A Benchmark for Multi-Robot Planning in Realistic, Complex and Cluttered Environments

Simon Schaefer[1], Luigi Palmieri[2], Lukas Heuer[2],
Ruediger Dillmann[1], Sven Koenig[3], Alexander Kleiner[2]

***Abstract*— Several successful approaches exist for solving the complex problem of multi-robot planning and coordination. Due to the lack of adequate benchmarking tools, comparing these approaches and judging their suitability for use in realistic scenarios is currently difficult. Therefore, we propose an open-source benchmark suite that aims to close this gap. Unlike existing benchmarks, our approach uses full-stack multi-robot navigation systems in realistic 3D simulated environments from the intralogistic and household domains. Using the open-source frameworks ROS 2, Gazebo and RMF allows the user to add other robot platforms easily.**

The framework provides easy-to-use abstractions, typical metrics and interfaces to several established planning libraries for multi-robot systems. With all these features, our framework successfully aids practitioners and researchers in comparing multi-robot planning and coordination systems to the state of the art. Our experiments show how the proposed benchmark simplifies gaining insights on relevant close to real-life robotics use cases.

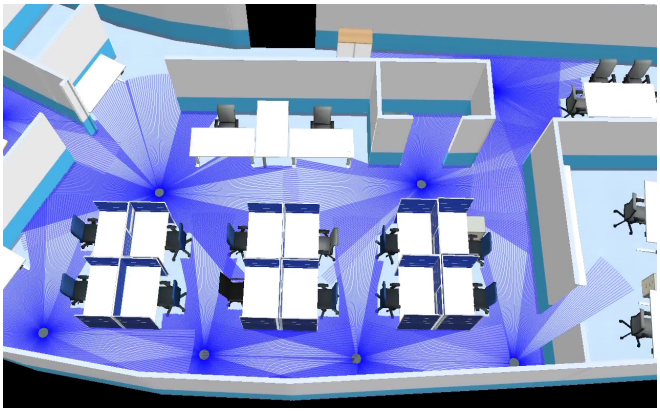

Fig. 1: Nine robots (some behind walls) moving in the office environment of the proposed benchmark. The individual laser scanners rays are shown in blue.

I. Introduction

Planning for and controlling a fleet of autonomous robots is a challenging task, especially in cluttered and dynamic environments. Such systems are controlled by a complex pipeline of components ranging from centralized path finders to local distributed controllers, each having their own limitations. For instance, optimal Multi-Agent Path Finding (MAPF) and generalized task assignment are known to be NP-hard [16] [36], even in static environments. Many suboptimal but faster algorithms have been proposed, but they are hard to compare and selecting the best one for a real-world application is not a trivial task. This is particularly true for fleets of robotic systems navigating in uncertain and dynamic environments.

Several benchmarks have been presented recently [8], [23], [33], [35]. However, they consider only a part of the pipeline for fleets of robots, with most of them focusing on path planning. With the goal of enabling practitioners and researchers to select the algorithms best suited for their robotic fleets, we propose a multi-robot planning and coordination benchmark. Our benchmark considers different planning and control layers, like centralized and decentralized MAPF algorithms as well as robot-navigation systems composed of global and local planners, and evaluates them in a physics-based simulation of service-robot-oriented environments. Its modular structure and versatile interfaces facilitate its extension with additional scenarios, algorithms and functionalities.

We make two contributions:
i) We propose `MRP-Bench`, an open-source benchmark (available at `https://github.com/boschresearch/mrp_bench`) for multi-robot planning and coordination, consisting of a set of environments, metrics and interfaces to available state-of-the-art planners and navigation systems. The approach is easy to use and extend.
ii) We show the benefit of using the developed benchmark by running a set of experiments considering state-of-the-art planning baselines in intralogistic and household robotic settings. We compare fully decentralized approaches (based on A*) against different variants of centralized conflict-based search algorithms (namely, CBS, ECBS and EECBS). Additionally, we study task assignment and compare different approaches (some considering task assignment coupled with path finding). Our experiments show that optimal CBS often fails to find a solution within a reasonable amount of time and suboptimal algorithms are viable alternatives. In fact, thanks to local collision resolution, even a completely decentralized approach can be considered.

[1]S. Schaefer and R. Dillmann are with the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany {mrp@simon-schaefer.net, ruediger.dillmann@kit.edu}
[2]L. Palmieri, L. Heuer, A. Kleiner are with Robert Bosch GmbH, Corporate Research, Stuttgart, Germany {luigi.palmieri, lukas.heuer, alexandre.kleiner}@de.bosch.com.
[3]S. Koenig is with the Computer Science Department of the University of Southern California {skoenig@usc.edu}
This work was partly supported by the EU Horizon 2020 research and innovation program under grant agreement No. 101017274 (DARKO).

II. Related Work

Benchmarking planning algorithms has received a lot of attention in the last years [5], [8], [11], [17], [23], [30], [33], [35]. However, few of them focus on multi-robot planning and coordination [8], [23], [33], [35]. Stern et al. [33] discuss the *Grid-Based MAPF* benchmark from MovingAI [34], [35], which provides 2D occupancy grid-based maps with various scenarios (consisting of tuples of starts and goal cells). This benchmark assumes perfect knowledge of the world, while our approach considers not only planning but also execution in realistic simulated environments (considering uncertainty). Asprilo [8] also offers a simulation environment, enabling the user to check and visualize the results. This framework is aimed specifically at intralogistic warehouse scenarios. The world is represented by a 2D occupancy grid. Specifics of intralogistics are also modelled, including shelves of items which need to be brought to picking stations. Therefore, agents can perform additional actions, such as picking up and setting down a shelf. While the constraints can be defined using answer set programming, the robot motion model is rather simple and the focus is on abstract representations of agents. Different from our approach, this approach neglects the higher-level control aspects (e.g. collision avoidance). Flatland [2], [23] focuses on the benchmarking of vehicle rescheduling problem and is not well suited for broader robotics domains. The environments are 2D grids with some restrictions on the transitions between cells: for example, there is no type of cell that can be entered and exited from all directions, as one would expect for most household or intralogistics robots. Contrary to Flatland, our approach considers more realistic robotic scenarios in terms of environment representation and robot models.

Several approaches exist for machine learning based solutions [27], [31] that are also able to train reinforcement learning agents. Differently from our benchmark, they do not consider a realistic multi-robot navigation system.

Moreover, our approach, different from all the others, aims to reduce the gap between simulation and real-world operation by making use of state-of-the-art robotic frameworks (namely, ROS 2 Galactic [22] with Gazebo [18], Navigation 2 (Nav2) [21] and the Robotics Middleware Framework (RMF) [24]). Different from [6], we do not limit ourselves to simulations but also provide metrics, scenarios and reports generation tools.

III. Background

In our benchmark we consider the following two planning problems: task assignment and path finding. Importantly, both problems are tightly coupled and can be solved separately or combined [13]. While hereinafter we focus on navigation tasks, the framework can be extended to handle different type of tasks, e.g. such as pick and delivery [20].

A. Task Assignment

We define our task assignment problem as follows. We are given a set of n Agents $A = \{a_1, ..., a_n\}$ and a set of n Goals $G = \{g_1, ..., g_n\}$. In a first step, an $n \times n$ matrix $M = [m_{ij}]$ is created that represents the estimated costs for each agent a_i to reach each goal g_j. Different heuristics are possible here, such as the Euclidean distance, the Manhattan distance or the path length calculated by A* between an agent and a goal. Following [29, p. 1f.], we model the problem using a bipartite graph $G = (A; G, E)$. The edges in E connect vertices in A and G. Each edge is given a weight that relates to the estimated cost between the respective agent and goal. Then, an optimization problem is solved to obtain the matching of A and G that minimizes the total weight.

B. Path Finding

Stern et al. [33] attempt to unify the terminology used in describing MAPF problems and the evaluation metrics. The authors first define what they call the *classical MAPF problem*. In their definition, the environment is described by an undirected graph. The problem consists of a set of agents, each of which is located at a start vertex and needs to move to a goal vertex. Agents can wait at their current vertex or move from their current vertex to an adjacent one. Both possibilities constitute an action. Different kinds of conflicts are defined to specify the constraints imposed on the agent. The simplest ones are the edge and vertex conflict, where no two agents may occupy the same space at the same time.

Following the definitions in [33], we represent the world as a binary occupancy grid W and use the bijection SG to represent our MAPF problem. This results in an $w \times h$ matrix, where each entry denotes a square cell c of size d in the physical world. The value of each cell denotes whether that cell is traversable. The path-finding algorithm generates a set of schedules (i.e. paths) $S = S_{a_1}, ..., S_{a_n}$, with $S_{a_i} = \{< c_1, t_1 >, < c_2, t_2 >, ..., < c_k, t_k >\}$ for each agent $a_i \in A, i = 1, \ldots, n$, that represents the cell that a_i occupies at each discrete time step in the simulation. The schedule must adhere to the constraint that no two agents ever occupy the same cell at the same time step. In addition, all cells ever occupied by agents must have an occupancy value of 0 in W, and, in the last step of each schedule, each agent a_i has to occupy its goal cell g_i. This problem can be solved by planning algorithms, optimizing for different metrics. Often, either total *cost* or the *makespan* are used. We use the common definition of individual cost for each agent as the number of cells traversed. Consequently, the total cost is the sum of individual costs. The makespan is defined as the highest individual cost.

IV. MRP-Bench Architecture

In this section, we explain the key decisions in designing the benchmark suite and outline its architecture. We describe the frameworks on which the benchmark is based, then present the algorithms for which we provide interfaces, and lastly present the proposed software architecture.

A. Background Frameworks

In our benchmark, we adopt the most common state-of-the-art frameworks used by the robotics community for robot control and navigation, namely:

i) `ROS 2`, short for Robot Operating System 2, is an open-source middleware for robotics applications using a publisher-subscriber architecture [7]. The last version of ROS

1, ROS Noetic, was released in 2020. While ROS 1 will still be supported for a few years, we base this benchmark on ROS 2 as the more advanced and future-proof version.

ii) `Gazebo` is the default 3D robot simulation environment for ROS [7]. Version 11, which we use, is the last major release of Gazebo [26].

iii) `Nav2` is a framework for mobile robot navigation [1]. It can compute paths and interpolate between waypoints, build a local costmap, and attempt recovery in case a robot is stuck. Nav2 is becoming the standard regarding robot navigation for modern systems; suggesting that the results obtained from the benchmark might be easy to replicate in real-world settings. Nav2 implements different heap for recovery and collision avoidance, both important tools for reducing conflicts during execution.

iv) `RMF` (Robotics Middleware Framework) is "a collection of reusable, scalable libraries and tools building on top of ROS 2 that enable the interoperability of heterogeneous fleets of any type of robotic systems" [25]. We use two components of RMF. The first component is the fleet adapter. The benchmark interacts with the interfaces provided by the fleet adapter to obtain the current status of the simulated robots and to sending control commands. A standardized interface is the basis for integrating different types of robots into the simulation. The second component is the demo environments and *Traffic Editor* of RMF, which is a GUI-based tool for creating benchmark scenarios quickly.

All robot models that run within Gazebo, ROS 2 and Nav2 can be used in our benchmark. Our initial experiments use the *TurtleBot3* as it is a small robot with a simple 2D LiDAR that is available at a reasonably low cost [3] [15].

B. Algorithms

As our work focuses on benchmarking, we selected an initial set of state-of-the-art algorithms based on the availability of proper documentation and open-source implementations. Notably, the ROS 2 interfaces allow for an easy integration of other solvers and additional algorithms. For our benchmark and the experiments in Section VI, we adopt the following tools:

i) OR-Tools [9] is an open software suite for optimization, maintained by Google [28]. Besides tasks such as routing and scheduling, it can also be used to solve the task-assignment problem. For this task, either a *Mixed-Integer Programming* or a boolean satisfiability problem (*SAT*) solver can be used.

ii) From libMultiRobotPlanning [12], we adopt the following algorithms: decentralized A* [10], Conflict-Based Search (CBS) [32], Enhanced Conflict-Based Search (ECBS) [4], CBS-TA [13] and EBCS-TA [13]. A* is not a multi-robot algorithm. However, we turned it into a very simple baseline by running A* for each agent and combining the resulting paths into a schedule (which is currently a standard approach used in industry). The optimal algorithm CBS works on two levels. On the low level, it performs path planning using A* for each agent and checks for conflicts. A conflict is defined as two agents occupying the same vertex at the same time. If such a conflict occurs, a new constraint that prevents this particular conflict is added to the lower-level search. Given a conflict with two agents, the constraint could be added to either of them. To ensure optimality, both possibilities are saved. The hierarchy of constraints is stored in a tree representation. The higher level search resolves the tree of conflicts until the optimal solution has been determined. ECBS is an extension to CBS that allows for suboptimality on both planning levels: the user can provide a suboptimality factor w, and the cost of the returned solution is guaranteed to be smaller than or equal to $w \cdot optimalCost$. This is achieved by using a bounded-suboptimal variant of A* for single-agent searches. CBS-TA combines CBS with task assignment. However, with N robots and goals to match, the number of possible assignments is $N!$. Checking $N!$ assignments quickly becomes impractical. CBS-TA therefore starts with the best assignment in theory, i.e. the assignment where the sum of individual shortest paths is minimized. If conflicts occur, the next-best assignment is also taken into consideration, besides introducing constraints as in plain CBS. This means CBS-TA creates a search forest instead of just a search tree. For EBCS-TA, the changes compared to plain ECBS are analogous to CBS and CBS-TA, respectively.

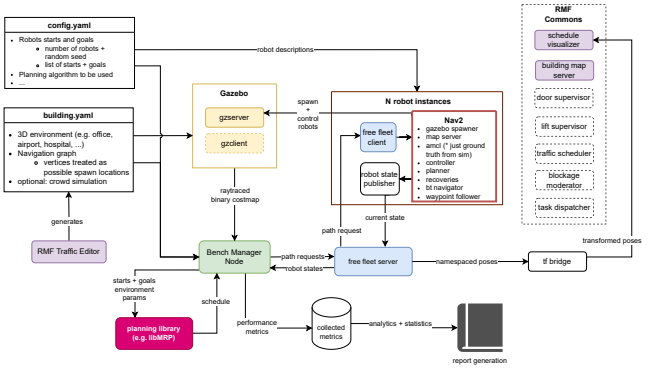

Fig. 2: A flow-chart of the proposed benchmarking architecture.

C. Software Architecture

Figure 2 provides an overview of `MRP-Bench`. The framework consists of the following sub-components: a traffic editor for defining the environment, a configuration file for settling the problem to solve, the simulator, the manager node for orchestrating the benchmarking phase, the local navigation systems and the evaluation sub-system (see Section V).

1) Starting Up: The workflow starts at the **RMF Traffic Editor** which can be used to generate a **Gazebo** world file with the intermediate step of a *building.yaml* description. Together with the *config.yaml*, this provides the necessary information for the **Bench Manager Node** to start the benchmark: e.g. number of robots, random seed, start and goals. Using the world file, Gazebo launches the simulated 3D environment. From this simulation, a binary costmap is

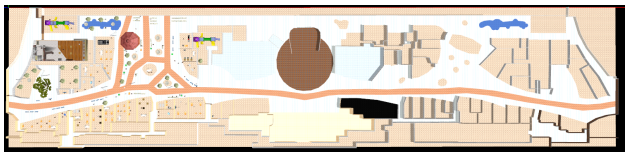

Fig. 3: The airport scenario is quite large and it offers the possibility to test the algorithms considering large automated ground vehicles.

obtained using raytracing. The *building.yaml* also contains a representation of the navigation graph, which denotes the lanes that robots may move in. From this navigation graph, which includes the spatial position of each vertex, another, simpler occupancy grid is created. This time, only the lanes of the navigation graph are marked as passable. This occupancy grid serves as an input for the path planning algorithm. Some algorithms can also directly use the navigation graph. In this case, no conversion to the occupancy grid is necessary. Additional RMF tasks can be activated (e.g. lift and doors management) to increase the complexity of the simulation.

2) Planning and Fleet Management: If the planning library has managed to create a schedule, the Bench Manager computes and saves performance metrics from the planning, converts the schedule into separate path requests for each agent, and continues with sending the path requests to the fleet server, which delivers them to the individual fleet client. The path is formed by several waypoints that will be then given to the local navigation units.

3) Local Navigation: Together with a state publisher and the fleet client, a full Nav2 stack is spawned for each robot. We use the standard global and local planning algorithms provided by the main repository. The benchmark user is free to choose the most interesting planners (i.e. local and global) for their scenarios. The Nav2 stack interpolates a local path between the waypoints of the provided high level path, controls the robots and in case of conflicts performs collision avoidance and local recovery. Ground truth position from the simulator can be used or the user can decide to run a SLAM algorithm. Currently, the benchmark operates under the assumption that robots progress from cell to cell with the same average speed.

4) Data Visualization and Collection: Robot poses are displayed on a map using the **RMF schedule visualizer**. While the agents are following their schedule, their states (e.g. poses and velocities) are recorded and can be analyzed later to gather additional metrics. Additionally the users can record more data in rosbag format.

All custom, self-written nodes are implemented in Python3. The architecture is heavily based on the ROS 2 launch system.

V. Evaluation Sub-System

In this section we detail the scenarios and the metrics included in the benchmarking suite. Those can be further extended by the user.

A. Scenarios

We provide three different types of environments, that stress different properties and capabilities of the multi-robot system, namely: the *airport*, the *office* and *warehouse*, see Figures 3 and 4. All those environments have different properties (see Table I). *Warehouse* and *airport* are the larger ones that require more planning efforts (both due to their sizes and layouts with high traffic *main roads*). *Warehouse* and *office* are the most cluttered ones. Other environments can be easily added.

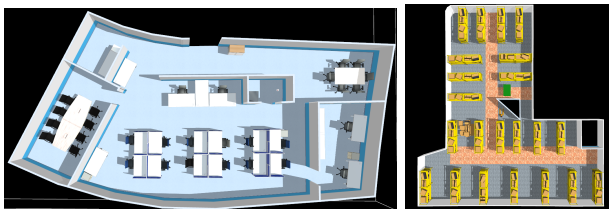

Fig. 4: **Left:** The office environment provides different homotopy classes and cluttered spaces. **Right:** The warehouse environment has been designed considering classical situations for robots working in intralogistic settings.

Property	Office	Warehouse	Airport Terminal
Width	21.53 m	22.16 m	282.22 m
Height	12.05 m	27.07 m	64.35 m
Nav. graph			
Vertices	29	54	210
Edges	32	59	211
Occupancy grid			
Total cells	1025	3009	105700
Free cells	333	788	7645
Blocked cells	692	2221	98055

TABLE I: Environments key facts. Grid statistics at 0.4 m grid resolution and two-way roads/edges.

B. Metrics

We adopt two common sets of metrics [33]. The first one includes planning performance and quality metrics: *success rate*, *planning time*, *makespan* and *cost*.

The second set is calculated by analyzing recorded data of the execution of the scenarios, namely:

i) Execution time: Optimally, the scenario finished when all agents have arrived at their assigned goals. However, a timeout for the execution can be provided via the config.

ii) Number of goals reached: The number of agents that reach their goal within the timeout.

iii) Minimum distance between two agents: At any point in time, we check the respective distances between all controlled agents. Accordingly, we can see whether some agents ran into each other, and if not, how close they got.

iv) Time blocked per agent and total: An agent is considered *blocked* if it has not progressed by at least one cell width within a certain time. For each agent, this metric calculates the amount of time spent blocked using a sliding window approach.

VI. Experiments and Results

To demonstrate the usefulness of the benchmark suite for gaining insights from different algorithms and scenarios, we

performed experiments that compare the algorithms introduced in section IV-B in different scenarios. The replicability experiment was run on a computer with a Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz, 16GB memory, the rest of the experiments on a computer with a Intel(R) Xeon(R) W-1270 CPU @ 3.40GHz and 16GB of memory.

A. Replicability

As the first evaluation, we take a look at how consistently the exact same scenario leads to the same outcome across different runs. In order to investigate this behavior, we used three different setups and executed them 100 times each.

To investigate the variability in a conflict-free schedule execution, we placed one single agent on the warehouse map. The distance from start to goal was 45 cells, with a cell size of $0.35\,\mathrm{m}$ side length. Here, 100% of the runs were successful, with a standard deviation in execution time of only $0.4\,\mathrm{s}$. This means without conflicts between agents, the simulation is very consistent.

In the other two scenarios, there are five agents on the warehouse map and the time available to complete the scenario is $300\,\mathrm{s}$. The calculation time for the schedule is very consistent for both scenarios, with the coefficient of variation (standard deviation divided by mean) below 2% for all three setups. While the schedule calculation is reproducible, the execution is less so.

One of the scenarios tests especially challenging start and goal configurations, namely those for which the schedules from all algorithms (A*, CBS, ECBS and EECBS) could not be executed successfully on the first attempt. We executed this scenario again for 100 times with a schedule planned by EECBS. The success rate shown now is 16%. The minimum execution time observed is $214\,\mathrm{s}$. Consistently, it was always the same agent that in case of failure did not reach its goal, seldomly accompanied by further failures.

In the second scenario, we used a setup where 3 out of 4 algorithms were successful in the sense that all agents reached their goals. Two algorithms, EECBS and ECBS, were used to plan and execute this scenario each 100 times. Both algorithms always found the same solution (same cost, same makespan), and both reached the goal in 95% of the cases. The mean execution time is only about 1% apart. This experiment shows that while the simulation contains some degree of randomness, it does affect different planning algorithms equally.

As to why the same scenario sometimes fails and sometimes completes successfully, the answer lies in conflict resolution. The conflicts occurring here, as determined by observation, are conflicts where two agents attempt to travel along the same path in opposite directions. The Nav2 navigation system attempts to perform a recovery, which sometimes is successful, but often fails. These results tell us that drawing conclusions from a single repetition of the scenario is not a good idea. Spurious errors can occur and should be evened out by repeating the same experiments or by setting up more scenarios.

B. Comparison of Algorithms Across Multiple Scenarios

We modify four parameters: the random seed determining the start positions of robots and goal locations, the map used (office and warehouse)[1] , the number of robots (5 and 9) and the algorithm performing the planning. With 54 different random seeds, this leads to a total of 864 experiments, split evenly across the possible parameter choices. For algorithms supporting suboptimality, we use a factor of 1.2.

Looking at the rate of success in planning a schedule within a timeout of $60\,\mathrm{s}$, we obtain the results listed in Table II. We group these results by the map being used, as there are significant differences in the planning success rate depending on the map.

Algorithm	Office	Warehouse
A*	100%	100%
CBS	99%	83%
ECBS	100%	97%
EECBS	100%	100%

TABLE II: Algorithms' success rate of finding a schedule within $60\,\mathrm{s}$, on a basis of 108 experiments for each combination of map and algorithm.

CBS gives us an indication of the difficulty of maps: with still 99% success for office, this goes down to 83% for the warehouse. The time limit of $60\,\mathrm{s}$ was selected as a high, but still reasonable number for real-life applications. In fact, lower times may be desired and some algorithms like EECBS can easily provide these, while others like CBS take significantly longer.

In the next step, we compare how well the plans generated by the different algorithms actually perform during execution in the simulation (see Table III). The normalized success rate describes all scenarios where all algorithms could calculate a schedule. For the overall success rate, cases where no schedule was found count as unsuccessful, as without a schedule, the task cannot be performed at all. Table III shows the differences between algorithms. On the office map, the success rate is high for all algorithms and the differences are relatively small. On the more difficult warehouse map, differences become more evident. The highest success rates are obtained by ECBS. CBS, on the other hand, scores decently in the normalized column, but obtains the lowest success rate in the overall examination. This is due to the fact that CBS is the computationally heaviest of the four algorithms. In 17 of 108 scenarios, CBS does not find a schedule within the timeout of $60\,\mathrm{s}$. For ECBS, this only occurs twice and for A* and EECBS, it is never the case. While the decentralized A* has a slightly lower success rate on the normalized warehouse column, it is not far below the other algorithms.

C. Task Assignment

In the previous experiments, the task assignment was performed using OR-Tools and the Euclidean distance as a heuristic. This takes about $5\,\mathrm{ms}$ and leads to the same

[1]Due to lack of space we do not report results obtained in the airport scenario, which follow similar trends seen in the warehouse one.

Algorithm	Success Rate			
	Normalized		Overall	
	Office	Warehouse	Office	Warehouse
A*	95%	81%	95%	77%
CBS	93%	84%	92%	70%
ECBS	95%	89%	95%	85%
EECBS	95%	84%	95%	78%

TABLE III: Algorithms' success rate of completing a scenario within the timeouts of 60 second (planning) and 5 min (execution). Data based on 108 experiments, except for *warehouse, normalized*, which is based on 91 experiments.

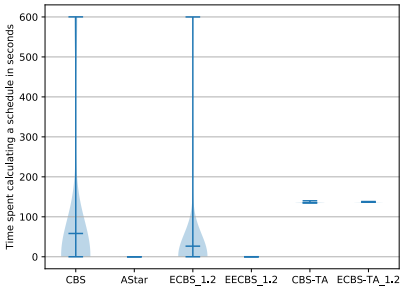

Fig. 5: Calculation time on 34 scenarios for each algorithm on the warehouse map. *1.2* denotes the suboptimality factor used.

setup for all algorithms. In this experiment we also show how CBS-TA and ECBS-TA perform. In Figure 5, the distribution of the total planning time on the warehouse map is shown for different algorithms. For EBCS-TA with suboptimality 1.2, the mean calculation time is 136.82 s with a standard deviation of 0.45 s. This was calculated for 17 scenarios with 5 and 9 agents respectively, where the increased calculation time of 600 s is available. As the low standard deviation indicates, the difference between 5 and 9 agents is negligible. For CBS-TA, the numbers are quite similar. EECBS and the decentralized A* find a solution almost instantly. For CBS and ECBS, the mean calculation time is quite low, but a few outliers exist where no solution was found even within 600 s. On the office map, the calculation time is much lower on average, but CBS-TA and ECBS-TA show a similar behavior with very consistent calculation times (around 10 s).

Algorithm	**Mean Makespan**	**Mean Cost**
A*	65.3	227.1
CBS	65.6	233.5
ECBS_1.2	65.6	234.1
EECBS_1.2	66.3	234.2
CBS-TA	59.3	213.5
ECBS-TA_1.2	66.2	227.3

TABLE IV: Algorithms' mean cost and makespan for the warehouse map with 9 agents. Only the 39 scenarios where all algorithms found a solution within 600 s are considered.

Table IV shows the differences in costs and makespan resulting from the different assignment strategies as described above. The upper four algorithms use the same assignment of agents to goals, whereas in the lower two, it can be different. For the first group, decentralized A* has the lowest total cost, which is expected as no conflict avoidance takes place. CBS, as an optimal algorithm, has a slightly smaller total cost than the other two which allow a suboptimality of factor 1.2.

On the other hand, for CBS-TA, the mean cost and makespan are both significantly lower than for CBS, which means a different, better assignment was found in more cases. The difference is less pronounced for ECBS and ECBS-TA respectively. The fact that mean cost is lower, but the mean makespan is actually higher for ECBS-TA shows us that the cost was the optimized value here.

D. Summary

In summary, our experiments suggest that using suboptimal algorithms is a viable solution. ECBS turned out to be faster than CBS and also delivered a higher success rate. EECBS is even faster and never failed to find a solution in our scenarios, at a small cost in the success rate. Using algorithms with included task assignment is promising, as a smaller total cost can be achieved. For some scenarios, their calculation time might be too long. In this case, some algorithms may offer parameters that can be used to limit the search space of possible assignments.

Dependent on the environment and the robots used, the local recovery feature offered by Nav2 may be sufficient in some cases to even use decentralized approaches such as A*. While the success rate is higher for variants of CBS, A* results are not that much worse that a decentralized approach should be ruled out. This applies especially if the environment is less static than in the scenarios we set up. If robots have to rely even more on local observations due to a rapidly changing environment, an approach using central planning is further disadvantaged.

E. Limitations

A main limitation of our benchmark concerns its scalability. In particular, the computationally expensive Nav2 and Gazebo pose limitations. Going well above ten agents on a normal computer is not possible for execution. To improve this, the researcher would need to scale out instead, distributing the agents over multiple machines connected in a network or in the cloud [14], similar to real-life mobile robots that usually possess individual onboard hardware.

VII. Conclusion and Outlook

In this paper, we introduce a novel benchmark for multi-agent task assignment and path planning in complex and cluttered environments. Our benchmark uses state-of-the-art robotic frameworks to study multi-agent robot navigation systems in realistic 3D simulated environments. To this end, we provide typical scenarios and metrics from the intralogistic and household domains. Our evaluation shows how the benchmark can be used to gain novel extensive insights from the interfaced multi robot planning algorithms. Overall, MRP-Bench allows practitioners and researchers to compare their novel multi-robot planning algorithms against the state of the art with little effort. As future work, we aim to extend the approach to dynamic environments, e.g. by including simulated humans and predicting their motion.

REFERENCES

[1] Nav2 — navigation 2 1.0.0 documentation. https://navigation.ros.org/. Accessed: 2022-5-27.

[2] Welcome to Flatland. https://flatland.aicrowd.com/intro.html. Accessed: 2022-3-31.

[3] R. Amsters and P. Slaets. Turtlebot 3 as a robotics education platform. In *Robotics in Education*, pages 170–181. unknown, 2020.

[4] M. Barer, G. Sharon, R. Stern, and A. Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*, July 2014.

[5] 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*, 7(2):882–889, 2021.

[6] A. Developers. multi-robot-fleet-sample-application. https://github.com/aws-samples/multi-robot-fleet-sample-application. Accessed: 2022-5-30.

[7] Esteve Fernandez, Tully Foote, William Woodall, Dirk Thomas. Next-generation ROS: Building on DDS, September 2014.

[8] M. Gebser, P. Obermeier, T. Otto, T. Schaub, O. Sabuncu, v. Nguyen, and T.C. Son. Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming*, 18(3-4):502–519, 2018.

[9] Google. OR-Tools. https://developers.google.com/optimization. Accessed: 2022-5-30.

[10] 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*, 4(2):100–107, July 1968.

[11] 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*, 6(3):4536–4543, 2021.

[12] W. Hönig. libMultiRobotPlanning. https://github.com/whoenig/libMultiRobotPlanning. Accessed: 2022-5-30.

[13] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian. Conflict-based search with optimal task assignment. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2018.

[14] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, H. Zhan, D. Xu, R. Ghassemi, J. Kubiatowicz, I. Stoica, J. Gonzalez, and K. Goldberg. Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2. *arXiv:2205.09778 [cs]*, 2022.

[15] IEEE Spectrum. TurtleBot 3. https://robots.ieee.org/robots/turtlebot3/, 2017. Accessed: 2022-5-30.

[16] O. Kaduri, E. Boyarski, and R. Stern. Algorithm selection for optimal multi-agent pathfinding. *ICAPS*, 30:161–165, 2020.

[17] L. Kästner, T. Bhuiyan, T.A. Le, E. Treis, J. Cox, B. Meinardus, J. Kmiecik, R. Carstens, D. Pichel, B. Fatloun, et al. Arena-Bench: A benchmarking suite for obstacle avoidance approaches in highly dynamic environments. *IEEE Robotics and Automation Letters*, 7(4):9477–9484, 2022.

[18] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[19] 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 (AAAI)*, pages 12353–12362, 2021.

[20] H. Ma, J. Li, T.S. Kumar, and S. Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 837–845, 2017.

[21] S. Macenski, F. Martín, R. White, and J. Ginés Clavero. The Marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[22] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66), 2022.

[23] S. Mohanty, E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, G. Vienken, I. Sturm, G. Sartoretti, and G. Spigler. Flatland-RL : Multi-agent reinforcement learning on trains, 2020.

[24] Open-RMF. RMF demos. https://github.com/open-rmf/rmf_demos. Accessed: 2022-5-19.

[25] Open Robotics. Programming multiple robots with ROS 2. https://osrf.github.io/ros2multirobotbook/print.html. Accessed: 2022-5-30.

[26] OSRF. Gazebo 11.0.0 release. https://classic.gazebosim.org/blog/gazebo11, January 2019. Accessed: 2022-5-30.

[27] G. Papoudakis, F. Christianos, L. Schäfer, and S.V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[28] L. Perron and V. Furnon. OR-Tools, July 2019.

[29] L. Ramshaw and R.E. Tarjan. On Minimum-Cost assignments in unbalanced bipartite graphs. 2012.

[30] L. Rocha and K. Vivaldini. Plannie: A benchmark framework for autonomous robots path planning algorithms integrated to simulated and real environments. In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 402–411, 2022.

[31] M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T.G. Rudner, C.M. Hung, P.H. Torr, J. Foerster, and S. Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188, 2019.

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

[33] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Satish Kumar, E. Boyarski, and R. Barták. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the 12th International Symposium on Combinatorial Search, SoCS 2019*, pages 151–158. AAAI press, 2019.

[34] N. Sturtevant. MAPF benchmarks. https://movingai.com/benchmarks/mapf.html. Accessed: 2022-3-30.

[35] N. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[36] M. Yagiura and T. Ibaraki. The generalized assignment problem and its generalizations. *St. Marys College of Maryland, St. Marys City, MD, USA, Tech. Rep.*, 1989.