with AABB and OBB consisting of 6/4 and 15/8 values for 3D/2D workspace, respectively. Here the first 3/2 values in both AABB and OBB represent the coordinates of the center point in the 3D/2D workspace, and the subsequent 3/2 values in both AABB and OBB denote the positive halfwidth extents of the box along each axis. In the case of OBB, the final 9/4 values express the rotation matrix that encapsulates the orientation information. When check is clear, SI-MBR-Tree Operator is activated to update SI-MBR-Tree stored in SRAM via using low-cost insertion method described in Section III-C.

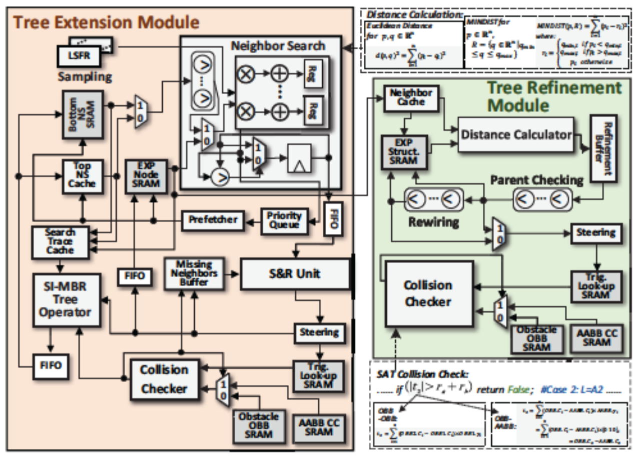

Fig. 11: Overall architecture of MOPED hardware.

For Tree Refinement Module, because its primary function is to rewire node connection of EXP-tree if needed, a EXP Struct SRAM storing structural information of EXP-tree (path cost and node connection) is budgeted. If the distance calculator finds there exists a shorter path cost and the structure of EXP-tree should be modified, another copy of collision checker is activated to determine whether the rewired connection of the EXP-tree, which corresponds to the planned movement, collides with the environmental obstacles or not.

B. Enabling Sampling-Level Parallelism

As shown in Fig. 11, MOPED hardware architecture puts an S&R unit between the processing of neighbor search component and collision checker. Here the main function of this special unit is to enable the sampling-level parallelism of the entire computing scheme. To be specific, as illustrated in Fig. 12, in conventional processing procedure, the neighbor search and collision check for the next round of sampling cannot be enabled until all the operations occurred in the current sampling finish. This is because the correctness of the neighbor search in the next sampling relies on the most recently updated SI-MBR-Tree, which can only become available after the online insertion operation of SI-MBR-Tree in the current sampling. Evidently, such strong data dependency prohibits exploring the potential sampling-level parallelism. Also, because neighbor search and collision check are processed in the different units, it causes low resource utilization.

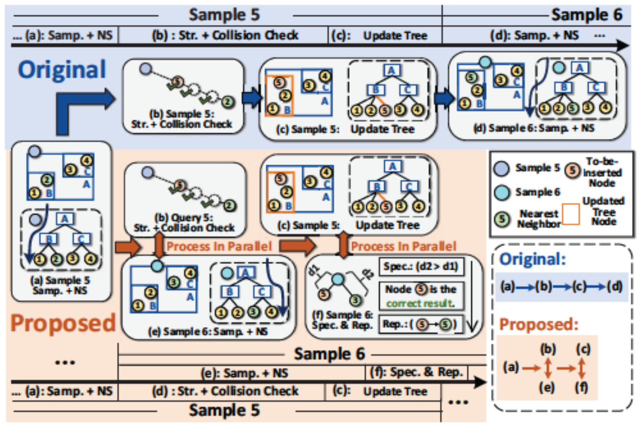

Fig. 12: Speculate-and-Repair. NS: neighbor search, Samp.: sampling, Str.: steering, Spec.: speculate, Rep.: repair.

To address this challenge, we develop a *"speculate-and-repair"* (S&R) strategy to enable sampling-level parallelism and improve utilization of computing units. As illustrated in Fig. 12, our key idea is that upon the completion of neighbor search in the current sampling, the Tree Extension Module immediately begins the next-round sampling and neighbor search operation in a speculative way. Once the collision check in the current sampling and neighbor search in the next sampling finish, a repair operation is performed to ensure that the correct result of the neighbor search in the next sampling can be identified. In general, the success of this repair effort is guaranteed by the fact that since the difference between the non-updated and updated SI-MBR-Tree is only one to-be-inserted node (e.g., node 5 in Fig. 12) that is steered from the nearest neighbor (node 2) identified in the current sampling (Sample 5), the correct nearest neighbor in the next sampling (Sample 6) must exist between the result (node 3) of the speculated neighbor search in the next sampling and this to-be-inserted node (node 5). Therefore, we can simply fix the potential incorrectness incurred by speculated neighbor search in Sample 6 via comparing the distances of node 3 and node 5 to the newly sampled point in Sample 6, thereby ensuring the correct output of neighbor search required in Sample 6. Notice that because the speculated neighbor search in the next sampling cannot find the to-be-inserted node (e.g., node 5) in the current sampling, we refer this type of node as "missing" neighbors and allocate a special buffer, the Missing Neighbors Buffer, to store them to facilitate the processing in S&R unit (see Fig. 11). More specifically at the hardware implementation level, the output from the neighbor search component, represented as $x_{nearest}$ and its associated distance to x_{rand}, is stored in a FIFO. Subsequently, when the S&R unit is ready and the Missing Neighbor Buffer is not empty, a comparison is made between the stored distance and the distances of the missing neighbors to x_{rand}. If any inaccuracy in the speculated neighbor search is detected, the nearest point, $x_{nearest}$, is promptly updated with the accurate value retrieved from the Missing Neighbors Buffer. This corrective action constitutes the 'repair' operation within the S&R unit.

Notice that using "speculate-and-repair" strategy essentially reduces the overall *planning latency* of MOPED. More specifically, as the sampling-based motion planner, MOPED calculates and outputs the collision-free path only after completing all the required sampling. This is because as shown in Fig. 2, the exploration tree dynamically extends and changes as sampling process progresses, and hence the planned path, which should be identified from the most recently updated exploration tree, will not be calculated until finishing the last round of sampling [2]. Therefore, the overall planning latency of MOPED is the end-to-end duration measured from first to last round of sampling, instead of the time consumed in each round. Consider 1) the neighbor search and collision check, which can be now largely overlapped by using the S&R strategy (see Fig. 12), are very costly since every neighbor search and collision check require the involvement of all the sampled nodes (up to thousands or more); and 2) the newly added speculate/repair operation, is instead very cheap because each time it only calculates the distances of very few missing neighbors (fewer than 10) to x_{rand}; such huge saving brought by overlapping old operations and minor overhead incurred by new operations, together, significantly reduces the end-to-end planning latency. For instance, with 5000 rounds of samplings in a 2D mobile workload, our proposed speculate-and-repair strategy brings about 2x reduction in overall planning latency. More evaluation on the speedup provided by this strategy can be referred to Section V-C.

Resource Availability & Overhead of FIFO/Missing Neighbors Buffer. As described in the above paragraphs, FIFO and Missing Neighbors Buffer are two key hardware components for realizing S&R strategy, making their resource availability and cost largely impacts the effectiveness of S&R

[2]Notice that here the number of rounds of sampling is the actual amount of sampling that the planner uses for generating the collision-free path. For instance, if the planner with 5000-sampling budget decides to early terminate and attempts to calculate the path after 100 sampling, the last round of sampling is the 100-th sampling.

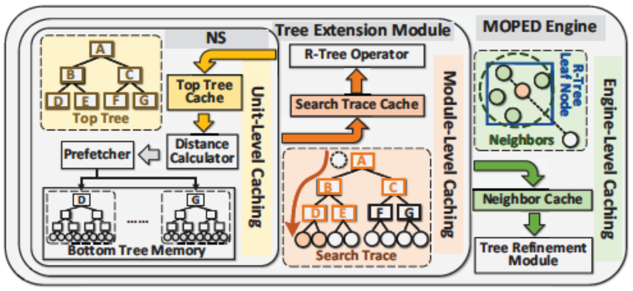

Fig. 13: Multi-level caching. NS denotes neighbor search.

strategy and the overall hardware overhead. A promising feature of the proposed S&R strategy is that it only consumes ultra-low-cost FIFO and Missing Neighbors Buffer. More specifically, the depth of FIFO is essentially the the maximum number of sampled points awaiting collision checks (collision checks may be slower than the neighbor searches, causing potential accumulation of sampled points awaiting collision checks). Fortunately, our experiments across different workloads show that such backlog phenomenon, if occurs, is very negligible, and using a 20-depth FIFO is already sufficient to avoid the unnecessary pipeline stalls. Meanwhile, the capacity of the Missing Neighbor Buffer is determined by the maximum number of collision checks that can be ascertained as collision-free within the duration of one neighbor search operation. Our experimental finding shows that this number is very small, and budgeting a buffer capable of storing up to 5 "missing neighbors" is already enough for different planning workloads. Such small FIFO and Missing Neighbors Buffer only consume 0.75KB extra storage, a ultra-low overhead meaning that the resource availability of these two hardware modules can be very easily satisfied.

Overall, after using this speculate-and-repair solution, the operations in the consecutive sampling can now be paralleled. Essentially, the original strong data dependency is mitigated and the hardware can now be properly pipelined, reducing the overall planning latency and increasing resource utilization. Notice that such benefits are enabled without sacrificing the correctness of functional validity – the missing neighbors buffer-powered repairing operation guarantees that the correct nearest neighbors can always be identified after performing speculated neighbor search. Therefore, the planner with and without using S&R strategy are functionally equivalent.

C. Hierarchical Multi-Level Caching

In addition to improving parallelism, we also develop a hierarchical multi-level caching strategy to further optimize the performance of MOPED hardware. To be specific, three levels of caching opportunities are explored in the design.

Unit-Level Caching. First, we propose to cache the data used in the neighbor search component via leveraging the *temporal locality* of SI-MBR-Tree. As shown in Fig. 13, each time when this neighbor search component searches the neighborhood, the top part of the tree is always accessed more frequently because the SI-MBR-Tree-based search process is performed from top towards bottom. Therefore, caching the corresponding data, e.g., information of bounding box, brings reduction in memory access and more energy-efficient search.

Module-Level Caching. In addition, we also cache the trace of neighbor search to address the potential resource conflict issue in the Tree Extension Module. More specifically, every time after the desired nearest neighbor leaf node of SI-MBR-Tree is identified, the search history for finding this node, i.e., the spatial information of the bounding boxes represented by the visited non-leaf nodes, will be saved. Here our rationale is that, it is very likely that the sizes of these corresponding bounding boxes will be adjusted because of the insertion of new leaf node; and meanwhile the information of the non-leaf nodes of SI-MBR-Tree is also heavily used in the speculative neighbor search at that moment. Consequently, caching the search trace is very desired to avoid the access conflict for SI-MBR-Tree memory.

Engine-Level Caching. We also observe that caching the identified neighborhood is required to facilitate data sharing between the two computing modules of MOPED accelerator. More specifically, after the neighbor search component in Tree Extension Module identifies the neighboring nodes of SI-MBR-Tree, it is very necessary to save the information of these neighbors to a cache, preparing for the future use in the rewiring operation performed by Tree Refinement Module. Otherwise, because the memory (Bottom NS SRAM and Top NS Cache) storing those information are always extensively accessed due to the frequent neighbor search query, severe memory access conflict may occur.

V. Evaluation

Robot Models. In this section, we evaluate the algorithmic and hardware performance of our proposed co-design solution on five robot models with different DOFs and bounding box representations. More specifically, they are:

- 2D Mobile: A 3-DOF robot with two translational degrees of freedom (x, y) and one rotational degree of freedom, bounded by one 2D OBB.
- 3D Drone: A 6-DOF robot featuring three translational degrees of freedom (x, y, z) and three rotational degrees of freedom (yaw, pitch, roll) [61], bounded by one 3D OBB.
- ViperX 300 Robot Arm [66]: A 5-DOF robot arm, described by five distinct joint angles that dictate the arm's configuration in the workspace. This robot arm is bounded by three 3D OBBs.
- ROZUM Robot Arm [65]: A 6-DOF robotic arm, specified by six unique joint angles that define the arm's configuration in the workspace. This robot arm is bounded by four 3D OBBs.
- xArm-7 Robot Arm [71]: A 7-DOF robotic arm, defined by seven independent joint angles that determine the arm's configuration in the workspace. This robot arm is bounded by seven 3D OBBs.

Environmental Settings. The performance is evaluated in a simulated 3D workspace of size $300 \times 300 \times 300$ (300×300 for 2D Mobile) with varying obstacle counts: 8, 16, 32, and 48. For each environment configuration, we generate

50 different planning tasks by randomly placing obstacles of various shapes (3D size limited to $30 \times 30 \times 50$, 2D size limited to 30×30) in random locations with random orientations. In addition, the start and goal configurations for each planning task are also randomly generated. This randomness enables us to robustly assess the effectiveness of our approach across a diverse range of scenarios. Moreover, the environment size and obstacle densities are similar with those utilized in other motion planning works [4], [49], [60], ensuring our performance evaluation are appropriately benchmarked. Because in real-world scenarios the obstacle information received by back-end planner is OBB-format data generated by front-end perception module, random generated obstacles in our simulation environment are also in OBB format. The corresponding spatial information, including the coordinates of central points, halfwidth extents of boxes along the axis and the orientation-related rotation matrix, are stored in the obstacle OBB SRAM (see Fig. 11). More details of obstacle encoding of MOPED can be referred to Section IV-A. Note that because MOPED first performs an AABB-OBB collision check at the first stage, the spatial information of AABB-format obstacles is calculated from OBB format and stored in another AABB SRAM (see Fig. 11).

A. Algorithmic Performance

Fig. 14 summarizes the algorithmic performance of MOPED across various robot models with different environmental settings with 5,000 sampling attempts. It is seen that MOPED significantly reduces the computational costs without compromising the path quality. Notably, for higher-dimensional applications, the reduction in computational complexity is more pronounced. Furthermore, as the environment becomes more complicated (characterized by the increasing number of obstacles), MOPED demonstrates its enhanced capability for complexity reduction. These findings highlight the efficacy and adaptability of MOPED in solving high-dimensional motion planning problems across wide range of robot models and environmental complexities.

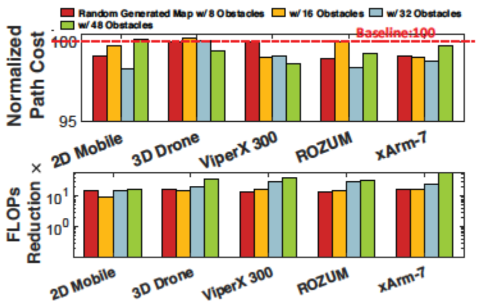

Fig. 14: Algorithmic performance of MOPED.

B. Hardware Performance

Experimental Setting. We first conduct a high-level functional simulator for the behavior modeling of MOPED. We then utilize Verilog to develop a RTL model which is synthesized with the Synopsys Design Compiler and a CMOS 28nm library. Synthesis results provide the datapath area and power consumption metrics, and CACTI is used for obtaining the memory performance.

Baseline: We compare MOPED with three baselines:

- CPU: A C++ version of original RRT* algorithm implemented based on RTRBench [5], a real-time robotics benchmark. Notice that we modify its planning environment and robot modeling method to better fit for real-world motion planning tasks with high DoFs. The computing platform is AMD EPYC 7601 CPU.
- RRT* ASIC: A hardware implementation of original RRT* algorithm, using the similar architecture (overlapping tree extension and refinement stages) used in [78] and with the same computing resource and very similar memory resources as MOPED has.
- RRT* ASIC with CODAcc: The above hardware implementation of RRT* with collision check using CODAcc [4], a recent collision check accelerator that performs OBB collision checks using occupancy grid method. We adopt a resolution of one unit length per cell and integrate four CODAcc accelerators.[3]

Design Configuration. We implement a design example of MOPED hardware architecture equipped with 168 16-bit multipliers and accumulators (MACs) and 198 KB of on-chip SRAM. With 28nm CMOS technology, this motion planning accelerator occupies 0.62 mm^2 silicon area and consumes 137.5 mW power under 1000 MHz operating frequency.

Hardware Performance Comparison. Fig. 15 illustrates the performance improvement of MOPED as compared to the baselines across various robot models and working environments. The latency of MOPED in executing these tasks are 0.35-0.96 milliseconds. Compared to CPU-based solutions, MOPED achieves $1066.2\times$ – $6149\times$ speedup and $453.6\times$ – $10744.6\times$ higher energy efficiency, respectively. Compared with RRT* ASIC design, MOPED demonstrates $2.3\times$ – $41.1\times$ speedup, $2.1\times$ – $38.3\times$ times area efficiency increase, and $2.1\times$ – $38.2\times$ improved energy efficiency, respectively. Compared with RRT* ASIC solution equipped with customized collision check hardware CODAcc, MOPED achieves $2\times$ – $9.2\times$ speedup, $1.7\times$ – $7.9\times$ better area efficiency, and $2\times$ – $9.3\times$ higher energy efficiency, respectively.

C. Analysis & Discussion

Breakdown in Computational Saving. Because the significant reduction in computational costs come from several efforts, Fig. 16 (Top) analyzes the impact of different approaches. Here V1 means only Two-Stage Processing Scheme (TSPS) is applied, bringing 33.9% – 77.7% reduction in computation across various robot models. V2 represents the design further applying SI-MBR-Tree-based Neighbor Search (STNS) on V1. And this effort leads to an additional 48.2% – 80.1% computational saving. V3 integrates the Steering-Informed Approximated Search (SIAS) to V2, bringing further

[3]Notice that since the occupancy grid method is built on the discretization of the environment, its memory consumption is huge. In our experimental setting, more than 3.2MB on-chip SRAM is needed to store the occupancy grid information of the $300 \times 300 \times 300$ 3D environment with a resolution of one unit length per cell. Following the setting in [4], we assume that an external CPU sends the occupancy grid information to the accelerator; and area, power and communication costs incurred by using CPU are not included in the performance report of this baseline. MOPED does not need CPU.

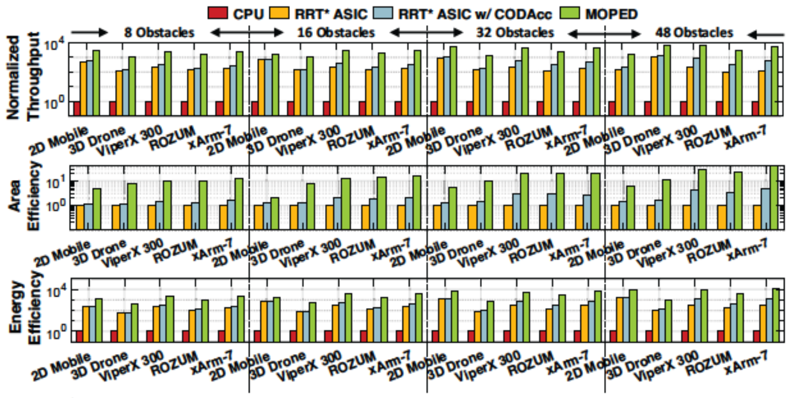
Fig. 15: Hardware performance of motion planning hardware across different robot models in different environments.

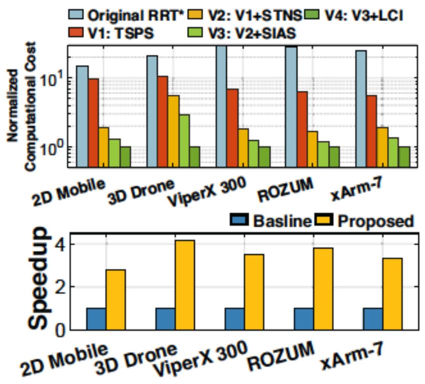
Fig. 16: (Top) Source of computational saving in MOPED. (Bottom) Software-only speed-up.

28.3% – 47% less computation as compared to V2. V4 is built on the top of V3 via featuring Low-Cost Insertion (LCI), achieving additional 14.6% – 66% computational saving.

Software-only Speedup. To evaluate the acceleration performance of our proposed algorithm without any hardware support, we benchmark it against the C++ software implementation described in Section V-B. As shown in Fig. 16 (bottom), our software-only solution brings 2.77 × – 4.14 × speedup over the baseline design. This result shows that 1) MOPED algorithm indeed brings significant runtime acceleration; and 2) to fully unleash the algorithmic benefits, i.e., 15× and more computational saving as shown in Fig. 16 (top), our proposed customized hardware support is very necessary.

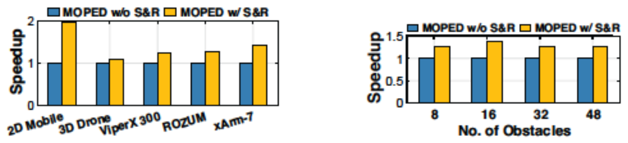
Fig. 17: Speedup brought by S&R across (Left) different robot models (Right) different environments (ViperX 300 robot).

Speedup & Scalability of "Speculate-and-Repair". Fig. 17 (left) and (right) illustrate the speedup after using S&R strategy across different robot models and different environmental complexities, respectively. It is seen that the proposed strategy brings consistently speedup for different planning workloads. Such stable acceleration performance with different robot models (from 2 to 7 DoFs) and different environmental settings (from 8 to 48 obstacles), also demonstrates the good scalability of S&R strategy when problem complexity increases. Notice that because the S&R strategy aims to overlap neighbor search and collision check operations, the varying performance improvement on different workloads are mainly determined by the condition that whether the costs of these two operations are more close or not.

Bounding Box for Collision Check: OBB vs AABB. Fig. 18 (left) compares the path cost using OBB and AABB bounding boxes to represent obstacles in the collision check. It is seen that OBB-based solution enjoys 20%-50% lower path costs. Such significant improvement in path quality brings huge time/energy saving for the overall robotic system, since the overall planned path has been largely shortened. In addition, we also evaluate the acceleration performance of MOPED only using AABB bounding box-based collision checker. As shown in Fig. 18 (right), compared with baseline RRT* ASIC using the same AABB checker, MOPED still brings 5.6× - 7.6× speedup.

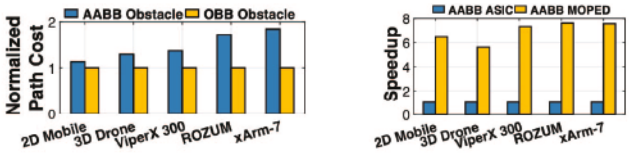
Fig. 18: (Left) Path cost using AABB and OBB-format obstacle. (Right) Speedup over baseline RRT* using AABB.

Speedup at Different Sampling Stages. The number of sampling points is an important factor that can affect the performance of motion planner, since it directly determines the complexity of neighbor search. Fig. 19 (left) shows the speedup of MOPED at varying sampling stages. It is seen that MOPED achieves steadily increasing speedup as more points in the configuration space are sampled. This is because as the complexity of neighbor search continues to scale due to the increasing sampled points added to the EXP-tree, the low-cost search featured by MOPED becomes more significant, bringing higher speedup as compared to early sampling stage. Notably, such property is particularly attractive when solving challenging ultra-high-dimensional planning problems that may require a large number of samplings.

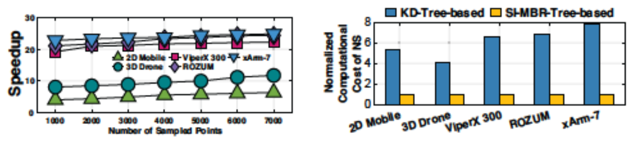
Fig. 19: (Left) Speedup of MOPED at different sampling stages. (Right) Computational costs of SI-MBR-Tree and KD-Tree -based neighbor search in RRT*.

SI-MBR-Tree vs KD-Tree for Neighbor Search in RRT*. Fig. 19 (Right) compares the computational costs of SI-MBR-Tree and KD-Tree-based neighbor search in RRT*. It is seen that using SI-MBR-Tree brings 4.12× – 7.76× computational saving over KD-Tree-based solution. As outlined in Section

III-C, this superior performance can be attributed to the inherent advantages of SI-MBR-Tree for neighbor search in RRT*, such as suitability for high-dimensional spaces and dynamic datasets and the elimination of repeated search requests.

VI. Related Work

Motion Planning Hardware. Despite the importance of motion planning, to date only few hardware accelerators are reported [4], [36]–[38], [49], [58]–[60], [78], [80], [81]. Among them, [60] (MICRO'16) proposes to use exhaustive offline collision check for each possible movement, and stores all the pre-computed results on chip for rapid reference. Such trading-storage-for-speed strategy is tightly integrated with the environment, and hence it needs hours of offline reset if obstacles change. Also, if the environment or resolution scales up, the required storing cost becomes very huge, hindering the scalability. Recently, [4] (ISCA'22) proposes a CPU-aided hardware accelerator for search-based planning. Due to the inherent limitation of search algorithm, its suitable application is 2D/3D instead of high-dimensional configuration space. Also, using CPU to store the discretized environmental information brings non-negligible overhead.

RRT* and its Variants. To date, many RRT/RRT* variants have been proposed to further improve the planning performance, e.g., biased sampling [22], [83], increasing parallelism [7], [13], [39], [45], [47] and adapting for dynamic environments [1], [8], [20]. Different from these existing works, MOPED aims to solve another general bottleneck for RRT*/RRT and variants: how to *reduce* the operational frequency of collision check and neighbor search *per sampling*. Because 1) this problem is very fundamental and universal and 2) RRT* serves as the kernel in its variants, the solution proposed in MOPED is not only beneficial to RRT*, but can also be directly applied in all types of RRT*/RRT variants to reduce computational cost.

Parallelizing RRT*. Parallelizing the RRT* procedure can be explored at different levels. RRT-Connect [45] aims to enable *exploration tree-level* parallelism via first growing trees from start and end points simultaneously, and then connect them. Also, [4], [7] propose to increase *collision check-level* parallelism via leveraging the nature that multiple collision checks within the same round of sampling can be performed simultaneously. MOPED, together with [39], [47], study the efficient approaches to improve *sampling-level parallelism*. A key difference between MOPED and [39], [47] is that MOPED focuses on parallelizing consecutive samplings performed by each thread; while the goal of [39], [47] is to parallelize multiple sampling performed by different threads (targeting for multi-core CPU/GPU). In other words, MOPED explores *temporal parallelism* and [39], [47] studies *spatial parallelism*. Another fundamental difference from prior parallel RRT* works is that MOPED also reduces the frequency and cost of collision check and neighbor search; while only increasing parallelism will not change the overall cost. Evidently, MOPED is complementary with these prior RRT* parallelization works and the joint use of them can bring further acceleration.

Learning/Factor Graph-based Motion Planning. DNN-based motion planners receive increasing attention in the recent literature [11], [40], [63], [82], [84], [86]. Despite the powerful capability of DNNs, learning-based planners suffer from high model costs and lack of theoretically guaranteed completeness that RRT* enjoys, hindering their practical deployment in real-world applications. In addition, due to its versatility in modeling diverse optimization challenges in robotics [15], [16], [35], [43], [57], factor graph, as a probabilistic graphical model for representing state estimation problems, has also been proposed to solve a series of robotic tasks (e.g., motion planning, localization, control), further motivating the research on factor graph-oriented hardware accelerators [28], [29], [69]. A potential challenge for applying factor graph in motion planning is that this trajectory optimization method might get trapped in the local optima and sometimes may not able to provide a feasible solution under tight planning constraints [35] and hence how to build reliable factor graph-based motion planners for real-world scenarios is an open research question and needs more exploration.

Space Subdivision Algorithms for Collision Check. Space subdivision algorithm [74], e.g., R-tree/AABB-tree, KD-tree and Octree are popularly used geometric data structures, and some of them have been adopted for checking collision in computer graphics [70], [75]. Motion planning has unique requirements for tree-based collision check, especially on path quality and hardware cost. Using computer graphics-oriented R-tree/AABB-tree suffers potential false positive problem, severely affecting path quality in motion planning. Octree spatially divides 3D space into small voxels. Because representation precision is an important factor determining the performance of robotic task using Octree, high resolution is typically required, bringing very high memory consumption. For instance, even for Octree-based works aiming to minimize memory usage [34], [77], the storage demand for environment modeling can still be up to hundreds of megabytes (e.g., 130MB), posing challenges for applying Octree-based solution in resource-constrained motion planning applications. KD-tree is constructed by recursively splitting the environment into two halves using a plane aligned with one of the domain axes. In practice, there are always obstacles that straddle the dividing plane during the construction process, and then the boundaries of one partition has to be modified to encompass the obstacles, bringing overlaps between the partitions [9]. This inherent and uncontrolled overlap causes many redundant collision check queries, makes entire planning process inefficient. Also, KD tree is less efficient in managing dynamic data, making it not suitable for collision check in the dynamic environments with moving obstacles.

VII. Conclusion

This paper develops MOPED, an algorithm and hardware co-design for efficient motion planning engine. Compared with the existing motion planner solutions, MOPED achieves very significant performance improvement.

Acknowledgements

This work is partially supported by National Science Foundation under Grant CCF-2239945.

References

[1] O. Adiyatov and H. A. Varol, "A novel rrt-based algorithm for motion planning in dynamic environments," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, 2017, pp. 1416–1421.

[2] W. G. Aguilar, S. Morales, H. Ruiz, and V. Abad, "Rrt* gl based optimal path planning for real-time navigation of uavs," in *Advances in Computational Intelligence: 14th International Work-Conference on Artificial Neural Networks, IWANN 2017, Cadiz, Spain, June 14-16, 2017, Proceedings, Part II 14*. Springer, 2017, pp. 585–595.

[3] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2421–2428.

[4] M. Bakhshalipour, S. B. Ehsani, M. Qadri, D. Guri, M. Likhachev, and P. B. Gibbons, "Racod: algorithm/hardware co-design for mobile robot path planning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 597–609.

[5] M. Bakhshalipour, M. Likhachev, and P. B. Gibbons, "Rtrbench: A benchmark suite for real-time robotics," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2022, pp. 175–186.

[6] A. Bauer and Z. Popović, "Rrt-based game level analysis, visualization, and visual refinement," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 8, no. 1, 2012, pp. 8–13.

[7] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the rrt and the rrt," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3513–3518.

[8] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ international conference on intelligent robots and systems*, vol. 3. IEEE, 2002, pp. 2383–2388.

[9] J. R. Bruce, "Real-time motion planning and safe navigation in dynamic multi-robot environments," *Ph. D. dissertation*, 2006.

[10] M. Čáp, P. Novák, J. Vokřínek, and M. Pěchouček, "Multi-agent rrt*: Sampling-based cooperative pathfinding," *arXiv preprint arXiv:1302.2828*, 2013.

[11] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song, "Learning to plan in high dimensions via neural exploration-exploitation trees," in *International Conference on Learning Representations*, 2019.

[12] F. Chen, R. Ying, J. Xue, F. Wen, and P. Liu, "Parallelnn: A parallel octree-based nearest neighbor search accelerator for 3d point clouds," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 403–414.

[13] L. Chen, Y. Shan, W. Tian, B. Li, and D. Cao, "A fast and efficient double-tree rrtˆ*-like sampling-based planner applying on mobile robotic systems," *IEEE/ASME transactions on mechatronics*, vol. 23, no. 6, pp. 2568–2578, 2018.

[14] K. L. Cheung and A. W.-C. Fu, "Enhanced nearest neighbour search on the r-tree," *ACM SIGMOD Record*, vol. 27, no. 3, pp. 16–21, 1998.

[15] F. Dellaert, M. Kaess *et al.*, "Factor graphs for robot perception," *Foundations and Trends® in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017.

[16] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using gaussian processes and factor graphs." in *Robotics: Science and Systems*, vol. 12, no. 4, 2016.

[17] E. Donsky and H. J. Wolfson, "Pepcrawler: a fast rrt-based algorithm for high-resolution refinement and binding affinity estimation of peptide inhibitors," *Bioinformatics*, vol. 27, no. 20, pp. 2836–2842, 2011.

[18] C. Ericson, *Real-time collision detection*. Crc Press, 2004.

[19] Y. Feng, G. Hammonds, Y. Gan, and Y. Zhu, "Crescent: taming memory irregularities for accelerating deep point cloud analytics," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 962–977.

[20] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. IEEE, 2006, pp. 1243–1248.

[21] P. R. Florence, J. Carter, J. Ware, and R. Tedrake, "Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7631–7638.

[22] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2997–3004.

[23] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 3067–3074.

[24] M. Ghallab and D. G. Allard, "A: An efficient near admissible heuristic search algorithm," in *Proceedings 8th IJCAI*, 1983, pp. 789–791.

[25] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[26] J. E. Goodman, O. Rourke *et al.*, "Handbook of discrete and computational geometry," Chapman & Hall; CRC, Tech. Rep., 2004.

[27] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984, pp. 47–57.

[28] Y. Hao, Y. Gan, B. Yu, Q. Liu, S.-S. Liu, and Y. Zhu, "Blitzcrank: Factor graph accelerator for motion planning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[29] Y. Hao, B. Yu, Q. Liu, S. Liu, and Y. Zhu, "Factor graph accelerator for lidar-inertial odometry," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–7.

[30] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[31] E. Heiden, L. Palmieri, S. Koenig, K. O. Arras, and G. S. Sukhatme, "Gradient-informed path smoothing for wheeled mobile robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1710–1717.

[32] R. C. Hoetzlein, "cunsearch," https://github.com/InteractiveComputerGraphics/cuNSearch.

[33] R. C. Hoetzlein, "Fast fixed-radius nearest neighbors: interactive million-particle fluids." in *Fast fixed-radius nearest neighbors: interactive million-particle fluids. In GPU Technology Conference*, 2014.

[34] J. Hou, M. Goebel, P. Hübner, and D. Iwaszczuk, "Octree-based approach for real-time 3d indoor mapping using rgb-d video data," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 48, pp. 183–190, 2023.

[35] E. Huang, M. Mukadam, Z. Liu, and B. Boots, "Motion planning with graph-based trajectories and gaussian process inference," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5591–5598.

[36] L. Huang, X. Zang, Y. Gong, C. Deng, J. Yi, and B. Yuan, "Img-smp: Algorithm and hardware co-design for real-time energy-efficient neural motion planning," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, 2022, pp. 373–377.

[37] L. Huang, X. Zang, Y. Gong, and B. Yuan, "Hardware architecture of graph neural network-enabled motion planner," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–7.

[38] L. Huang, X. Zang, Y. Gong, B. Zhang, and B. Yuan, "Vlsi hardware architecture of neural a* path planner," in *2022 56th Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2022, pp. 334–338.

[39] c. Ichnowski and c. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1206–1212.

[40] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

[41] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "Rrt-smart: Rapid convergence implementation of rrt towards optimal solution," in *2012 IEEE international conference on mechatronics and automation*. IEEE, 2012, pp. 1651–1656.

[42] J. Jiang and K. Wu, "Cooperative pathfinding based on memory-efficient multi-agent rrt," *IEEE Access*, vol. 8, pp. 168 743–168 750, 2020.

[43] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2012.

[44] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[45] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[46] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[47] R. C. Lawson, L. Wills, and P. Tsiotras, "Gpu parallelization of policy iteration rrt," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 4369–4374.

[48] S. T. Leutenegger, M. A. Lopez, and J. Edgington, "Str: A simple and efficient algorithm for r-tree packing," in *Proceedings 13th international conference on data engineering*. IEEE, 1997, pp. 497–506.

[49] S. Lian, Y. Han, X. Chen, Y. Wang, and H. Xiao, "Dadu-p: A scalable accelerator for robot motion planning in a dynamic environment," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[50] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm." in *ICAPS*, vol. 5, 2005, pp. 262–271.

[51] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," *Advances in neural information processing systems*, vol. 16, 2003.

[52] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation." in *ICRA*, 2017, pp. 5759–5765.

[53] R. Luna, I. A. Şucan, M. Moll, and L. E. Kavraki, "Anytime solution optimization for sampling-based motion planning," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 5068–5074.

[54] Y. Manolopoulos, A. N. Papadopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-Trees: Theory and Applications: Theory and Applications*. Springer Science & Business Media, 2006.

[55] M. Muja, "Flann - fast library for approximate nearest neighbors," https://github.com/flann-lib/flann.

[56] M. Muja and D. Lowe, "Flann-fast library for approximate nearest neighbors user manual," *Computer Science Department, University of British Columbia, Vancouver, BC, Canada*, vol. 5, p. 6, 2009.

[57] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.

[58] S. Murray, W. Floyd-Jones, G. Konidaris, and D. J. Sorin, "A programmable architecture for robot motion planning acceleration," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 185–188.

[59] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. D. Konidaris, "Robot motion planning on a chip." in *Robotics: Science and Systems*, vol. 6, 2016.

[60] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, "The microarchitecture of a real-time robot motion planning accelerator," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[61] A. Nüchter, *3D robotic mapping: the simultaneous localization and mapping problem with six degrees of freedom*. Springer, 2008, vol. 52.

[62] R. Pinkham, S. Zeng, and Z. Zhang, "Quicknn: Memory and performance optimization of kd tree based nearest neighbor search for 3d point clouds," in *2020 IEEE International symposium on high performance computer architecture (HPCA)*. IEEE, 2020, pp. 180–192.

[63] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.

[64] A. Ranganathan and S. Koenig, "Pdrrts: Integrating graph-based and cell-based planning," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2799–2806.

[65] R. Robotics. Roboticarms pulse. [Online]. Available: https://rozum.com/robotic-arm/#about

[66] T. Robotics. Viperx 300 robot arm. [Online]. Available: https://www.trossenrobotics.com/viperx-300-robot-arm.aspx

[67] M. P. Strub and J. D. Gammell, "Advanced bit (abit): Sampling-based planning with advanced graph-search techniques," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 130–136.

[68] N. R. Sturtevant, Z. Zhang, R. Holte, and J. Schaeffer, "Using inconsistent heuristics on a search," in *Proceedings of the AAAI Workshop on Search Techniques in Artificial Intelligence and Robotics*, 2008.

[69] I. Sugiarto, C. Axenie, and J. Conradt, "Fpga-based hardware accelerator for an embedded factor graph with configurable optimization," *Journal of Circuits, Systems and Computers*, vol. 28, no. 02, p. 1950031, 2019.

[70] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser *et al.*, "Collision detection for deformable objects," in *Computer graphics forum*, vol. 24, no. 1. Wiley Online Library, 2005, pp. 61–81.

[71] UFACTORY. Ufactory xarm 7. [Online]. Available: https://www.ufactory.cc/product-page/ufactory-xarm-7

[72] J. M. Van Verth and L. M. Bishop, *Essential mathematics for games and interactive applications*. CRC Press, 2015.

[73] P. Verbari, L. Bascetta, and M. Prandini, "Multi-agent trajectory planning: A decentralized iterative algorithm based on single-agent dynamic rrt," in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 1977–1982.

[74] Wikipedia, "Space partitioning — wikipedia," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Space_partitioning

[75] T. H. Wong, G. Leach, and F. Zambetta, "An adaptive octree grid for gpu-based collision detection of deformable objects," *The Visual Computer*, vol. 30, no. 6-8, pp. 729–738, 2014.

[76] R. J. Wood, B. Finio, M. Karpelson, K. Ma, N. O. Pérez-Arancibia, P. S. Sreetharan, H. Tanaka, and J. P. Whitney, "Progress on "pico" air vehicles," in *Robotics Research: The 15th International Symposium ISRR*. Springer, 2017, pp. 3–19.

[77] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems," in *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 2, 2010, p. 3.

[78] S. Xiao, N. Bergmann, and A. Postula, "Parallel rrt architecture design for motion planning," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–4.

[79] T. Xu, B. Tian, and Y. Zhu, "Tigris: Architecture and algorithms for 3d perception in point clouds," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 629–642.

[80] Y. Yang, X. Chen, and Y. Han, "Dadu-cd: Fast and efficient processing-in-memory accelerator for collision detection," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[81] Y. Yang, S. Lian, X. Chen, and Y. Han, "Accelerating rrt motion planning using tcam," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 481–486.

[82] C. Yu and S. Gao, "Reducing collision checking for sampling-based motion planning using graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4274–4289, 2021.

[83] L. Yuan, J. Zhao, W. Li, and J. Hou, "Improved informed-rrt* based path planning and trajectory optimization for mobile robots," *International Journal of Precision Engineering and Manufacturing*, vol. 24, no. 3, pp. 435–446, 2023.

[84] X. Zang, M. Yin, L. Huang, J. Yu, S. Zonouz, and B. Yuan, "Robot motion planning as video prediction: A spatio-temporal neural network-based motion planner," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 12 492–12 499.

[85] X. Zang, M. Yin, J. Xiao, S. Zonouz, and B. Yuan, "Graphmp: Graph neural network-based motion planning with efficient graph search," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[86] R. Zhang, C. Yu, J. Chen, C. Fan, and S. Gao, "Learning-based motion planning in dynamic environments using gnns and temporal encoding," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 003–30 015, 2022.

[87] W. Zhang, X. Zang, L. Huang, Y. Sui, J. Yu, Y. Chen, and B. Yuan, "Dyngmp: Graph neural network-based motion planning in unpredictable dynamic environments," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[88] Y. Zhu, "Rtnn: accelerating neighbor search using hardware ray tracing," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 76–89.

[89] C. Zito, R. Stolkin, M. Kopicki, and J. L. Wyatt, "Two-level rrt planning for robotic push manipulation," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 678–685.