Deterministic Static Grid-Maze Planner Benchmarking for Reproducible Robotics Experiments

Cesar Contreras

***Abstract*—This paper presents a deterministic maze-navigation benchmark stack for reproducible robotics experiments across multiple simulation backends. The implementation unifies planner execution, path validation, benchmarking, and artifact generation in a single repository workflow built around deterministic seeds and explicit benchmark exports. In the executed snapshot, the benchmark evaluates 12 planners on 50 generated 15×15 mazes with complete trial-level reporting for success, runtime, path length, and expansions. All planners solve all trials; within this static regime, weighted A* has the lowest mean solve time (0.35 ms), while search-effort profiles show non-trivial runtime-expansion tradeoffs across methods. The pipeline also includes deterministic simulation regression checks with fixed screenshot expectations to support repeatability audits. The contribution is a repository-grounded, traceable benchmarking workflow for static grid-maze navigation; claims are intentionally scoped to this benchmark regime and do not assert general superiority in dynamic or kinodynamic navigation domains.**

***Index Terms*—mobile robot navigation, deterministic benchmarking, path planning, reproducibility, simulation**

I. Introduction

Grid-based planning remains a practical foundation for robotic navigation in constrained environments, and recent surveys show that graph-search, heuristic, and hybrid variants remain competitive baselines in deployed systems [1], [2], [3], [4]. A persistent experimental gap is not planner availability, but comparable execution: simulator differences, heterogeneous planner return schemas, and weak path-validity checks can dominate reported deltas. This work targets that gap for one bounded domain: static occupancy-grid mazes.

The implemented stack links a command-level entrypoint (`scripts/sim_runner.py`) to deterministic execution in `robotics_maze/src/main.py`: argument parsing to `RunConfig`, per-episode seeding, module discovery for maze generator/planner/simulator, and standardized episode logging. The runtime is robust to machine variability by selecting PyBullet or MuJoCo when available, with deterministic fallback when dependencies are missing. This emphasis on portable execution is aligned with recent ROS2/cloud robotics infrastructure trends [5], [6], [7], [8].

The same repository provides a benchmark harness (`robotics_maze/src/benchmark.py`) that normalizes planner outputs, validates geometric path feasibility against occupancy grids, and emits trial-level CSV/Markdown artifacts. In the tracked snapshot (`robotics_maze/results/benchmark_summary.md`), 12 planners are evaluated on 50 generated 15×15 backtracker mazes.

This paper contributes three bounded, repository-grounded results:

- **Infrastructure method contribution:** a deterministic planner-evaluation runtime that combines explicit seed propagation, heterogeneous planner I/O normalization, geometric path validation, and backend fallback behavior in one executable workflow.
- **Evaluation-protocol contribution:** a shared-success comparability policy with rotated planner order and deterministic lexicographic ranking by success rate, comparable solve time, mean expansions, and overall mean solve time. Path length is reported descriptively but excluded from ranking when any-angle outputs are present.
- **Evidence contribution:** an executed benchmark snapshot (12 planners, 50 mazes) with trial-level artifacts, plus deterministic simulation regression checks and screenshot outputs for reproducibility auditing.

Traceability artifacts (claims tables, figure manifests, citation audits) are included to support reproducibility and reviewability, but are not presented as standalone novelty claims.

The remainder of the manuscript details the implemented architecture and protocol, reports current benchmark evidence, and delineates what remains future work.

II. Related Work

Recent navigation literature continues to improve planner quality across classical, hybrid, and learned approaches [1], [2], [3], [4], [9], [10]. At the same time, multiple surveys report that reproducibility bottlenecks often come from integration variance (runtime stack, simulator configuration, and evaluation protocol) rather than algorithmic novelty alone [11], [12].

For this reason, the closest prior art for our scope is benchmark/runtime infrastructure rather than a single planner family. Benchmark platforms such as Bench-MR [13], MotionBenchMaker [14], and CoBRA [15] focus on reusable benchmark construction and reporting. Runtime frameworks in ROS2/cloud settings emphasize deployment portability and orchestration across heterogeneous environments [5], [6], [7], [8]. Our manuscript is positioned at their intersection for one bounded regime: deterministic static grid-maze planner comparison with repository-local traceability.

Cesar Contreras is an independent robotics researcher based in California, USA.

TABLE I
CLOSEST INFRASTRUCTURE-ORIENTED PRIOR WORK AND THE BOUNDED INCREMENTAL DELTA OF THIS PAPER.

Closest work	Reported primary scope	Incremental delta in this paper
Bench-MR [13]	Motion-planning benchmark for wheeled mobile robots.	Repository-level deterministic seed propagation tied to CLI entry-points, plus trial-level geometric path raster validation before success is counted.
MotionBenchMaker [14]	Tooling for generating and benchmarking motion-planning datasets.	Planner output normalization across heterogeneous Python planner schemas, coupled with shared-success comparability ranking and committed CSV/Markdown artifacts.
CoBRA [15]	Composable benchmark framework for robotics applications.	Single-stack integration of planner benchmark execution with explicit `pybullet` $\rightarrow$ `mujoco` $\rightarrow$ deterministic fallback runtime behavior and deterministic screenshot-regression checks.

Adjacent systems such as FogROS2-LS and Arena 4.0 provide important deployment/simulation context [7], [8], but are not treated as the closest benchmark-protocol comparators in Table I. The concrete incremental delta claimed in this paper is integration-level: not a new planner, but a deterministic execution-and-evaluation stack that couples seed policy, output normalization, geometric path validation, and backend-fallback simulation regression in one auditable repository workflow.

III. METHOD

A. Implemented System Architecture

The implemented architecture is a modular runtime centered on `robotics_maze/src/main.py`, with four concrete layers:

1) **Orchestration layer** (`scripts/sim_runner.py`, `main.py`): parses command-line arguments into `RunConfig`, supports optional GUI setup overrides, and executes an episode loop.
2) **Maze and planning layer** (`maze.py`, `planners.py`, `alt_planners/`): generates deterministic mazes and computes paths using baseline and alternative planners.
3) **Simulation layer** (`sim.py`, `robot.py`, `geometry.py`): converts plans to waypoints, instantiates obstacles from maze walls, and executes motion in physics backends.
4) **Evaluation layer** (`benchmark.py`): runs planner trials, validates returned paths, ranks methods, and writes CSV/Markdown artifacts.

For episode $e \in \{1, \ldots, E\}$, `main.py` first computes an episode seed $s_e^{\text{main}} = s_0 + (e - 1)$, where s_0 is the user-provided base seed. The current deterministic generator adapter then applies an additional episode offset, yielding the effective maze-generation seed $s_e^{\text{gen}} = s_e^{\text{main}} + e = s_0 + 2e - 1$. This manuscript reports the behavior as currently implemented in code.

B. Maze Generation and Grid Conversion

The maze model (`Maze` dataclass) stores explicit horizontal and vertical wall arrays, start/goal cells, and generation metadata. Two algorithms are implemented: recursive backtracker and randomized Prim. After carving, generation is accepted only if (i) all $W \times H$ cells are reachable from start and (ii) a BFS shortest path exists between start and goal. This provides a deterministic solvable-maze contract before planning is invoked.

For planner compatibility, `benchmark.py` converts wall-based mazes into occupancy grids of size $(2H+1) \times (2W+1)$, where free corridors are explicitly opened between neighboring cells. Start and goal are transformed from cell coordinates into occupancy-grid indices and forced free.

C. Planner Interface and Normalization

Baseline planners are registered by name in `planners.py` (A*, Dijkstra, BFS, and greedy best-first, plus aliases). Additional methods are loaded from `alt_planners/` through module-symbol mapping. To support heterogeneous planner APIs, `FunctionPlannerAdapter` and benchmark-side normalization convert outputs to a common payload with path plus metrics (e.g., expansions and runtime).

Planner outputs are accepted from multiple schemas (dictionary, tuple, or raw path sequence), then normalized by `_normalize_planner_output`. Path validity is enforced by `_validate_and_measure_path`: endpoints must match benchmark start/goal, all path cells must stay in bounds and collision free, and each segment is checked with Bresenham rasterization before success is credited.

D. Simulation Backends and Controller Path

`MazeEpisodeSimulator` receives `(maze, plan)` and performs three steps:

1) extract or infer a path (`path`, `waypoints`, tuple payload, or maze fallback),
2) map the path to world-frame waypoints (including occupancy-grid to metric conversion),
3) execute via selected backend (`pybullet`, `mujoco`, or deterministic kinematic fallback).

Backend selection is explicit: `auto` prefers PyBullet when available, then MuJoCo; forced backend requests degrade gracefully when dependencies are missing. For robot models, custom URDF loading is attempted first, then default fallback (`husky/husky.urdf`, then `r2d2.urdf`) to preserve run continuity. In PyBullet runs, a waypoint follower (`MobileRobotController`) applies heading-aware speed control, differential-drive wheel commands when available, and bounded command slew rates.

E. Benchmark Protocol and Ranking

Let $\mathcal{P}$ denote the planner set and $\mathcal{M}$ the generated mazes. The benchmark executes every $p \in \mathcal{P}$ on every $m \in \mathcal{M}$, rotating planner order per maze to reduce warm-start/caching bias. Each trial records success, solve time (ms), path length, expansions, and error text. This structure aligns with recent benchmarking toolchains that separate dataset/task generation, planner execution normalization, and metric aggregation [13], [14], [15].

To avoid unfair timing comparisons when planners fail on different mazes, the method computes a shared-success subset:

$$\mathcal{M}_{\text{shared}} = \{m \in \mathcal{M} \mid \forall p \in \mathcal{P},\ p \text{ succeeds on } m\}.$$

Ranking follows the implemented lexicographic policy in `benchmark.py`:

$$\text{rank}(p) \sim \big(-\text{SR}_p,\ T_p^{\text{shared}},\ E_p,\ T_p,\ \text{name}_p\big),$$

where SR is success rate, T^{shared} comparable solve time, E mean expansions, and T mean solve time. Planner name is used only as a deterministic final tie-break. Comparable path length L^{shared} is still reported as a descriptive metric, but excluded from ranking when any-angle planners are present to avoid mixed-geometry bias.

This protocol outputs `benchmark_results.csv` and `benchmark_summary.md`, enabling traceable comparison and direct linkage between manuscript claims and generated artifacts.

IV. Experimental Setup

This section documents (i) experiments already executed in the current repository snapshot and (ii) planned studies that are intentionally out of scope for the current results.

A. Executed Experiments

System and workflow context (executed). Figure 1 summarizes the deterministic runtime and artifact path used for the reported experiments, from configuration parsing and planner execution through benchmark export. Manuscript-process coordination artifacts are tracked in `coordination/` files and are not treated as primary scientific evidence in the main experimental narrative.

Planner benchmark (executed). We evaluate planners with the repository benchmark harness (`robotics_maze/src/benchmark.py`). The harness generates mazes with fixed settings (50 mazes, 15 × 15 cells, `backtracker`, base seed 7, per-maze seed = 7 + `maze_index`) and, by default, fail-closed enforces the canonical 12-planner evaluation set used in this paper. In the committed benchmark artifacts (`robotics_maze/results/benchmark_summary.md` and `robotics_maze/results/benchmark_results.csv`), this corresponds to 12 planners: `astar`, `dijkstra`, `greedy_best_first`, `r1_weighted_astar`, `r2_bidirectional_astar`, `r3_theta_star`, `r4_idastar`, `r5_jump_point_search`, `r6_lpa_star`, `r7_beam_search`, `r8_fringe_search`, and `r9_bidirectional_bfs`. The baseline BFS implementation exists in the planner registry but is intentionally excluded from the benchmark harness default discovered set; all reported rankings and tables use this 12-planner execution set for consistency with committed artifacts.

Per trial, the harness records success, wall-clock solve time (ms), path length, and node expansions. Reported paths are validated against occupancy and bounds constraints, including segment-level collision checks; a trial is marked successful only if both planner output and path validation succeed. Solve time is measured using Python `time.perf_counter()` around each planner invocation and converted to milliseconds. To reduce first-run cache bias, planner execution order is rotated per maze. In the current snapshot, each planner-maze pair is executed once, so timing differences should be interpreted as descriptive. Aggregated ranking follows the executable lexicographic order in `benchmark.py`: success rate, comparable shared-success solve time, mean expansions, overall mean solve time, then planner name as deterministic tie-break. Comparable path length is reported but excluded from ranking because any-angle and cardinal-grid path geometries are not directly commensurate.

Deterministic simulation regression (executed). In addition to planner benchmarking, the repository executes deterministic simulation checks (`robotics_maze/testing/run_sim_tests.sh`) for three representative planners (`astar`, `weighted_astar`, `fringe_search`) over three episodes (`maze-size`=11, `seed`=42). The same test run invokes the deterministic screenshot pipeline (`robotics_maze/scripts/capture_regression_screensho` which enforces six expected PNG outputs (three MuJoCo and three fallback renderer images). The latest committed test log reports all deterministic runs and screenshot checks as PASS.

Table II separates executed protocol components (used for all current claims) from planned, not-yet-executed studies.

B. Planned Future Experiments (Not Yet Executed)

The following experiments are planned and are **not** included in the current result tables:

1) **Cross-algorithm robustness sweep:** expand benchmark runs across `backtracker` and `prim` generators, larger mazes, and broader seed sets to quantify ranking stability.
2) **Dynamic-environment stress tests:** introduce moving obstacles and replanning triggers to evaluate incremental planners under online disturbances.
3) **Post-2021 planner integration study:** add learned local heuristics, Hybrid-A*, and uncertainty-aware planning baselines from the repository shortlist, then rerun the same benchmark protocol for direct comparability.
4) **Backend sensitivity analysis:** run matched scenarios with explicit `pybullet` and `mujoco` settings to isolate simulator-dependent effects on runtime and path execution behavior.

No quantitative claims from these planned experiments are used in the present manuscript version.

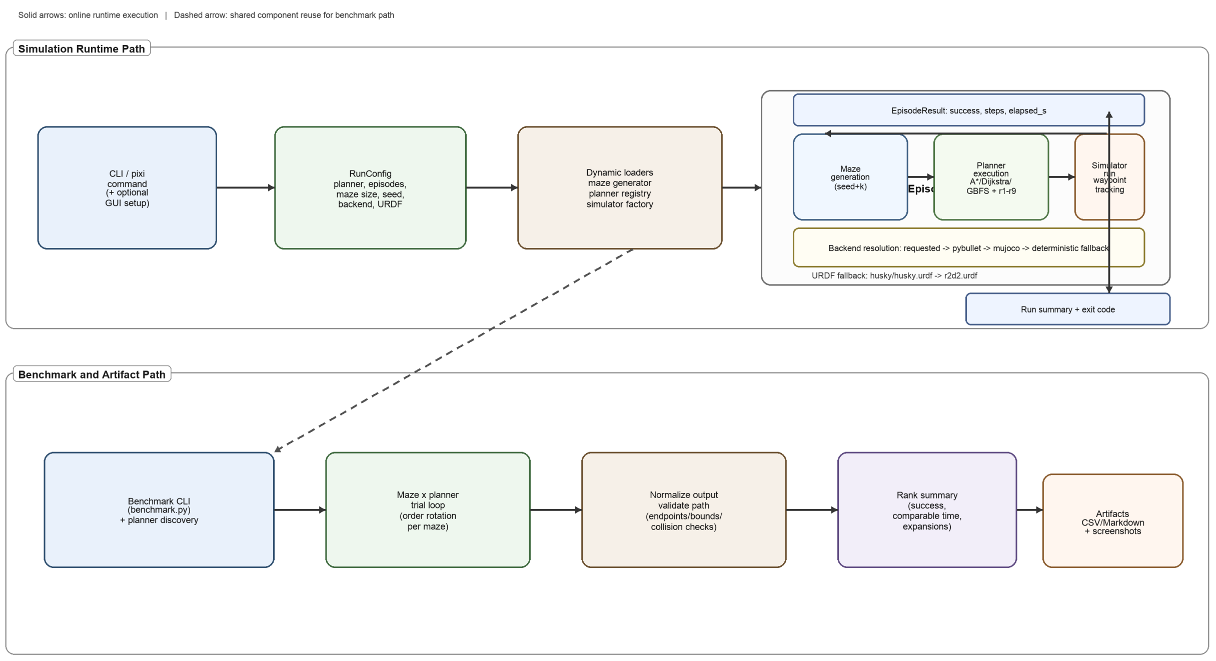
Solid arrows: online runtime execution | Dashed arrow: shared component reuse for benchmark path
Simulation Runtime Path
CLI / pixi command (+ optional GUI setup)
RunConfig planner, episodes, maze size, seed, backend, URDF
Dynamic loaders maze generator planner registry simulator factory
EpisodeResult: success, steps, elapsed_s
Maze generation (seed+k)
Episc
Planner execution A*/Dijkstra/ GBFS + r1-r9
Simulator run waypoint tracking
Backend resolution: requested -> pybullet -> mujoco -> deterministic fallback
URDF fallback: husky/husky.urdf -> r2d2.urdf
Run summary + exit code
Benchmark and Artifact Path
Benchmark CLI (benchmark.py) + planner discovery
Maze x planner trial loop (order rotation per maze)
Normalize output validate path (endpoints/bounds/ collision checks)
Rank summary (success, comparable time, expansions)
Artifacts CSV/Markdown + screenshots

Fig. 1. Deterministic system pipeline used in the executed experiments, including planner benchmarking and artifact emission. The runtime path captures backend fallback behavior (`pybullet` → `mujoco` → deterministic fallback) and the benchmark output branch used for reproducible reporting.

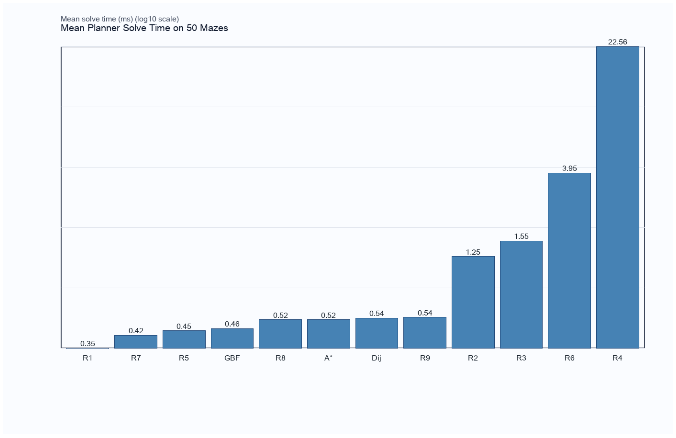

Fig. 2. Mean planner solve time (ms) on a logarithmic scale over 50 benchmark mazes. Lower values indicate faster planning.

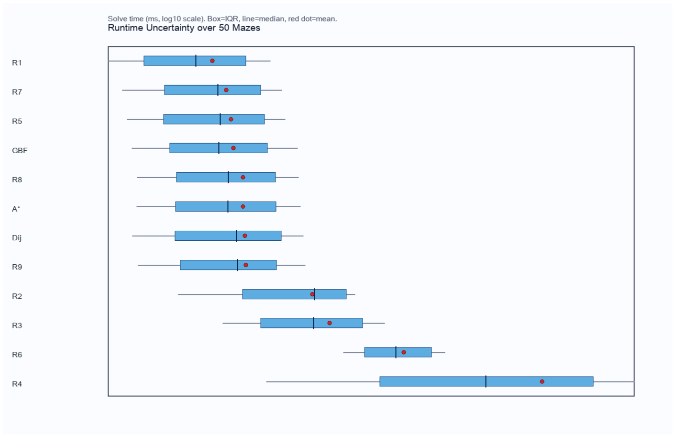

Fig. 3. Runtime uncertainty over the same 50 paired mazes. Horizontal box summaries are shown on a logarithmic scale (box: IQR, center line: median, red dot: mean).

V. Results

Table III summarizes planner performance on 50 generated 15×15 backtracker mazes (seeds 7–56). All 12 planners solved all trials (600/600) with no runtime errors, so differences here primarily reflect computational efficiency rather than reachability.

Figures 2, 3, 4, and 5 provide complementary visual summaries of runtime, runtime uncertainty, search effort, and success rate using the same benchmark snapshot as Table III.

a) Overall ranking and runtime spread.: `r1_weighted_astar` is fastest in mean solve time (0.35 ms), followed by `r7_beam_search` (0.42 ms). A second tier—`r5_jump_point_search`, `greedy_best_first`, `r8_fringe_search`, `astar`, `dijkstra`, and `r9_bidirectional_bfs`—is tightly clustered between 0.45 and 0.54 ms (at most +0.19 ms versus the top row). At maze level, `r1_weighted_astar` is fastest on 48/50 mazes, but the best-vs-second-best margin is small (median 0.049 ms; 43/50 mazes within 0.1 ms),

TABLE II
REPRODUCIBILITY PROTOCOL, EXPLICITLY SEPARATED INTO EXECUTED VS PLANNED STUDIES.

Component	Executed protocol (current artifacts)	Planned protocol (not yet executed)
Entrypoints	`python robotics_maze/src/benchmark.py` for planner benchmarks; `bash robotics_maze/testing/run_sim_tests.sh` for deterministic simulation and screenshot regression.	Add CI jobs that run full benchmark and regression protocols on pinned hardware classes and publish signed artifacts per run.
Benchmark workload	50 mazes, 15×15, `backtracker`, base seed 7, maze seed = 7+ maze index. Current snapshot evaluates 12 planners.	Scale to larger mazes and multi-generator sweeps (`backtracker` + `prim`) with expanded seed ranges to evaluate ranking stability.
Simulation regression workload	3 planners (`astar`, `weighted_astar`, `fringe_search`), 3 episodes each, `maze-size`=11, `seed`=42, backend preference `auto`.	Matched paired runs with forced `pybullet` and `mujoco`; additional stress scenarios with dynamic obstacles and re-planning triggers.
Primary metrics	Per trial: success, solve_time_ms, path_length, expansions, and error text on failure. Summary ranking prioritizes success, then comparable shared-success solve time, mean expansions, and overall mean solve time (planner name as deterministic tie-break). Path length is reported descriptively.	Add variance statistics (median, p95), per-backend deltas, and disturbance-recovery metrics for dynamic scenes.
Validity checks	Path-level validation enforces endpoint consistency, in-bounds traversal, and collision-free segments; benchmark marks success only when planner and validator agree.	Add robustness checks for near-collision margins, execution drift under dynamics, and failure-mode taxonomy by planner family.
Visual regression	Deterministic screenshot pipeline requires six fixed filenames (3 MuJoCo + 3 fallback). Latest committed run reports PASS on all checks.	Add pixel-diff thresholds and automated anomaly triage reports in CI for each benchmark commit.

TABLE III
MAIN BENCHMARK RESULTS ON 50 GENERATED 15×15 BACKTRACKER MAZES. ROWS ARE RANKED BY SUCCESS RATE (DESCENDING), THEN COMPARABLE MEAN SOLVE TIME ON SHARED-SUCCESS MAZES, MEAN EXPANSIONS, AND OVERALL MEAN SOLVE TIME (ASCENDING), WITH PLANNER NAME AS DETERMINISTIC TIE-BREAK. TIME IS REPORTED AS MEAN $\pm$ STANDARD DEVIATION OVER MAZES; MEDIAN AND INTERQUARTILE RANGE (IQR) ARE INCLUDED TO EXPOSE SKEW. LOWER IS BETTER FOR TIME, PATH LENGTH, AND EXPANSIONS.

Rank	Planner	Success	Time (ms) $\downarrow$	Median [IQR] (ms) $\downarrow$	Path Length $\downarrow$	Expansions $\downarrow$
1	R1 Weighted A*	50/50	0.35 ± 0.21	0.29 [0.15, 0.54]	142.72	187.16
2	R7 Beam Search	50/50	0.42 ± 0.24	0.38 [0.20, 0.65]	142.72	198.36
3	R5 Jump Point Search	50/50	0.45 ± 0.26	0.39 [0.19, 0.69]	142.72	57.26
4	Greedy Best-First	50/50	0.46 ± 0.27	0.38 [0.21, 0.71]	142.72	171.96
5	R8 Fringe Search	50/50	0.52 ± 0.31	0.43 [0.23, 0.79]	142.72	190.30
6	A*	50/50	0.52 ± 0.31	0.43 [0.22, 0.79]	142.72	189.06
7	Dijkstra	50/50	0.54 ± 0.32	0.48 [0.22, 0.84]	142.72	200.52
8	R9 Bidirectional BFS	50/50	0.54 ± 0.31	0.49 [0.24, 0.80]	142.72	201.52
9	R2 Bidirectional A*	50/50	1.25 ± 0.69	1.28 [0.52, 1.92]	142.72	468.02
10	R3 Theta*	50/50	1.55 ± 0.92	1.27 [0.65, 2.35]	$97.96^{\dagger}$	189.18
11	R6 LPA*	50/50	3.95 ± 1.59	3.56 [2.41, 5.59]	142.72	295.10
12	R4 IDA*	50/50	22.56 ± 22.13	11.05 [2.93, 43.00]	142.72	7061.34

†Theta* uses any-angle motion, so path-length values are not directly comparable to cardinal-grid planners.

indicating limited practical separation among the fastest methods in this setup. This narrow spread is visible in Figures 2 and 3.

b) Inferential runtime comparison.: To characterize runtime consistency across paired mazes, each planner was compared against `r1_weighted_astar` using exact two-sided paired sign tests with Holm correction (family-wise $\alpha = 0.05$). Effect sizes are reported as paired median runtime deltas (Δ = comparator − `r1_weighted_astar`, ms) with 95% bootstrap confidence intervals from 40,000 paired resamples. For the closest comparator (`r7_beam_search`), the median paired delta is 0.068 ms (95% CI [0.046, 0.085]) with `r1_weighted_astar` faster on 50/50 mazes; this indicates a consistent but small absolute gain in this dataset. Larger separations appear for slower planners (e.g., `r6_lpa_star`: 3.30 ms [2.73, 4.27]; `r4_idastar`: 10.76 ms [5.70, 30.03]). Because each planner-maze pair is measured once, these inferential statistics should be interpreted as exploratory consistency indicators, not hardware-controlled latency certification.

c) Search-effort trade-offs.: `r5_jump_point_search` attains the lowest expansion count (57.26) while remaining in the fast runtime cluster (0.45 ms), reducing expansions by roughly 69.7% relative to baseline `astar` (189.06). `r2_bidirectional_astar` and `r3_theta_star` are slower (1.25 and 1.55 ms). `r6_lpa_star` and `r4_idastar` are clear outliers at 3.95 ms and 22.56 ms, with `r4_idastar` also showing the highest variability (std 22.13 ms; IQR 2.93–43.00 ms) and expansion count (7061.34). The expansion profile in Figure 4 highlights this separation between the efficient frontier (`r5_jump_point_search`) and high-overhead outliers.

TABLE IV

Exploratory paired runtime comparisons against `r1_weighted_astar` on the same 50 mazes (single run per planner-maze pair). Positive Δ means the comparator is slower. Confidence intervals are percentile bootstrap intervals from 40,000 paired resamples (fixed seed). p-values are exact two-sided paired sign tests with Holm correction across 11 comparisons.

Comparator	Median Δ (ms)	95% CI for Δ (ms)	Slower/Faster (of 50)	Holm-adjusted p
R7 Beam Search	0.068	[0.046, 0.085]	50/0	1.95×10^{-14}
R5 Jump Point Search	0.081	[0.055, 0.124]	50/0	1.95×10^{-14}
Greedy Best-First	0.078	[0.056, 0.112]	48/2	2.27×10^{-12}
R8 Fringe Search	0.154	[0.113, 0.228]	50/0	1.95×10^{-14}
A*	0.149	[0.097, 0.230]	50/0	1.95×10^{-14}
Dijkstra	0.172	[0.106, 0.244]	50/0	1.95×10^{-14}
R9 Bidirectional BFS	0.191	[0.121, 0.232]	50/0	1.95×10^{-14}
R2 Bidirectional A*	0.983	[0.593, 1.320]	50/0	1.95×10^{-14}
R3 Theta*	1.003	[0.723, 1.714]	50/0	1.95×10^{-14}
R6 LPA*	3.296	[2.726, 4.265]	50/0	1.95×10^{-14}
R4 IDA*	10.762	[5.697, 30.026]	50/0	1.95×10^{-14}

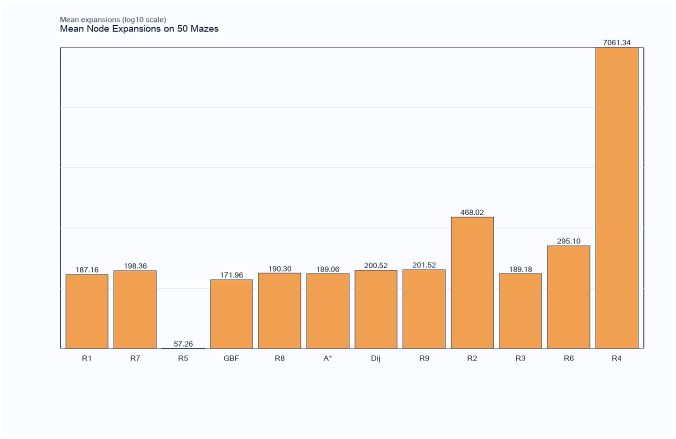

Fig. 4. Mean node expansions on a logarithmic scale for the same benchmark runs. Lower values indicate lower search effort.

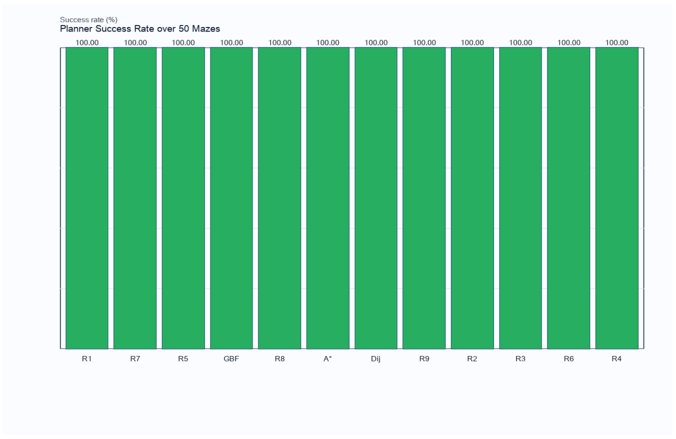

Fig. 5. Planner success rate over 50 mazes. All methods achieve 100% in this static benchmark setting.

d) Uncertainty and limitations.: These results are limited to one benchmark regime: 50 mazes, one maze size (15×15), one generator (backtracker), and one seed schedule. Timings are wall-clock and mostly sub-millisecond for the fastest methods, so statistical significance should not be conflated with large practical effect size in deployment settings. The benchmark also covers static, fully known mazes only; conclusions may not transfer to dynamic, partially observable, or larger-scale environments. Finally, Theta* uses any-angle motion, so its path-length values are not directly comparable to cardinal-grid planners. The flat success-rate profile in Figure 5 further emphasizes that this dataset mainly distinguishes planners by efficiency rather than solvability.

VI. Discussion

The current repository snapshot establishes a reproducible baseline for grid-maze navigation with measurable algorithmic tradeoffs. The benchmark artifact reports 12 planners evaluated on 50 generated 15×15 mazes, with 100% success for every planner in this static setting. Under the executable ranking policy (success rate, then comparable solve time on shared-success mazes, then mean expansions, then overall mean solve time, then planner-name tie-break), `r1_weighted_astar` is first (0.35 ms mean solve time), while `r4_idastar` is last (22.56 ms), showing that admissible but deep iterative search remains expensive for this map class.

A. What the Current Results Establish

Three descriptive findings are consistent within the present benchmark regime. First, weighted heuristic guidance is already beneficial: `r1_weighted_astar` is approximately 35% faster than baseline `astar` while keeping equivalent mean path length in this dataset. Second, frontier-pruning methods reduce search effort but do not always dominate runtime: `r5_jump_point_search` has the lowest mean expansions (57.26) yet is not the fastest due to additional bookkeeping overhead. Third, algorithm classes optimized for other settings underperform in this static benchmark; for example, `r6_lpa_star` and `r4_idastar` add overhead without gains when maps are regenerated rather than incrementally repaired. These observations are descriptive summaries from the executed benchmark snapshot and are not presented as inferential superiority claims.

Beyond planner speed, the infrastructure contributes practical value. The simulator supports explicit backend selection (`pybullet`, `mujoco`, or `auto`), URDF fallback behavior, and deterministic screenshot regression capture for visual checks. These implementation details improve reproducibility and reduce benchmark fragility across machines.

B. Implemented Versus Not-Yet-Implemented Methods

Implemented and used in the reported benchmark.

- Baseline grid planners: `astar`, `dijkstra`, `greedy_best_first`.
- Alternative planners: `r1_weighted_astar`, `r2_bidirectional_astar`, `r3_theta_star`, `r4_idastar`, `r5_jump_point_search`, `r6_lpa_star`, `r7_beam_search`, `r8_fringe_search`, and `r9_bidirectional_bfs`.
- Runtime stack support: backend auto-resolution, robust URDF fallback, and deterministic screenshot regression scripts.

Implemented in repository but not part of the current summary table.

- Additional adapter modules `r11_dijkstra`, `r12_bfs`, and `r13_greedy_best_first` exist, but the current benchmark artifact reports the baseline/alternative set above.

Not yet implemented (currently documented as future directions).

- Learned local heuristics for search (LoHA-style A* extensions).
- D* Lite + DWA and AD* + DWA hybrid global-local replanning.
- Heading-aware SE(2) Hybrid-A* / state-lattice planning and IGHA* style incremental hybrid search.
- Guided RRT* variants and uncertainty-aware MPC for dynamic, continuous navigation.

C. Limitations and Threats to Validity

The present evidence is strong for reproducible static-grid comparison, but limited for broader robotics claims.

- **Task distribution**: all reported mazes are 15×15 and generated by one algorithm family in one snapshot, so topology diversity is limited.
- **Metric saturation**: with 100% success across planners, failure robustness cannot be ranked; only efficiency metrics differentiate methods.
- **Path comparability**: any-angle methods (e.g., `r3_theta_star`) are not directly comparable to strict lattice paths without additional smoothness/feasibility metrics.
- **Physics realism**: MuJoCo fallback currently executes a simplified waypoint progression, and no dynamic-obstacle uncertainty benchmark is yet integrated.
- **Generalization**: current conclusions are for occupancy-grid planning and do not yet establish superiority in SE(2)/kinodynamic domains.

D. Future Work

The next implementation priorities are: (i) learned local heuristics with admissibility safeguards, (ii) incremental global-local replanning under dynamic disturbances, (iii) heading-aware SE(2) planners for executable trajectory quality, and (iv) uncertainty-aware continuous planners for large-map robustness. All four directions require the same acceptance discipline before being promoted into default comparisons: multi-generator and multi-seed evaluation, repeated-run timing protocols, and feasibility-oriented metrics beyond grid-path length.

These items are intentionally framed as planned work rather than current contributions. The present manuscript claims are limited to the executed static-grid benchmark infrastructure and its repository-grounded evidence.

VII. Conclusion

This manuscript stage delivers a reproducible baseline for maze-navigation planning with explicit evidence from code and benchmark artifacts. In the current snapshot, 12 implemented planners solve all 50 static 15×15 benchmark mazes, with `r1_weighted_astar` leading runtime and `r4_idastar` showing the highest computational burden. The implemented system is strongest today as a controlled occupancy-grid comparison and simulation harness (backend fallback, URDF robustness, and deterministic visual-regression support), not yet as a fully dynamic or kinodynamically realistic planner stack.

Method scope is therefore explicit: classical and alternative grid-search planners are implemented and benchmarked, while LoHA-style learned heuristics, D* Lite/AD* + DWA coupling, SE(2) Hybrid-A* or IGHA*, guided RRT*, and uncertainty-aware MPC remain not-yet-implemented research directions. The next milestone is to convert these forward hypotheses into acceptance-tested experiments with dynamic obstacles, larger maps, and feasibility-oriented metrics so future claims are supported by statistically grounded evidence rather than extrapolation.

Appendix A
Additional Details

A. Notation and Occupancy-Grid Construction

Let the maze have width W and height H in cell coordinates. The benchmark converts each maze to a binary occupancy lattice

$$G \in \{0,1\}^{R\times C}, \quad R = 2H+1, \quad C = 2W+1, \qquad (1)$$

where $G_{r,c} = 0$ denotes free space and $G_{r,c} = 1$ denotes blocked space. The cell-to-lattice map used by the implementation is

$$\phi(x,y) = (2y+1,\, 2x+1), \qquad (2)$$

so the benchmark start and goal nodes are

$$s = \phi(x_s, y_s), \qquad g = \phi(x_g, y_g). \qquad (3)$$

For a lattice node $n = (r,c)$, the neighborhood is either 4-connected or 8-connected:

$$\mathcal{N}_4(n) = \{(r-1,c),(r,c+1),(r+1,c),(r,c-1)\} \cap \Omega, \quad (4)$$

$$\mathcal{N}_8(n) = \mathcal{N}_4(n) \cup \{(r-1,c-1),(r-1,c+1),(r+1,c+1),(r+1,c-1)\} \cap \Omega, \quad (5)$$

with Ω the in-bounds lattice set. The step cost in the baseline planners is

$$c(u,v) = \begin{cases} \sqrt{2}, & \text{if } u \to v \text{ is diagonal,} \\ 1, & \text{otherwise.} \end{cases} \tag{6}$$

Heuristic options are

$$h_{\text{man}}(n,g) = |r - r_g| + |c - c_g|, \tag{7}$$

$$h_{\text{euc}}(n,g) = \sqrt{(r-r_g)^2 + (c-c_g)^2}, \tag{8}$$

$$h_{\text{cheb}}(n,g) = \max\{|r-r_g|, |c-c_g|\}. \tag{9}$$

B. Planner and Benchmark Objectives

For a path $\pi = (n_0, \dots, n_K)$ with $n_0 = s$ and $n_K = g$, the weighted path-cost objective is

$$J_{\text{cost}}(\pi) = \sum_{k=1}^{K} c(n_{k-1}, n_k). \tag{10}$$

The baseline priority keys used in the benchmark are:

$$f_{\text{A*}}(n) = g(n) + w\,h(n), \qquad w \geq 1, \tag{11}$$

$$f_{\text{Dijkstra}}(n) = g(n), \tag{12}$$

$$f_{\text{GBFS}}(n) = h(n), \tag{13}$$

and BFS minimizes hop count

$$J_{\text{hop}}(\pi) = K. \tag{14}$$

For A*, the secondary tie key is

$$\tau(n) = \begin{cases} h(n), & \text{low\_h}, \\ -g(n), & \text{high\_g}, \\ 0, & \text{fifo}, \end{cases} \tag{15}$$

and the heap key is $(f(n), \tau(n), t(n))$ where $t(n)$ is a monotonically increasing insertion counter.

Path validation and measured length: Given returned path $P = (p_0, \dots, p_K)$, validity requires:

$$p_0 = s, \quad p_K = g, \quad p_k \in \Omega, \quad G_{p_k} = 0,\ \forall k, \tag{16}$$

and for each segment, every rasterized Bresenham cell is also in-bounds and free. If $\mathcal{B}(p_{k-1}, p_k)$ denotes the rasterized segment cell sequence, the measured benchmark path length is

$$L(P) = \sum_{k=1}^{K} \left(|\mathcal{B}(p_{k-1}, p_k)| - 1\right). \tag{17}$$

Planner ranking objective: For planner p over M mazes, with success indicator $I_{p,m} \in \{0,1\}$, solve time $t_{p,m}$, measured length $L_{p,m}$, and expansions $e_{p,m}$:

$$S_p = \frac{1}{M}\sum_{m=1}^{M} I_{p,m}, \qquad \mathcal{M}_p^+ = \{m \mid I_{p,m} = 1\}. \tag{18}$$

Define the shared-success set

$$\mathcal{M}_{\text{shared}} = \{m \mid I_{q,m} = 1,\ \forall q \in \mathcal{P}\}. \tag{19}$$

The comparable means used for ranking are

$$\bar{t}_p^c = \begin{cases} \frac{1}{|\mathcal{M}_{\text{shared}}|}\sum_{m \in \mathcal{M}_{\text{shared}}} t_{p,m}, & |\mathcal{M}_{\text{shared}}| > 0, \\ \frac{1}{M}\sum_{m=1}^{M} t_{p,m}, & \text{otherwise,} \end{cases} \tag{20}$$

$$\bar{L}_p^c = \begin{cases} \frac{1}{|\mathcal{M}_{\text{shared}}|}\sum_{m \in \mathcal{M}_{\text{shared}}} L_{p,m}, & |\mathcal{M}_{\text{shared}}| > 0, \\ \frac{1}{|\mathcal{M}_p^+|}\sum_{m \in \mathcal{M}_p^+} L_{p,m}, & \text{otherwise.} \end{cases} \tag{21}$$

With

$$\bar{e}_p = \frac{1}{|\mathcal{M}_p^+|}\sum_{m \in \mathcal{M}_p^+} e_{p,m}, \qquad \bar{t}_p = \frac{1}{M}\sum_{m=1}^{M} t_{p,m}, \tag{22}$$

ranking is lexicographic on

$$\left(-S_p,\ \bar{t}_p^c,\ \bar{L}_p^c,\ \bar{e}_p,\ \bar{t}_p,\ \text{name}_p\right). \tag{23}$$

C. Local Control Abstraction

Path-to-waypoint map: The simulator converts planner outputs to world-frame waypoints. If a path is recognized as cell-grid coordinates (x_k, y_k):

$$w_k = \left((x_k + 0.5)s_c,\ (y_k + 0.5)s_c\right), \tag{24}$$

where s_c is `cell_size`. If recognized as occupancy-lattice coordinates:

$$w_k = \left(0.5\, s_c\, x_k,\ 0.5\, s_c\, y_k\right). \tag{25}$$

Cell-grid waypoints are additionally compressed by removing collinear interior points.

Waypoint tracking law: At control step t, with robot pose (x_t, y_t, ψ_t) and active waypoint (x_j, y_j):

$$d_t = \sqrt{(x_j - x_t)^2 + (y_j - y_t)^2}, \tag{26}$$

$$\theta_t = \operatorname{atan2}(y_j - y_t,\ x_j - x_t), \qquad e_{\psi,t} = \operatorname{wrap}_{[-\pi,\pi]}(\theta_t - \psi_t). \tag{27}$$

The angular command is the clipped proportional target

$$\omega_t^\star = \arg\min_{|\omega| \leq \omega_{\max}} |\omega - k_\psi e_{\psi,t}| = \operatorname{clip}(k_\psi e_{\psi,t}, -\omega_{\max}, \omega_{\max}). \tag{28}$$

The linear target is the largest feasible speed under speed, distance, and braking bounds:

$$v_t^\star = \alpha_t \cdot \max_{v \geq 0} v \quad \text{s.t.} \quad v \leq v_{\max},\quad v \leq d_t,\quad v \leq \sqrt{2 a_{\text{dec}} d_t}, \tag{29}$$

equivalently

$$v_t^\star = \alpha_t \cdot \min\left\{v_{\max},\ d_t,\ \sqrt{2 a_{\text{dec}} d_t}\right\}. \tag{30}$$

The heading gate $\alpha_t \in [0,1]$ is

$$\alpha_t = \begin{cases} 0, & |e_{\psi,t}| \geq \theta_{\text{turn}}, \\ 1, & |e_{\psi,t}| \leq \theta_{\text{drive}}, \\ 1 - \sigma\left(\dfrac{|e_{\psi,t}| - \theta_{\text{drive}}}{\theta_{\text{turn}} - \theta_{\text{drive}}}\right), & \text{otherwise}, \end{cases} \tag{31}$$

with smoothstep

$$\sigma(z) = z^2(3 - 2z), \quad z \in [0,1]. \tag{32}$$

Commands are passed through acceleration/deceleration slew limits with control step Δt:

$$u_t = u_{t-1} + \text{clip}(u_t^\star - u_{t-1}, -\Delta_u, \Delta_u), \tag{33}$$

where

$$\Delta_u = \begin{cases} a_{u,+}\Delta t, & \text{sgn}(u_t^\star) = \text{sgn}(u_{t-1}) \ \wedge \ |u_t^\star| > |u_{t-1}|, \\ a_{u,-}\Delta t, & \text{otherwise}, \end{cases} \tag{34}$$

applied separately to $u = v$ and $u = \omega$.

Then the turn-rate-dependent linear cap is enforced:

$$\eta_t = \text{clip}\left(\frac{|\omega_t|}{\omega_{\max}}, 0, 1\right), \qquad v_{\text{turn},t} = v_{\max} \max\left(0.18,\ 1 - 0.68\,\sigma(\eta_t)\right), \tag{35}$$

$$v_t \leftarrow \text{clip}(v_t, -v_{\text{turn},t}, v_{\text{turn},t}). \tag{36}$$

For differential drive with axle track b and wheel radius r:

$$v_{L,t} = v_t - \frac{b}{2}\omega_t, \qquad v_{R,t} = v_t + \frac{b}{2}\omega_t, \tag{37}$$

$$\Omega_{L,t} = v_{L,t}/r, \qquad \Omega_{R,t} = v_{R,t}/r. \tag{38}$$

Waypoint index j is advanced when $d_t \leq \varepsilon_j$, where ε_j is the per-waypoint tolerance (larger at the final waypoint).

D. Complexity Statements

Let $V = RC$ and $E \leq 8V$ for the occupancy lattice graph.

$$T_{\text{A}*/\text{Dijkstra/GBFS}} = O(E \log V), \qquad M_{\text{A}*/\text{Dijkstra/GBFS}} = O(V), \tag{39}$$

because open lists are binary heaps and closed/g-score/came-from are hash maps/sets.

$$T_{\text{BFS}} = O(V + E), \qquad M_{\text{BFS}} = O(V), \tag{40}$$

from deque frontier plus visited/parent maps.

For one returned path P:

$$T_{\text{validate}} = O\left(\sum_{k=1}^{K} |\mathcal{B}(p_{k-1}, p_k)|\right) = O(L(P)). \tag{41}$$

For M mazes and $|\mathcal{P}|$ planners:

$$T_{\text{benchmark}} = O\left(\sum_{m=1}^{M} \sum_{p \in \mathcal{P}} \left(T_{\text{plan}}(p,m) + T_{\text{validate}}(p,m)\right)\right). \tag{42}$$

Per control update, command synthesis is constant-time, and actuator dispatch is linear in wheel-joint count n_w:

$$T_{\text{control step}} = O(1 + n_w). \tag{43}$$

TABLE V
REPOSITORY-GROUNDED REPRODUCIBILITY CHECKLIST SNAPSHOT (2026-02-26).

Item	Status	Evidence
Environment specification and dependencies	complete	`pixi.toml` (workspace tasks), `robotics_maze/pixi.toml` (Python 3.11, PyBullet, MuJoCo, pytest).
CLI entrypoints and run commands	complete	`scripts/sim_runner.py` and root `pixi.toml` task definitions.
Physics backend fallback behavior	complete	`robotics_maze/src/sim.py` backend resolution and deterministic fallback branch.
Planner benchmark protocol and ranking policy	complete	`robotics_maze/src/benchmark.py`; generated `robotics_maze/results/benchma`
Benchmark trial artifacts	complete	`robotics_maze/results/benchmark_resu` and `robotics_maze/results/benchmark_s`
Regression screenshot generation	complete	`robotics_maze/scripts/capture_regres` strict expected filename checks.
Smoke and regression test logs	complete	`robotics_maze/tests/test_core.py` and `robotics_maze/testing/TEST_RUN_LC`
Reference inventory target (≥40 refs from 2021+)	complete	`references.bib` and `coordination/citations_audit.csv` contain populated, post-2021 citation records in the current snapshot.
Citation quality target (≥80% peer reviewed)	complete	`coordination/citations_audit.csv` marks peer-review status per entry and is populated beyond header-only state.
Figure manifest inventory completeness	complete	`coordination/figure_manifest.csv` lists tracked figure IDs, source paths, and reproducibility fields.
Claim-traceability freshness gate	complete	`coordination/claims_traceability.csv` refreshed to verified entries for current manuscript claims and artifact checks.

E. Reproducibility Checklist

This checklist reflects the repository snapshot date listed in the caption. Coordination artifacts are maintained as explicit evidence ledgers and updated when benchmark/code/manuscript states change.

References

[1] J. R. Sánchez-Ibáñez, C. J. Pérez-del Pulgar, and A. García-Cerezo, "Path planning for autonomous mobile robots: A review," *Sensors*, vol. 21, no. 23, p. 7898, Nov. 2021. [Online]. Available: http://dx.doi.org/10.3390/s21237898

[2] K. Karur, N. Sharma, C. Dharmatti, and J. E. Siegel, "A survey of path planning algorithms for mobile robots," *Vehicles*, vol. 3, no. 3, p. 448–468, Aug. 2021. [Online]. Available: http://dx.doi.org/10.3390/vehicles3030027

[3] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion planning and control for mobile robot navigation using machine learning: a survey," *Autonomous Robots*, vol. 46, no. 5, p. 569–597, Mar. 2022. [Online]. Available: http://dx.doi.org/10.1007/s10514-022-10039-8

[4] L. Liu, X. Wang, X. Yang, H. Liu, J. Li, and P. Wang, "Path planning techniques for mobile robots: Review and prospect," *Expert Systems with Applications*, vol. 227, p. 120254, Oct. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2023.120254

[5] C. Baumann and A. Martinoli, "A modular functional framework for the design and evaluation of multi-robot navigation," *Robotics and Autonomous Systems*, vol. 144, p. 103849, Oct. 2021. [Online]. Available: http://dx.doi.org/10.1016/J.ROBOT.2021.103849

[6] J. He, J. Zhang, J. Liu, and X. Fu, "A ros2-based framework for industrial automation systems," in *2022 2nd International Conference on Computer, Control and Robotics (ICCCR)*. IEEE, Mar. 2022, pp. 98–102. [Online]. Available: http://dx.doi.org/10.1109/ICCCR54399.2022.9790247

[7] K. Chen, M. Wang, M. Gualtieri, N. Tian, C. Juette, L. Ren, J. Ichnowski, J. Kubiatowicz, and K. Goldberg, "Fogros2-ls: A location-independent fog robotics framework for latency sensitive ros2 applications," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2024, pp. 10 581–10 587. [Online]. Available: http://dx.doi.org/10.1109/ICRA57147.2024.10610759

[8] V. Shcherbyna, L. Kastner, D. Diaz, H. G. Nguyen, M. H.-K. Schreff, T. Seeger, J. Kreutz, A. Martban, Z. Shen, H. Zeng, and H. Soh, "Arena 4.0: a comprehensive ros2 development and benchmarking platform for human-centric navigation using generative-model-based environment generation," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2025, pp. 9138–9144. [Online]. Available: http://dx.doi.org/10.1109/ICRA55743.2025.11127635

[9] J. A. Abdulsaheb and D. J. Kadhim, "Classical and heuristic approaches for mobile robot path planning: A survey," *Robotics*, vol. 12, no. 4, p. 93, Jun. 2023. [Online]. Available: http://dx.doi.org/10.3390/robotics12040093

[10] Y. Hu, S. Wang, Y. Xie, S. Zheng, P. Shi, I. Rudas, and X. Cheng, "Deep reinforcement learning-based mapless navigation for mobile robot in unknown environment with local optima," *IEEE Robotics and Automation Letters*, vol. 10, no. 1, p. 628–635, Jan. 2025. [Online]. Available: http://dx.doi.org/10.1109/LRA.2024.3511437

[11] H. S. Hewawasam, M. Y. Ibrahim, and G. K. Appuhamillage, "Past, present and future of path-planning algorithms for mobile robot navigation in dynamic environments," *IEEE Open Journal of the Industrial Electronics Society*, vol. 3, p. 353–365, 2022. [Online]. Available: http://dx.doi.org/10.1109/OJIES.2022.3179617

[12] A. Loganathan and N. S. Ahmad, "A systematic review on recent advances in autonomous mobile robot navigation," *Engineering Science and Technology, an International Journal*, vol. 40, p. 101343, Apr. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.jestch.2023.101343

[13] E. Heiden, L. Palmieri, L. Bruns, K. O. Arras, G. S. Sukhatme, and S. Koenig, "Bench-mr: A motion planning benchmark for wheeled mobile robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4536–4543, Jul. 2021. [Online]. Available: http://dx.doi.org/10.1109/LRA.2021.3068913

[14] C. Chamzas, C. Quintero-Pena, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, "Motionbenchmaker: A tool to generate and benchmark motion planning datasets," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 882–889, Apr. 2022. [Online]. Available: http://dx.doi.org/10.1109/LRA.2021.3133603

[15] M. Mayer, J. Kulz, and M. Althoff, "Cobra: A composable benchmark for robotics applications," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2024, pp. 17 665–17 671. [Online]. Available: http://dx.doi.org/10.1109/ICRA57147.2024.10610776