VAMP-MR: Vector-Accelerated Motion Planning and Execution for Multi-Robot-Arms

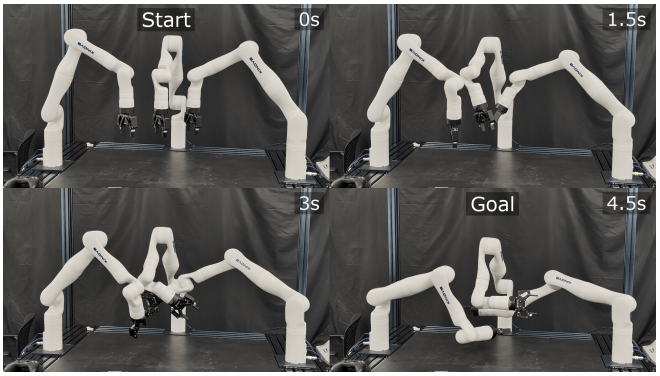
Philip Huang[1], Chenrui Gao[2], Jiaoyang Li[1]

***Abstract*— Multi-robot-arm motion planning is a key challenge in deploying multiple manipulators for industrial tasks such as manufacturing. Existing search-based and sampling-based solvers often require significant computation time to produce collision-free, high-quality motions suitable for safe real-world execution. In this work, we introduce a new suite of multi-robot-arm motion planners capable of near real-time motion generation, combining classical planning algorithms with state-of-the-art vectorized collision-checking techniques. Based on CPU SIMD instructions, our new planners accelerate their primary bottleneck, motion validation, and achieve up to two orders of magnitude speedup in both motion planning and execution postprocessing for multi-arm manipulation tasks. We also release the implementation of our vector-accelerated multi-robot planning and execution algorithms, and we believe this will lower the barrier for research and development of multi-robot-arm planning and manipulation problems. Code is available at `https://vamp-mr.github.io/vamp-mr`**

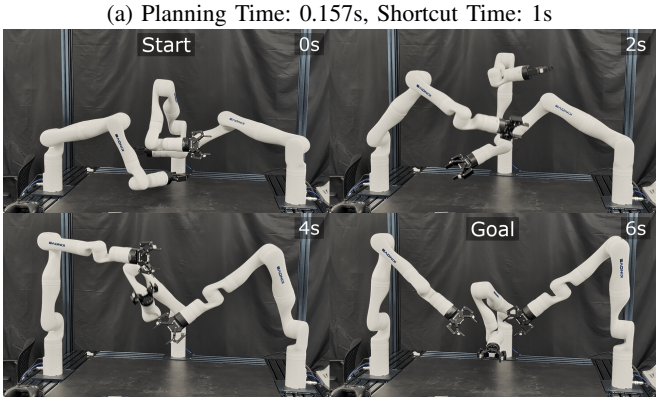
I. Introduction

Robotic systems have the potential to transform industries such as construction and manufacturing by automating hazardous and physically demanding tasks like welding, polishing, assembly, and handling heavy objects. In theory, deploying multiple robots within a shared workcell (such as Fig. 1) can greatly enhance efficiency and throughput through collaboration and parallelization. In practice, however, this vision remains elusive: coordinating teams of robot arms to operate safely, efficiently, and robustly is highly challenging, and most multi-robot setups still require extensive manual programming that is difficult to adapt when tasks or environments change.

In this work, we target a key bottleneck shared by nearly all multi-robot-arm motion planning (M-RAMP) methods—collision checking and motion validation. Our approach builds upon Vector-Accelerated Motion Planning (VAMP) [1], a high-performance single-robot-arm motion planner featuring CPU-SIMD-vectorized forward kinematics and collision checking, and extends it to multi-robot settings. SIMD (Single Instruction, Multiple Data) increases the throughput of processing robot geometry data by parallelizing vector operations over multiple robot configurations. As shown in Fig. 2, we introduce a general-purpose vectorized collision-checking module that supports multiple robot arms with flexible input definitions for relative transformations, robot count, attachments, obstacles, and configurations. This design allows our system to serve as a drop-in replacement



(a) Planning Time: 0.157s, Shortcut Time: 1s



(b) Planning Time: 0.162s, Shortcut Time: 1s

Fig. 1: Examples of multi-robot-arm trajectories planned with our vector-accelerated motion planner. We use composite RRT-Connect [4] and postprocess the trajectories with 1s of randomized shortcutting [5].

for collision-checking frameworks such as FCL [2] and Bullet [3], achieving up to 100× speedups in both motion planning and postprocessing—without requiring any other algorithmic changes. Our experiments demonstrate that standard planning and postprocessing pipelines can efficiently generate the motion for four robot arms in under a second on average, and that existing multi-agent pathfinding (MAPF) algorithms perform well even with minimum algorithmic adaptation to articulated robots. Lastly, we believe this work lowers the barrier to research in multi-robot-arm manipulation, including task and motion planning, and workcell layout optimization.

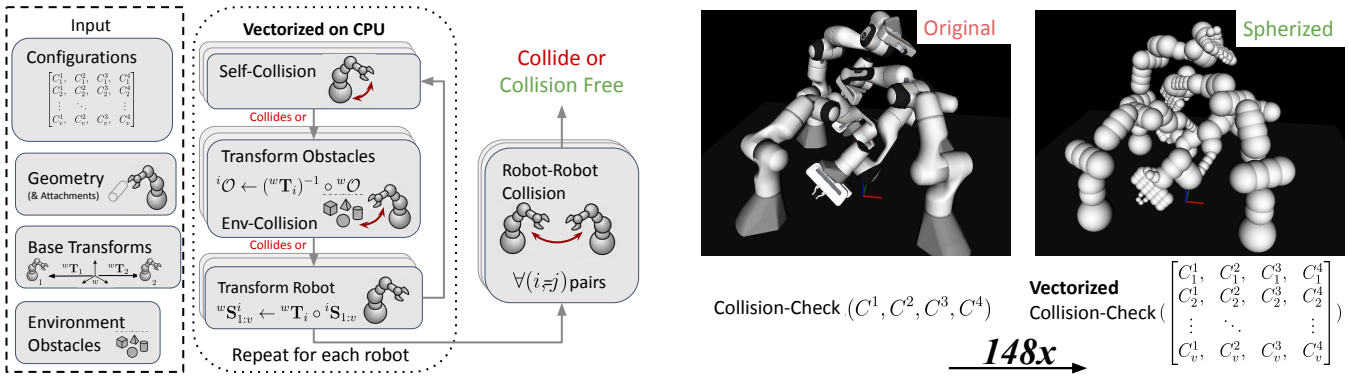
II. Background

A. Multi-Robot-Arm Motion Planning

Multi-Robot-Arm Motion Planning (M-RAMP) is an emerging research area focused on centrally coordinating

[1]Philip Huang and Jiaoyang Li is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Corresponding email: philiphuang@cmu.edu

[2]Chenrui Gao is with the University of Michigan, Ann Arbor.



(a) Our VAMP-MR method

(b) Speedup on Panda Four Arms

Fig. 2: Description of our multi-robot-arm collision checking primitive. Given environment geometries and a batch of multi-robot configurations, VAMP-MR can determine if any collisions exist within the batch up to two orders of magnitude faster than FCL-based methods. The key is to use spherized geometries and leverage SIMD instructions to parallelize over the batch, along with a structured arrangement of self, robot-environment, or robot-robot checks for early collision detection.

the motion of multiple robot arms operating in tight and cluttered environments. We define the M-RAMP problem as follows. Given a set of N robot arms, each with a fixed base transform ${}^{w}\mathbf{T}_i \in SE(3)$ and a configuration space $\mathcal{C}^i \subseteq \mathbb{R}^{\text{DoF}^i}$, where DoF^i denotes the number of degrees of freedom of robot i, all robots operate in a shared environment with obstacles ${}^{w}\mathcal{O}$. The objective is to find a set of collision-free joint space trajectories $\tau = \{\tau^1, \tau^2, \dots \tau^N\}$ from a pair of start configuration $[C^1_{\text{start}}, \dots, C^N_{\text{start}}]$ to a goal configuration $[C^1_{\text{goal}}, \dots, C^N_{\text{goal}}]$ for each robot.

A straightforward approach is to model all robots as a single composite system with the combined DoF and apply a single-agent planner such as RRT-Connect [4] or graph-of-convex-sets planning [6]. However, this approach can be computationally prohibitive as the dimensionality of the joint configuration space grows rapidly with the number of robots. To address scalability, researchers have extended multi-agent pathfinding (MAPF) techniques—originally developed for 2D discrete grid worlds—to continuous, high-dimensional configuration spaces. For instance, [7], [8], and [9] adapt variants of conflict-based search (CBS) [10] to incrementally plan motions for individual robot arms while resolving inter-robot collisions through a high-level tree search with motion constraints. Prioritized planning generates trajectories sequentially for each robot arm [11], [12], although these approaches lack the theoretical guarantees of CBS-style solvers. For executing a multi-robot-arm plan in the real world, uncertainties due to controller and sensor delay must be addressed. One solution is to use a temporal plan graph (TPG) [13], a partially ordered graph, to identify all potential collisions and coordinate each robot based on their precomputed actions and task precedences.

Beyond traditional planning-based techniques, learning-based approaches such as [14] train centralized neural networks to control the synchronized motion of multiple robot arms using reinforcement learning in randomized environments. Our proposed method is complementary to all these works: it can accelerate motion validation in both sampling- and search-based planners and can also speed up training by accelerating feature and reward evaluations in learning-based controllers.

B. Accelerated Motion Planning

For most planning- and control-based motion generation methods, forward kinematics (FK) and collision checking (CC) are among the most computationally expensive operations [15]. Other notable costs include nearest-neighbor searches in sampling-based planners and priority-queue management in search-based methods.

Numerous strategies have been proposed to reduce the computational burden of collision checking. Lazy evaluation delays collision checking until necessary (e.g., LazyPRM [16]), while experience-based methods exploit past search data to accelerate planning [8]. Learning-based approaches have trained approximate collision detectors [17] for sampling-based planning or learned signed distance fields [18] for gradient-based control. Other work amortizes the cost of motion generation via neural motion planners, such as MPNet [19], which produce collision-free trajectories directly from sensory input after large-scale training. In parallel, several efforts have explored hardware or parallelization strategies, such as parallel RRT on multi-core CPUs[20].

Our work aligns with approaches that directly parallelize collision checking itself. [15] parallelize collision checking on GPUs, and [21] implement robot-specific collision detection circuitry on FPGAs. However, these approaches often introduce nontrivial communication overhead. In complex settings such as task and motion planning, a search-based planner running on the CPU must repeatedly query a GPU or FPGA for motion feasibility across varying geometric configurations, and the resulting data transfer latency can accumulate substantially. Alternatively, we build upon the approach introduced by [1], which batches multiple configurations into a vector to evaluate forward kinematics and collisions with SIMD instructions.

C. Comparison of Traditional and Vector Accelerated Collision Checking (VAMP)

Traditional collision checkers such as FCL [2] and Bullet [3] are primarily designed for single-threaded execution and typically follow a common pipeline. Each collision object—such as a primitive, mesh, or point cloud—is represented by a hierarchical data structure that encodes both its bounding volumes (e.g., axis-aligned or oriented bounding boxes) and its underlying geometric primitives (e.g., triangles or points). During collision checking, the system updates the world transforms of all bounding volumes based on kinematics and recursively traverses the corresponding bounding volume hierarchies (BVHs). The process is generally divided into two stages: a broad-phase filter that identifies potentially overlapping bounding volumes, followed by a narrow-phase test (e.g., GJK [22]) to determine exact intersections. While this approach is widely adopted, it is inherently difficult to parallelize. The recursive BVH traversal introduces significant conditional branching and irregular workloads, complicating efficient task distribution across multiple threads. Moreover, the high memory bandwidth required to access BVH transforms and geometry data often becomes a limiting factor without careful data layout and caching strategies.

VAMP overcomes these challenges by fusing forward kinematics (FK) and collision checking (CC) into a single optimized kernel implemented as a C++ header. It reorganizes batches of joint-space configurations into a struct-of-arrays memory layout, which compactly represents a batch of robot configurations for data parallelism and reduces memory overhead. Robot geometries are approximated by sets of spheres [23], while obstacles are represented as simple primitives such as spheres, cubes, cylinders, or capsules. FK is unrolled via a custom tracing compiler that computes the positions of these spheres directly, minimizing branching and data dependencies. Self-collisions are interleaved with the FK computation, allowing early terminations if some links collide with each other. Computing FK and CC for each sphere becomes fully parallelizable for a batch of configurations, and if any one collides, the entire batch is rejected. With batching, VAMP discards traditional broad-phase CC and instead opts for a "rake" strategy. It evaluates a set of uniformly distanced configurations along an edge in a batch to maximize the likelihood of detecting collisions early for that edge. VAMP's vectorized FK and CC routines can be seamlessly integrated with search- or sampling-based planners, achieving millisecond-level motion planning.

While VAMP demonstrates the substantial impact of fast collision checking on single-robot motion planning, we bring these ideas to VAMP-MR, a vectorized multi-robot-arm collision checker that accelerates multiple stages of the planning pipeline, including motion planning, trajectory shortcutting, and safe execution.

III. Vectorized Multi-Robot Collision Checking

A. Method

Generating safe and high-quality motion for real-world multi-robot-arm systems involves several key stages—from task assignment and motion planning to postprocessing for execution. In many current multi-arm systems, such as those used for assembly [13], [12], collision checking remains the dominant computational bottleneck, e.g., in roadmap construction, task allocation, and motion validation. We overcome this bottleneck by accelerating forward kinematics and collision checking through approximate robot modeling and vectorized computation. This significantly reduces runtime overhead across the entire multi-robot planning and execution stack. Below, we describe the design of our vectorized collision-checking framework.

Algorithm 1 FK_CC_Multi: Vectorized FK and Collision Checking for Multiple Robots

Require: For each robot i: base transform ${}^{w}\mathbf{T}_i$, batch of configurations $\{C_j^i\}_{j=1}^{v}$, optional attachments $\mathcal{A}^i$; environment obstacles ${}^{w}\mathcal{O}$; allowed contacts $\mathcal{W}_{allow}$

for each robot i **do**
  ${}^{i}\mathbf{S}_{1:v} \leftarrow \mathrm{FK}(C_{1:v}^i, \mathcal{A}^i)$ ▷ spheres in robot-i frame
  if SelfCC$({}^{i}\mathbf{S}_{1:v})$ detects collision **then**
    return Invalid
  end if
  ${}^{i}\mathcal{O} \leftarrow ({}^{w}\mathbf{T}_i)^{-1} \circ {}^{w}\mathcal{O}$ ▷ obstacles in robot-i frame
  if EnvCC$({}^{i}\mathbf{S}_{1:v}, {}^{i}\mathcal{O}, \mathcal{W}_{allow})$ detects collision **then**
    return Invalid
  end if
  ${}^{w}\mathbf{S}_{1:v}^i \leftarrow {}^{w}\mathbf{T}_i \circ {}^{i}\mathbf{S}_{1:v}$ ▷ spheres in world frame
end for
for each robot pair $(i, k),\ i < k$ **do**
  if InterCC$({}^{w}\mathbf{S}_{1:v}^i, {}^{w}\mathbf{S}_{1:v}^k)$ detects collision **then**
    return Invalid
  end if
end for
return Valid ▷ iff all v batch lanes are collision-free

A naive approach to adapt VAMP for multi-robot planning would be to merge all DoFs across robots into a single composite system and generate a corresponding VAMP kernel. However, many multi-robot planning algorithms require distinguishing between different types of collisions—for example, counting inter-robot collisions in CBS-style motion planning, computing pairwise collisions between selected robot pairs in TPG construction, or separating self-, robot-environment-, and robot-robot collisions for single-robot roadmap generation. We also aim to simplify practical multi-robot-arm manipulation tasks, such as assembly, where it is useful to easily calibrate inter-robot transformations ${}^{w}\mathbf{T}_i$, adjust attachments $\mathcal{A}^i$, and specify intentional contacts between robots and shared environment obstacles $\mathcal{W}_{\text{allow}}$.

To support these capabilities, we design a new routine: FK_CC_Multi outlined in Algorithm 1. Given a batch of v sets of multi-robot configurations $(C_j^1, \ldots, C_j^n)$ for N robots—represented as a $v \times n$ matrix as illustrated in Fig. 2b, FK_CC_Multi computes forward kinematics, checks collisions, and returns a boolean validity flag if and only if all v sets of multi-robot configurations are collision-

Fig. 3: Multi-robot motion planning environments.

free. Each robot first passes through the single-robot FK then CC routine to perform self-collision checks. If a collision is detected, the configuration is immediately rejected. Environment obstacles ${}^{w}\mathcal{O}$ transformed to the robot's base frame and checked against each robot while ignoring any allowed environment collisions $\mathcal{W}_{\text{allow}}$. In practice, we cache transformed environment obstacles in each robot's base frame since base transforms and obstacles often remain fixed during planning, which allows robot-environment collisions to be detected quickly. Then, each robot's spheres are transformed to the world frame based on its base transform ${}^{w}\mathbf{T}_i$. Subsequently, the sphere representations of all robots are compared pairwise to detect robot-robot collisions. This modular design is extremely flexible and still enables efficient, vectorized collision evaluation for complex multi-robot-arm setups.

Fig. 2a illustrates these steps. Compared to the single-robot VAMP collision checker [1], VAMP-MR (a) natively supports robot-robot collision checking required for multi-arm systems and (b) enables flexible multi-robot environment specification for easy integration with existing multi-robot planners without kernel recompilation.

B. Evaluation of Collision Checking and Motion Validation

To evaluate the runtime improvement of our new collision checking method, we test on three challenging multi-robot-arm environments—Panda Two Rod, Panda Four, and Panda Four Bins (see Fig. 3)—introduced in [5]. Each Panda arm has 7 DoF, and the 59-sphere approximation in Fig. 2b is generated with the tool in [23]. These environments also involve attachments in Panda Two Rod and obstacles in Panda Four Bins.

We first assess the speedup achieved by our vectorized collision-checking primitives using randomly sampled robot configurations. Table I reports results over 10,000 random samples for both (1) collision checking of single sets of multi-robot configurations and (2) motion validation between pairs of configurations. The collision checking resolution, defined as L_1 distance between two consecutive interpolated configurations along the motion, is set to 0.1 radian. To ensure consistent modeling, we use the same simplified spherized robot geometry representation in FCL. All experiments are conducted on an AMD 7840HS Laptop CPU. We use C++ compiled with GCC 9 and 256-bit AVX2 instructions (e.g., -march=native, -mavx2, -O3). This allows a SIMD batch size of 8 with 32-bit single-precision floating-point for vectorized collision checking.

Our method achieves 11-27x speedup for single-configuration collision checking, and up to 148x speedup for motion validation between two random configurations. The compiler optimization of the robot-specific forward kinematics and collision checking kernel makes our FK_CC_MULTI routine significantly faster for spherized robot collision checking. For motion validation, the runtime speedup sometimes even exceeds the single-check speedup by more than 8 times (i.e., the number of SIMD lanes). We believe this shows the combined effect of vectorization, good cache utilization, and the "rake"-style scan when evaluating discretized configurations during motion validation. We also observe less variance across different environments compared to FCL. This is because randomly sampled motions between a pair of configurations are very likely to collide in the obstacle-rich Panda Four Bins environment, and FCL can invalidate a motion much more quickly than in other environments.

TABLE I: Average collision checking runtime and speedup over 10,000 random samples. Our method is compared against FCL with the same approximated spherized geometries to ensure consistency.

Check	Environment	FCL (µs)	Ours (µs)	Speedup
Single	Panda Two Rod	100.99	8.60	**11.7x**
	Panda Four	206.53	15.01	**13.7x**
	Panda Four Bins	382.63	13.70	**27.9x**
Motion	Panda Two Rod	3936.13	36.03	**109.2x**
	Panda Four	4836.50	32.61	**148.3x**
	Panda Four Bins	930.03	14.23	**65.4x**

IV. Applications of Vector-Accelerated Collision Checking

A. Vectorized Multi-Robot-Arm Planning

Building on our vectorized multi-robot collision checking framework, we integrate it into two multi-robot-arm motion planners to enable fast and scalable multi-robot-arm motion planning. Our first planner is composite RRT-Connect [4], which combines the DoFs from all robots and treats them as a single robot for the RRT-Connect algorithm. We integrate our FK_CC_MULTI routine in RRT-Connect to accelerate collision checking for validating randomly sampled configurations and motions during tree expansion.

Our second planner is based on CBS-MP [7], a conflict-based search (CBS) motion planner originally designed for multi-agent systems operating on roadmaps. In CBS-MP, each agent (in our case, a robot arm) constructs its own probabilistic roadmap (PRM) [24]. The planner then incrementally searches for collision-free trajectories, resolving inter-robot collisions through a high-level constraint tree that imposes avoidance constraints. However, the original avoidance constraint used in CBS-MP is incomplete, as discussed in [8]. To ensure theoretical completeness of CBS on the roadmaps, we adopt an asymmetric constraint scheme inspired by [25]. Specifically, when a collision occurs at time t between robot i and robot j, we create two mutually exclusive constraints during high-level CBS expansion: one forbidding robot i from occupying any volume of robot j's volume at time t, and the other forbidding robot j from

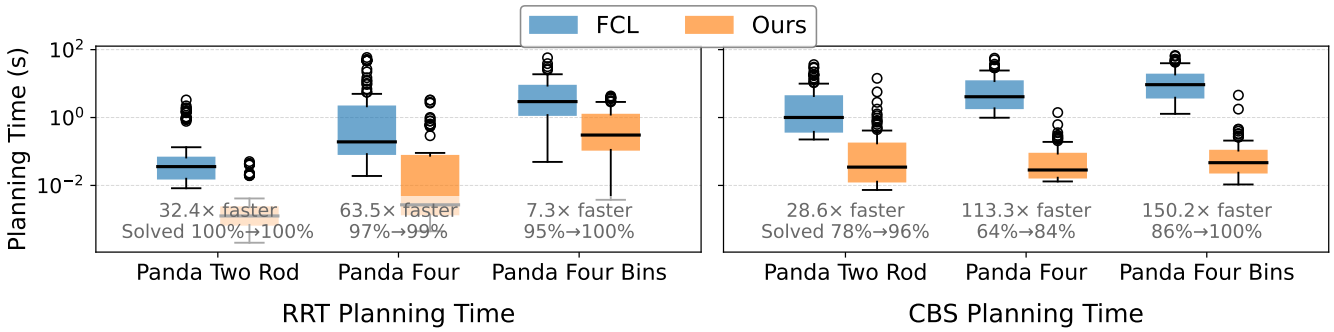


Fig. 4: Planning time comparison between FCL-based and our VAMP-based motion planners. The median time of both RRT and CBS planning of solved instances in each environment is up to 150x faster with vectorization. Reported speedups are averaged over all planning instances in each environment.

TABLE II: Cache behavior comparison between FCL-based and our VAMP-based motion planners. We report the cache miss rate and cache misses per 1,000 instructions (MPKI).

Environment	Planner	Cache Miss (%)		MPKI	
		FCL	Ours	FCL	Ours
Panda Two Rod	CBS	4.45	**0.79**	0.83	**0.38**
	RRT	**4.11**	7.49	**0.85**	2.08
Panda Four	CBS	5.49	**1.86**	0.83	**0.65**
	RRT	7.92	**2.43**	1.49	**0.55**
Panda Four Bins	CBS	8.94	**2.13**	1.50	**0.83**
	RRT	13.81	**1.72**	2.62	**0.51**

taking the same configuration at that time. With our SIMD-accelerated FK_CC_MULTI routine, we can significantly reduce the computational overhead in roadmap construction, constraint evaluation, and collision checking within CBS-MP.

Results To evaluate how VAMP-MR can improve planning time and analyze its algorithmic implications, we create 110, 132, and 132 unique pairs of start and goal poses as different planning instances in Panda Two Rod, Panda Four, and Panda Four Bins, as in [5]. We evaluate both the composite RRT-Connect and CBS-MP motion planner. We sample 5,000 nodes for each robot's probabilistic roadmap before planning in CBS-MP and run both planners with a one-minute timeout, excluding roadmap construction time.

Fig. 4 shows the planning time before and after vector acceleration for all three environments. Our results show that vectorization provides at least an order-of-magnitude speedup for both planners, with even greater improvement for CBS-MP. Table II compares the cache behavior of FCL and our VAMP collision checker across CBS and RRT planners. In general (except RRT in the Panda Two Rod environment), we notice that VAMP-MR achieves substantially lower cache miss rates, indicating improved memory locality during the frequent collision queries required by multi-arm planners.

For RRT-Connect, the runtime acceleration correlates closely with the motion validation speedup reported in Table I, as motion validation dominates the computational cost. However, CBS-MP exhibits the largest performance gain in Panda Four Bins, reaching a 100% success rate, whereas some instances in Panda Four remain unsolved. The difference results from the coordination requirement of the tasks. In Panda Four, several target poses (e.g., the one in Fig. 2b) require precise sequencing between robot arms, which leads to thousands of high-level CBS expansions necessary. Conversely, Panda Four Bins requires less coordination, as long as each robot can navigate around obstacles independently. Thus, CBS outperforms RRT in Panda Four Bins as it can quickly deconflict agents and find collision-free solutions. We also noticed that the number of expanded constraint tree (CT) nodes in CBS increased significantly. For example, our vectorized CBS can solve an instance with 5,303 expanded CT nodes on Panda Four, compared to a maximum of 53 expanded CT nodes among solved Panda Four instances with FCL with the same one-minute time limit.

These results demonstrate that our vectorized framework effectively addresses the primary bottleneck in motion planning—motion validation and constraint evaluation—enabling CBS-based methods to scale to more complex setups. However, even with accelerated collision checking, it remains the primary computational bottleneck, accounting for 90.2% of planning time for RRT and 64.6% for CBS in Panda Four. This highlights the potential for applying advanced CBS or MAPF techniques to multi-robot-arm systems, such as symmetry reasoning [26], enhanced bounded-suboptimal search [27], and even fast suboptimal planners [28], to further address the planning challenge.

B. Vectorized Multi-Robot Shortcutting

Shortcutting is a widely used postprocessing technique for improving the smoothness and optimality of planned trajectories, particularly in multi-robot-arm motion planning [8], [11]. This is because, given finite time, search-based and sampling-based motion planners are typically not optimal in the continuous space. Sampling-based planners are only probabilistically complete and asymptotically optimal, and CBS is only resolution-complete and resolution-optimal given the roadmap. Performance can usually be improved more with postprocessing.

We define a shortcut as a linear interpolation between two (composite) configurations. The endpoints of the shortcut

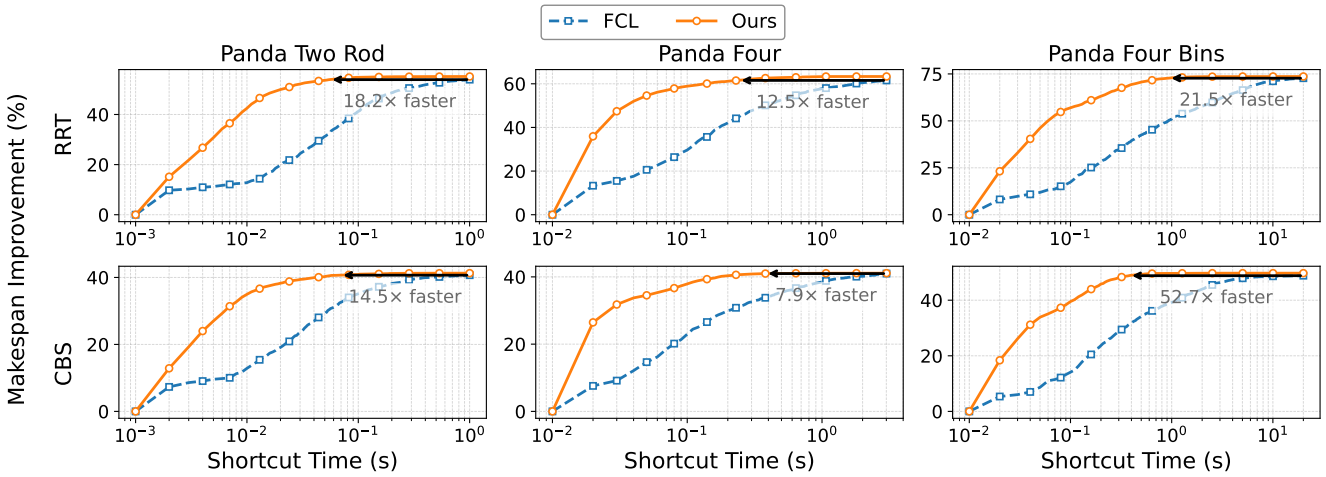


Fig. 5: Average makespan improvement over time of multi-robot shortcutting across three environments, using DTS [5]. The performance-runtime curves are averaged across all instances in each environment. Our method achieves the same level of makespan improvement within one second and is up to 50 times faster than FCL.

are randomly sampled along the current trajectory. After sampling a shortcut, we check whether replacing the original segment with the shortcut introduces any collisions with other robots or obstacles. If the shortcut is collision-free, the trajectory is updated. This process repeats until a time limit is reached. Since collision checking dominates the computational cost of shortcutting, this process is highly compatible with our vectorized collision-checking routine.

We adopt the Dynamic Thompson Shortcutting (DTS) algorithm from [5], an anytime shortcutting framework that adaptively combines three complementary strategies: composite, prioritized, and path shortcutting. A composite shortcut is sampled in the composite configuration space and jointly modifies all robot trajectories. A prioritized shortcut modifies a single robot trajectory while also reducing the timesteps of configurations following the shortcut endpoint. A path shortcut is similar to prioritized shortcutting but preserves the original trajectory timing. Empirically, composite shortcutting often provides rapid early-stage improvement, while prioritized and path shortcutting can converge to higher-quality solutions, albeit more slowly. DTS formulates the selection among these strategies as a multi-armed bandit problem and uses Thompson sampling to balance fast early improvement with final trajectory quality.

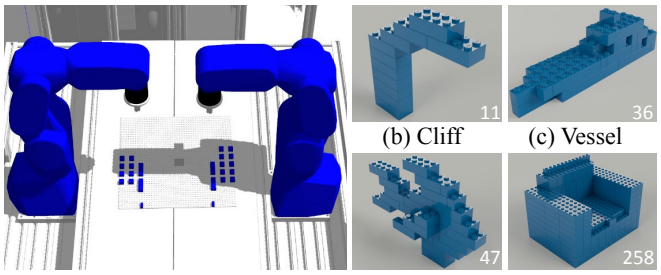
Results We evaluate the performance improvement of DTS, as shown in Fig. 5, using initial trajectories generated by RRT and CBS. Without any other modifications, integrating a vectorized collision checking routine reduces the time required to converge to final shortcut trajectories by up to 50x. The vector-accelerated shortcutter achieves substantial improvement even within 0.1s, and all trajectories converge within 1s. When combined with either an RRT or CBS motion planner, this enables fast generation of high-quality trajectories for multi-robot-arm systems.

C. Vectorized Safe Execution Framework

For many multi-robot-arm manipulation tasks, such as object rearrangement or collaborative assembly, it is often necessary to execute a sequence of coordinated multi-robot trajectories derived from a higher-level task plan. However, safe execution in real-world environments must account for execution uncertainties—such as kinematic inaccuracies and unpredictable physical interactions with the environment. Beyond modeling challenges, many manipulation tasks also use closed-loop control or learned policies that make it impossible to predict accurate execution time in planning.

We build upon the multi-modal Temporal Plan Graph (TPG) framework introduced by [13], which systematically postprocesses a given multi-robot task and motion plan to enable safe, asynchronous execution under such uncertainties. A multi-modal TPG $G = (V, E)$ is a partially ordered graph that captures kinematic transitions, changes in the environments due to objects being moved, and inter-robot task and motion dependencies. Each node v_n^i is either a pose node that corresponds to a target configuration C_n^i for robot i or a skill node that corresponds to a manipulation skill. A manipulation skill is defined as a set of object-centric motions executed by a feedback controller. We assume that each skill has a reference robot trajectory for the purpose of TPG computation. Each edge $v_n^i \rightarrow v_{n'}^{i'}$ encodes a precedence constraint between two nodes. Edges between nodes of the same robot are added between every consecutive node, and inter-robot edges can be added for task dependencies or motion dependencies to prevent executing two spatially colliding nodes simultaneously.

During execution, each node can be safely executed if all incoming nodes are completed. Each robot maintains its own action queue and the TPG serves as a central scheduler that sends nodes that can be safely executed to the action queues. Each robot's controller executes nodes in its action queue and updates node completion status back to the central



(a) Robot Environment (b) Cliff (c) Vessel (d) Fish (e) Chair

Fig. 6: Dual-arm LEGO assembly environment in (a) and assembly tasks in (b)-(e). Numbers in (b)-(e) indicate the number of assembly steps (LEGO bricks).

scheduler, allowing new nodes to be scheduled. The partial-order structure naturally tolerates execution delays, since each node can take an arbitrary amount of time to complete.

Constructing a TPG involves converting a sequential or synchronous task and motion plan into a temporally dependent multi-robot schedule. This process requires finding all inter-robot motion dependencies with extensive pairwise collision checking across all robot trajectories, which scales quadratically with the number of robots N and the number of discrete trajectory waypoints. While Huang et al. [13] used CPU multithreading to parallelize this computation, our vectorized collision-checking routine is even better suited to accelerate TPG construction with less communication overhead. However, the original TPG formulation requires identifying all colliding pairs of configurations, whereas our vectorized collision checker FK_CC_MULTI terminates early if any configuration in the batch collides. To effectively leverage SIMD parallelism within this framework, we propose a modified grouping strategy.

For a group of k consecutive pose nodes $[v_n^i, \dots v_{n+k-1}^i]$ of the TPG, we merge them to a larger transit node $\mathbf{v}_n^i$ that represents a short path segment $[C_n^i, \dots C_{n+k-1}^i]$. We then perform collision checks between every pair of transit nodes from different robots. If any part of the path segment of a transit node $\mathbf{v}_n^i$ collides with that of another transit node $\mathbf{v}_{n'}^{i'}$, we insert an inter-robot edge $\mathbf{v}_n^i \to \mathbf{v}_{n'}^{i'}$ from the earlier node to the later node ($n < n'$) that prevents collision in execution. In this formulation, we only need to determine whether any part of the two path segments of two transit nodes $\mathbf{v}_n^i$ and $\mathbf{v}_{n'}^{i'}$ collide without iterating over individual configurations. This reduces the computational complexity of identifying inter-robot motion dependencies when paired with our vectorized collision checker. Although this reduces the number of edges and is more conservative than the old TPG formulation, empirically, the execution makespan difference is negligible. As a result, our approach achieves much more efficient TPG construction for complex multi-robot assembly tasks.

Results We evaluate four long-horizon dual-arm LEGO assembly tasks (see Fig. 6) from APEX-MR [13], which involve up to 258 assembled parts. These tasks provide more realistic and complex test cases for multi-robot-arm systems. For each task, we measure the runtime across all stages of the pipeline, including task assignment, motion planning, and TPG construction. We follow the methodologies and experimental setup of [13]. Given a sequential assembly plan, we perform task planning and assign robot, grasping, and support poses targets for each assembly step using an integer-linear program. The motion of each transit task and each manipulation skill are planned sequentially with single-agent RRT-Connect. RRT-Connect trajectories can be jerky and suboptimal, so the resulting motion is passed to a vectorized single-agent shortcutter for 0.1s [29]. The TPG execution framework then converts the sequential task and motion plan to an asynchronous, parallelized execution order. The makespan of TPG is further optimized with randomized TPG shortcutting as in [13].

TABLE III: Runtime and makespan of dual-arm LEGO assembly plans generated following the procedure in APEX-MR [13], with and without vector acceleration. Results are averaged over 4 random seeds. Note that FCL uses 16 CPU threads to parallelize TPG construction, whereas our collision checker is single-threaded.

Metric	**Cliff**	**Vessel**	**Fish**	**Chair**
TPG Shortcut Time (s)	1.0	1.0	1.0	5.0
FCL-Based				
Task Assignment (s)	0.60	4.90	5.81	13.0
Motion Planning (s)	0.63	0.76	4.91	9.98
TPG Construction (s)	9.88	25.6	66.0	817.2
Total Time (s)	**12.1**	**32.2**	**71.5**	**845.2**
Final Makespan (s)	**185**	**397**	**549**	**2180**
Ours				
Task Assignment (s)	0.55	4.68	5.45	10.2
Motion Planning (s)	0.42	0.15	1.96	0.88
TPG Construction (s)	1.42	2.55	3.11	43.22
Total Time (s)	**3.39**	**8.38**	**11.8**	**59.3**
Final Makespan (s)	**159**	**340**	**489**	**2146**
# of Bricks	11	36	29	258

We implement the RRT-Connect algorithm without using MoveIt for sequential motion planning for both FCL and our vector-accelerated planner. FCL uses the original robot mesh for collision checking since we found it is faster than using spheres. We also found that our custom RRT-Connect implementation is significantly faster than the result reported in [13].

Table III summarizes the runtime of each planning step and the final makespan after shortcutting. Motion planning is accelerated between 1.25x to 11.3x, and TPG construction is accelerated between 6.9x to 21x. In particular, the FCL-based TPG construction uses 16 CPU threads for parallelization, whereas our modified TPG construction with vector acceleration uses a single thread. We find that multi-core parallelization is largely unnecessary with vectorization due to additional communication overhead. Our accelerated planner produces higher-quality solutions after TPG shortcutting. With the same one-second time limit, vectorization achieves more than 10% lower makespan for cliff, vessel, and fish; however, the improvement for chair is minimal due to the large overhead of modifying obstacle positions when evaluating random shortcuts. Finally, the task assignment component has little improvement, as our acceleration affects

only the verification of collision-free grasp and support poses. The integer-linear program responsible for computing optimal assignments can be a significant bottleneck in the overall task and motion planning pipeline.

V. Real Robot Deployment

We deploy our vector-accelerated multi-robot-arm motion planner and anytime shortcutter on three Kinova arms in the real world, as shown in Fig. 1. The experiment consists of consecutively planning and executing motions through 11 complex and highly entangled configurations. We use the composite RRT planner to generate an initial trajectory and then post-process it with our anytime shortcutter for 1 s to improve trajectory quality. The resulting trajectory is further time-parameterized using TOPPRA [30] in the composite configuration space to satisfy the acceleration limits of the Kinova arms. Since the controller delay is minimal in the absence of contact or manipulation policies, the robots execute the planned trajectories synchronously without requiring a TPG. Across the 11 planning calls, the median planning time is 0.16 s with a sample standard deviation of 0.70 s. The relatively large standard deviation is due to two difficult instances that required 2.36 s and 1.06 s to solve. Video is provided in the supplementary material.

VI. Conclusion

We present VAMP-MR, a CPU-accelerated multi-robot-arm collision checking framework tightly integrated with multi-robot-arm motion planning algorithms. We deliver 10 to 100x speedup across diverse planning and shortcutting benchmarks without requiring any algorithmic changes. Our framework also accelerates the asynchronous planning and execution framework for long-horizon, dual-arm LEGO assembly tasks. We believe this work is a crucial step towards lowering the barrier to developing efficient multi-robot-arm planning algorithms. Potential future research directions include integrating advanced MAPF techniques into manipulation and exploring real-time, online planning for long-horizon multi-robot-arm tasks.

References

[1] W. Thomason, Z. Kingston, and L. E. Kavraki, "Motions in microseconds via vectorized sampling-based planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 8749–8756, 2024.

[2] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3859–3866, 2012.

[3] E. Coumans and Y. Bai, "PyBullet, a python module for physics simulation for games, robotics and machine learning." `http://pybullet.org`, 2021. Accessed: 2026-01-06.

[4] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 995–1001, 2000.

[5] P. Huang, Y. Shaoul, and J. Li, "Benchmarking shortcutting techniques for multi-robot arm motion planning," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pp. 13258–13265, 2025.

[6] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science Robotics*, vol. 8, no. 84, p. eadf7843, 2023.

[7] I. Solis, J. Motes, R. Sandström, and N. M. Amato, "Representation-optimal multi-robot motion planning using conflict-based search," *IEEE Robotics and Automation Letters*, pp. 4608–4615, 2021.

[8] Y. Shaoul, I. Mishani, M. Likhachev, and J. Li, "Accelerating search-based planning for multi-robot manipulation by leveraging online-generated experiences," in *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 523–531, 2024.

[9] Y. Shaoul, R. Veerapaneni, M. Likhachev, and J. Li, "Unconstraining multi-robot manipulation: Enabling arbitrary constraints in ECBS with bounded sub-optimality," in *Proceedings of the International Symposium on Combinatorial Search*, pp. 109–117, 2024.

[10] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[11] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, "Long-horizon multi-robot rearrangement planning for construction assembly," *IEEE Transactions on Robotics*, pp. 239–252, 2023.

[12] J. Chen, J. Li, Y. Huang, C. Garrett, D. Sun, C. Fan, A. Hofmann, C. Mueller, S. Koenig, and B. C. Williams, "Cooperative task and motion planning for multi-arm assembly systems," 2022. arXiv preprint arXiv:2203.02475.

[13] P. Huang, R. Liu, C. Liu, and J. Li, "APEX-MR: Multi-robot asynchronous planning and execution for cooperative assembly," in *Proceedings of Robotics: Science and Systems*, 2025.

[14] M. Lai, K. Go, Z. Li, T. Kröger, S. Schaal, K. Allen, and J. Scholz, "RoboBallet: Planning for multirobot reaching with graph neural networks and reinforcement learning," *Science Robotics*, vol. 10, no. 106, p. eads1204, 2025.

[15] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT*," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pp. 3513–3518, 2011.

[16] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 521–528, 2000.

[17] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1096–1114, 2020.

[18] M. Koptev, N. Figueroa, and A. Billard, "Neural joint space implicit signed distance functions for reactive robot manipulator control," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 480–487, 2023.

[19] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48–66, 2021.

[20] J. Ichnowski and R. Alterovitz, "Scalable multicore motion planning using lock-free concurrency," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1123–1136, 2014.

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

[22] 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.

[23] S. Coumar, G. Chang, N. Kodkani, and Z. Kingston, "FOAM: A tool for spherical approximation of robot geometry," 2025. arXiv preprint arXiv:2503.13704.

[24] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[25] J. Li, P. Surynek, A. Felner, H. Ma, T. S. Kumar, and S. Koenig, "Multi-agent path finding for large agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7627–7634, 2019.

[26] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, vol. 301, p. 103574, 2021.

[27] J. Li, W. Ruml, and S. Koenig, "EECBS: A bounded-suboptimal search for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 12353–12362, 2021.

[28] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 10256–10265, 2022.

[29] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.
[30] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018.