MOTIONBENCHMAKER: A Tool to Generate and Benchmark Motion Planning Datasets

Constantinos Chamzas[1], Carlos Quintero-Peña[1], Zachary Kingston[1], Andreas Orthey[2], Daniel Rakita[3], Michael Gleicher[3], Marc Toussaint[2] and Lydia E. Kavraki[1]

***Abstract*—Recently, there has been a wealth of development in motion planning for robotic manipulation—new motion planners are continuously proposed, each with their own unique strengths and weaknesses. However, evaluating new planners is challenging and researchers often create their own ad-hoc problems for benchmarking, which is time-consuming, prone to bias, and does not directly compare against other state-of-the-art planners. We present MOTIONBENCHMAKER, an open-source tool to generate benchmarking datasets for realistic robot manipulation problems. MOTIONBENCHMAKER is designed to be an extensible, easy-to-use tool that allows users to both generate datasets and benchmark them by comparing motion planning algorithms. Empirically, we show the benefit of using MOTIONBENCHMAKER as a tool to procedurally generate datasets which helps in the fair evaluation of planners. We also present a suite of 40 prefabricated datasets, with 5 different commonly used robots in 8 environments, to serve as a common ground to accelerate motion planning research.**

***Index Terms*—Motion and Path Planning; Manipulation Planning; Data Sets for Robot Learning**

I. INTRODUCTION

MOTION planning is a core component of robotic manipulation [1]. For example, motion planning is essential in pick-and-place tasks [2], finding geometrically-constrained motions such as opening drawers and doors [3], and as a tool in task and motion planners to evaluate the feasibility of long-horizon plans [4]. The multitude of applications of motion planning has given rise to a multitude of motion planners to tackle these specific problems, each employing their own heuristics [5] to address the challenging general problem [6].

Despite the plethora of planning methods proposed over the years, little emphasis has been placed on creating a common ground to evaluate these planners—there are no shared benchmarking datasets tailored to manipulation problems that are commonly found in the literature [7]. The lack of shared environments for evaluation often forces researchers to create their own, making it challenging for practitioners to understand the advantages or disadvantages of a particular method if not directly compared. Additionally, crafting bespoke planning problems to evaluate a method is very time consuming, and could lead to incorrect conclusions due to unintentional biases in design. Finally, with the advent of learning-based planning methods (e.g., [8]–[10]), there has been an increased need for readily available open-source datasets that can be used for training and testing.

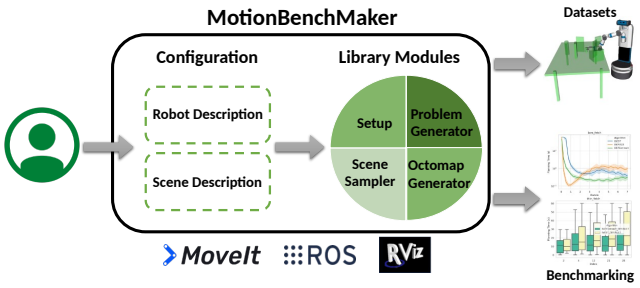

Fig. 1. MOTIONBENCHMAKER architecture

We introduce MOTIONBENCHMAKER, a tool that facilitates the creation of motion planning datasets to ease the evaluation of motion planning algorithms in "realistic" manipulation tasks. MOTIONBENCHMAKER was inspired by common issues found in evaluating sampling-based planners on high-DOF robots. Unlike most existing benchmarking resources, which are designed for low-DOF robots or free-flying systems (see Table I), MOTIONBENCHMAKER is intended for modern high-DOF robots in "realistic" scenes, and its capabilities are broadly useful to other types of planners, e.g., classical, optimization-based, and learning-based. MOTIONBENCHMAKER consists of a set of tools in the form of modules Fig. 1, which can be utilized by user scripts and human-readable configuration files. The two main use cases for MOTIONBENCHMAKER are the generation of motion planning datasets and subsequent evaluation of motion planners on these datasets. We also provide 40 prefabricated datasets (5 different robots in 8 different environments) which are open source along with MOTIONBENCHMAKER[1]. A video is also provided that visually presents this work [2].

MOTIONBENCHMAKER specifies motion planning problems as *robot-agnostic manipulation queries* which depend only on the environment geometry—with this, it is easy to integrate new problems and new robots to create new datasets (e.g., see Fig. 6). Planning problems within a dataset are randomly generated

Manuscript received: September, 9, 2021; Accepted November, 18, 2021.

This paper was recommended for publication by Editor Hanna Kurniawati upon evaluation of the Associate Editor and Reviewers' comments. At Rice University, this work was supported in part by NSF 1718478, NSF 2008720, NSF-GRFP 1842494, and Rice University Funds. Work on this project by DR and MG was supported in part by NSF 1830242.

[1]CC, CQP, ZK, and LEK are with the Department of Computer Science, Rice University, Houston, TX, USA {chamzas,carlosq,zak,kavraki}@rice.edu

[2]AO and MT are with the Learning and Intelligent Systems Lab, TU Berlin, Germany {orthey}@campus.tu-berlin.de, {toussaint}@tu-berlin.de

[3]DR and MG are with the Department of Computer Sciences, University of Wisconsin-Madison, {rakita, gleicher}@cs.wisc.edu

Digital Object Identifier (DOI): see top of this page.

[1]https://github.com/KavrakiLab/motion_bench_maker

[2]https://youtu.be/t96Py0QX0NI

given a nominal environment and a set of tunable parameters. These parameters specify how objects in the environment can vary in their pose and control how new samples of planning problems are be procedurally generated. MOTIONBENCHMAKER also provides the ability to convert scenes described with geometric primitives and meshes to a "sensed" representation, i.e., point clouds and octomaps [11]. MOTIONBENCHMAKER is fully compatible with the ROS [12] ecosystem of tools and interfaces such as visualization with RViz and motion planning through MoveIt [13] and Robowflex [14]. To summarize, with MOTIONBENCHMAKER we contribute a tool which

- has a modular and open architecture to facilitate creating new datasets,
- procedurally generates new datasets by randomly varying scenes,
- can convert scenes to "sensed" representations,
- has benchmarking capabilities,
- and is easy to integrate into the existing ROS ecosystem.

The rest of the paper is organized as follows. In Sec. II we review other works in robotic benchmarking and dataset. In Sec. III we describe the modules of MOTIONBENCHMAKER and in Sec. IV, we show how MOTIONBENCHMAKER facilitates the generation of motion planning datasets incorporating new robots into existing scenes without much effort. In Sec. V-A, we show that it is possible to infer an incorrect conclusion when comparing motion planning algorithms due to limited data, emphasizing the importance of MOTIONBENCHMAKER's problem generation. In Sec. V-B we show that the prefabricated datasets in MOTIONBENCHMAKER are challenging even for fine-tuned planners, and no sampling-based planner rules over all.

II. RELATED WORK

Well-maintained datasets such as ImageNet [34] or Tencent ML-Images [35] are fundamental for algorithmic breakthroughs in research fields like computer vision. To achieve similar feats, the robotics community has developed several high-quality datasets. We give a brief overview of the most popular ones (as of August 2021) with a focus on datasets for manipulation planning. A more detailed overview can be found in [36] (up until 2015).

We compare datasets with each other based on six desirable properties. First, we compare if a dataset is *procedurally generated*, meaning if there exists an algorithmic generation of problems from a given scenario. Second, we compare *planner benchmarking* capabilities, meaning if there exists a tool to benchmark different motion planning algorithms on the dataset. Third, if a dataset is procedurally generated, we check if there is an interface with *tunable parameters*, i.e., if users can influence the generation process. Fourth, we check if the dataset is *high-dof*, i.e., if there exist robots with more than 6-DOF. Fifth, we check if the dataset contains *sensed representations*, i.e., if there exist environments in the dataset which are built from sensor information. Finally, we check for *articulated robots*, i.e., if the datasets contain robots that are beyond rigid bodies in 2D or 3D. Other properties could be examined, but we consider these properties necessary for a tool that focuses on manipulation.

TABLE I
RELEVANT DATASETS IN ROBOTICS AS OF AUGUST 2021.

Paper	Year	Procedurally Generated	Planner Benchmarking	Tunable Parameters	High-Dof (> 6)	Sensed Representation	Articulated Robots
Vehicle Navigation							
CommonRoad [15]	2017	×	✓	×	×	×	×
Robot@Home [16]	2017	×	×	×	×	✓	×
Multi-Agent Path-Find Benchmark [17]	2019	×	✓	×	×	×	×
MAVBench [18]	2020	×	×	×	×	✓	×
BARN [19]	2020	✓	✓	✓	×	×	×
Bench-MR [20]	2021	×	✓	×	×	✓	×
PathBench [21]	2021	✓	✓	✓	×	×	×
General Robotics							
OMPLBenchmarks [22]	2015	×	✓	×	×	×	×
Robobench [23]	2016	×	✓	×	✓	✓	✓
Roboturk (Teleoperation database) [24]	2019	×	×	×	✓	✓	✓
RLBench [25]	2020	✓	×	✓	✓	✓	✓
OCRTOC [26]	2021	✓	✓	×	✓	✓	✓
Robot Manipulation							
ACRV picking benchmark [2]	2017	×	✓	×	✓	✓	✓
RoboNet [27]	2019	×	×	×	✓	✓	✓
GraspNet [28]	2020	×	×	×	×	×	×
Brown Planning Benchmarks [29]	2020	✓	✓	×	✓	×	✓
Aerial Manipulation [30]	2020	×	×	×	✓	✓	✓
Bimanual Manipulation Benchmark [31]	2020	×	✓	×	✓	✓	✓
In-hand manipulation benchmark [32]	2020	×	×	×	✓	×	✓
ProbRobScene [33]	2021	✓	×	✓	✓	×	✓
MOTIONBENCHMAKER **(ours)**	2021	✓	✓	✓	✓	✓	✓

As can be seen in Table I, we divide the datasets into three categories. The first category is datasets for vehicle navigation. Several high-quality datasets exists like common road [15], bench mobile robot [20] and the benchmark for autonomous robot navigation (BARN) [19]. Similar datasets concentrate on indoor-navigation [16], 2D multi-agent path-finding [17], discrete point-robot path finding in 2D and 3D [21], free-flying robots [22] or drones [18]. Our paper is complementary to vehicle navigation in that we concentrate on robot manipulation tasks.

The second category of datasets is focused on general robotics. These works aim at covering broad robotic categories like providing datasets and tools for remote teleoperation [24] or object rearrangement [26]. While many papers are concentrating on learning-based approaches [25], there is also a trend towards more reproducibility, for example by using containerization [21] to ease comparison over different operating systems or configurations.

However, several tools in robotics have been developed specifically for manipulation tasks. While the data generation is often similar, approaches differ by focusing either on learning-based algorithms or on planning-based algorithms. In learning-based approaches like RobotNet [27], the focus is more on generating diverse camera streams. In planning-based approaches like the Brown planning benchmark [29] the focus is more on creating mesh-based representations of the world useful to benchmark motion planners [37]. Other frameworks like ProbRobScene [33] are independent of the algorithm used and focus instead on generating scenes automatically. A particular

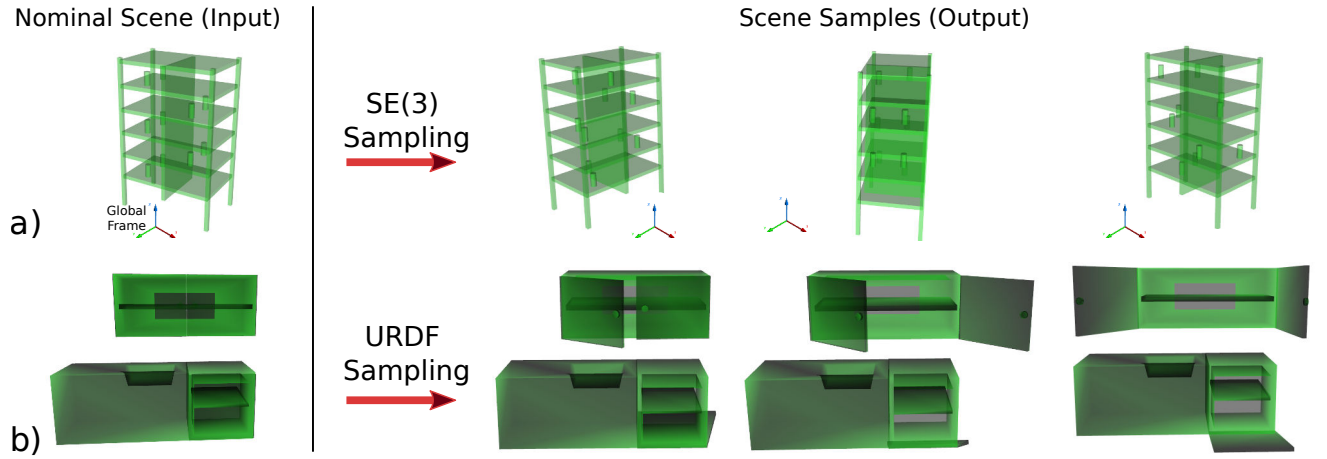

Fig. 2. *Scene Sampler*: The scene sampler module generates variations on a nominal input scene given sampling parameters. When performing $\mathrm{SE}(3)$ sampling (shown in **a)**), variation in the pose of objects in the scene can be specified *globally* (e.g., the bookshelf moving) and *locally* (e.g., the cylinders moving on the shelves of the bookshelf). When performing URDF sampling (shown in **b)**), valid configurations of the kinematic structure specified by the URDF are sampled, which results in different configurations of the cabinets.

dataset aimed at grasping is GraspNet, which concentrates on using the YCB dataset of objects [28] to generate large sets of grasping poses. Similar datasets and benchmark utilities concentrate on specific aspects of manipulation. This involves tasks like bimanual manipulation [31], in-hand manipulation [32], cloth manipulation [38], aerial manipulation [30], or solving Rubik's cube [39].

MOTIONBENCHMAKER differs from all those approaches by (a) focusing on benchmarks specifically for motion planning algorithms, (b) having an incremental generation tool to create diverse sets of manipulation tasks, and (c) by concentrating on broad manipulation capabilities for diverse high-dimensional robotic arms. This involves not only single gripper grasps but also bimanual manipulation (e.g., using the Baxter robot) and multi-finger manipulation (e.g., using the ShadowHand robot).

III. LIBRARY MODULES

MOTIONBENCHMAKER is a flexible modular library composed of four basic modules, shown in Fig. 1. The *Scene Sampler* shown in Fig. 2, creates variations of a given nominal scene. The *Octomap Generator*, shown in Fig. 3, converts a geometric scene to a point cloud and subsequently an octomap. The *Problem Generator* generates motion planning problems given a scene, robot, and necessary configuration files. Finally, the *Setup* module enables the easy creation and usage of the generated datasets.

A. Scene Sampler

Given variation parameters, the *Scene Sampler* module procedurally generates multiple scenes by randomly changing the nominal scene. Currently, two complementary types of sampling are provided namely $\mathrm{SE}(3)$ and *URDF* sampling as shown in Fig. 2.

1) $\mathrm{SE}(3)$ *sampling:* For $\mathrm{SE}(3)$ sampling, the nominal scene is a set of collision objects with $\mathrm{SE}(3)$ poses relative to the global frame (shown in Fig. 2a). New scenes are generated by adding random noise to the $\mathrm{SE}(3)$ poses of the collision objects [40]. The pose of the collision objects in the nominal scene serves as the mean of the sampling distribution and the variance (Gaussian) or bounds (Uniform) parameters are specified through a configuration file. Finally, the random perturbations to the collision objects' poses can happen both *globally*, e.g., the shelf in Fig. 2a is moved with respect to the global frame, and *locally*, e.g., the cylinders in Fig. 2a are perturbed with respect to the local frame of the shelf. Examples of samples drawn are shown on the right side of Fig. 2a.

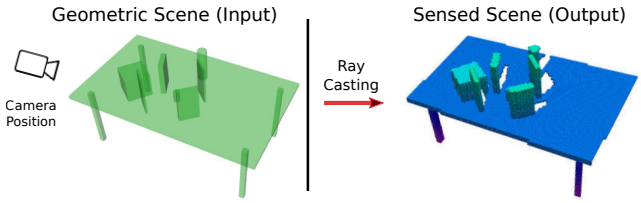

Fig. 3. *Octomap Generator*: The octomap generator module generates a sensed representation (point cloud and octomap) by emulating a depth camera from different positions.

2) URDF sampling: In this type of sampling, the nominal scene is specified as a URDF (Unified Robot Description Format) file. The URDF specifies the number of joints that describe the kinematic relations of objects in the scene. By sampling valid configurations of this URDF (that is, collision-free with itself), we can generate different scenes. This type of sampling emulates movements of objects subject to kinematic constraints such as cabinets opening and closing, shown in Fig. 2b.

B. Octomap Generator

Octomap Generator is an optional module that provides a way to convert geometric scene representations (i.e., geometric primitives and meshes) to point clouds and octomaps [11], as shown in Fig. 3. The point cloud is generated by specifying in the frame(s) of the depth camera which is simulated with `gl_depth_sim`[3]. This point cloud is later converted to an octomap. When a dataset is generated, all three representations

[3]https://github.com/Jmeyer1292/gl_depth_sim

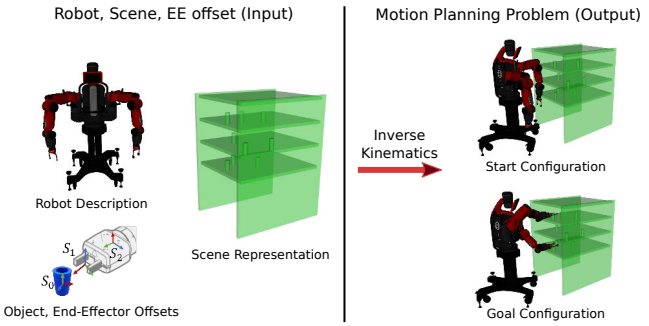

Fig. 4. *Problem Generator*: Given a robot description (URDF), geometric scene, and object-centric end-effector poses, the problem generator creates a motion planning problem. To have robot-agnostic problems, the start and goal of the problem can be specified as end-effector grasp poses, e.g., by specifying pose S_1 relative to the object's reference frame S_0. Additionally, a robot-specific transformation S_2 relative to the robot's end-effector frame is applied to the pose to account for different gripper idiosyncrasies. Full joint configurations can also be used as start/goal. Finally, a robot also has an optional base offset (not shown) that can be specified.

(geometric, point cloud, octomap) can be simultaneously produced. Note that for motion planning, an octomap representation usually has a much higher collision checking time and is an over-approximation of the geometry, leading to harder motion planning problems. Nevertheless, we consider the sensed representation more "realistic" since it can be provided from any RGB-D camera, and is often used in practice.

C. Problem Generator

One critical idea in MOTIONBENCHMAKER is the fact that motion planning problems can be easily generated for any robot-scene pair. When done by hand, this process can be challenging and time-consuming since valid start and goal joint configurations in a motion planning problem depend on both the robot and the scene. The *Problem Generator* provides this functionality by defining a set of start and goal manipulation *queries*. These *queries* are specified as pose offsets expressed in the frame of collision objects in the nominal scene. For example as seen in Fig. 4, the query expresses how the blue cup can be grasped. This is achieved by defining appropriate object-centric offsets that are robot agnostic. This specification is conceptually similar to the affordance templates proposed by [41]. Finally, objects in the scene can be attached to the end-effector(s) to emulate pick and place tasks.

The *Problem Generator* creates full-motion planning requests (i.e., start/goal configurations in joint space) by performing collision-aware inverse kinematics. These requests can be readily used together with a scene, to create a varied set of motion planning problems. Note that there can be multiple queries defined for a scene, but the generated requests will consist of a single start-goal pair. During generation, a planner can optionally be used to verify the feasibility of a problem.

As an additional feature, the *Problem Generator* supports the specification of manipulation queries for multiple end-effectors in the kinematic chain, e.g., multi-tip queries. This is useful for applications in bimanual manipulation and when planning for dexterous hand robots such as the bookshelf with Baxter and the Shadowhand examples respectively (see Fig. 6).

```
1  // Load the dataset given a meta-data file
2  auto setup = std::make_shared<Setup>("conf.yaml");
3  auto robot = setup->getRobot();
4  auto planner = setup->createPlanner("planner");
5  Experiment experiment("exp", Profiler::Options());
6
7  for (int i = 1; i <= setup->getNumSamples(); i++)
8  {
9      // Load the ith scene in the dataset
10     auto scene = std::make_shared<Scene>(robot);
11     setup->loadGeometricScene(i, scene);
12
13     for (auto planner_name : {"PRM", "BiEST"})
14     {
15         // Load the start and goal configuration
16         auto request = setup->createRequest();
17         setup->loadRequest(i, request);
18
19         // Set planner.e.g., PRM, BiEST
20         request->setConfig(planner_name);
21         experiment.addQuery( //
22           planner_name, scene, planner, request);
23     }
24 }
25
26 auto data = experiment.benchmark();
27 OMPLPlanDataSetOutputter output("results.log");
28 output.dump(*data);
```

Fig. 5. A code snippet demonstrating how to load a dataset and benchmark different planners through the *Setup* module.

D. Setup

A convenient *Setup* class provides an easy-to-use interface to load created datasets and create *planner*, *scene* and *robot*. An example script with *Setup* is shown in Fig. 5. A dataset created by MOTIONBENCHMAKER comes with a meta-data manifest (line 2, "conf.yaml"). This manifest contains all the relevant parameters that define the dataset and allow the user to access the sampled scenes and requests. Once loaded, *Setup* can create instances of a *robot* (line 3) and a *planner* (line 4). Our library takes advantage of the Robowflex [14] library to provide these constructs—Robowflex encapsulates the MoveIt [13] library for motion planning and provides capabilities for planning inside simple scripts.

The *Setup* class also provides a simple way to access each *scene* (line 11) and corresponding *request* (line 17) within a dataset. After creating an *experiment* (line 5), it is easy to add this specific problem (a *scene* and *request*, line 22) to the set of problems to benchmark. After a benchmark is executed (line 26), the collected data can be output into a variety of formats, e.g., a SQL database compatible with PlannerArena [22].

IV. EXAMPLE USECASES

The user interacts with MOTIONBENCHMAKER in two ways: by creating C++ scripts that call library modules or by specifying values in configuration files to define new problems. The first case of using C++ scripts was shown in Sec. III-D. There (shown in Fig. 5) the user loads an existing dataset in MOTIONBENCHMAKER to benchmark different motion planners— benchmarking results can be plotted and analyzed through PlannerArena [22].

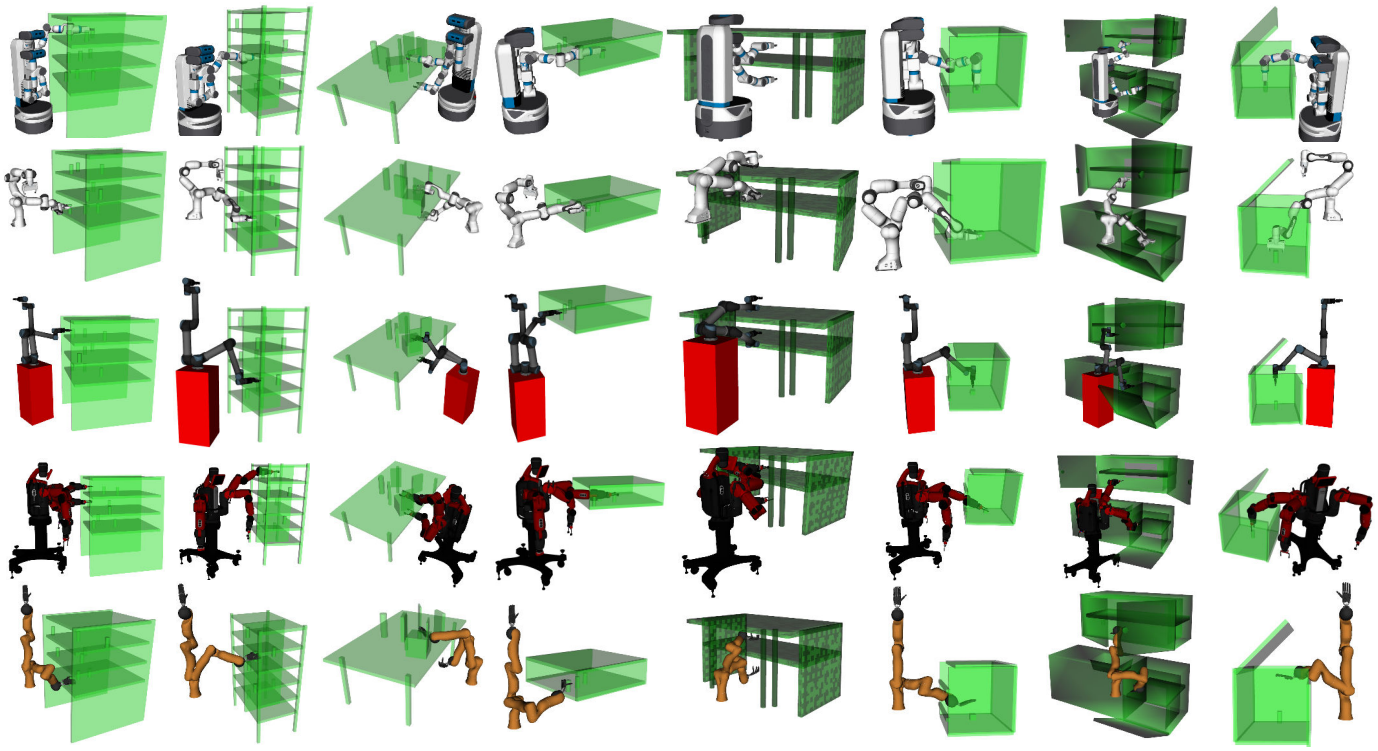

Fig. 6. Representative problems from the 40 different prefabricated datasets provided with MOTIONBENCHMAKER. There are 8 nominal scenes and 5 robots, which create the 40 datasets, each consisting of 100 different problems.

The second case considered is a user who desires to create a new dataset with a robot or scene not currently in MOTIONBENCHMAKER. The user simply needs to provide a robot description and scene description file along with the required offsets (Sec. III-C). Given these files, MOTIONBENCHMAKER through a script will procedurally generate varied motion planning problems, without the burden of manually creating valid start/goal configurations and scene samples for different problems. For example, we used this script for the 40 different prefabricated datasets, shown in Fig. 6. We created 8 nominal scenes and specified the end-effector and base offsets of the following 5 robots: a Fetch (7-8-DOF) a Panda (7-DOF), a UR5 (6-DOF), a Baxter (7-14-DOF) and a ShadowHand mounted on a KUKA arm (31-DOF).

To verify that each generated problem is feasible, we used a highly-tuned sampling-based planner with a large timeout (60 seconds) and discarded problems that could not be solved in time. For each dataset, an arbitrary number of motion planning problems can be generated but for our purposes, we created 100 motion planning problems for each dataset.

Finally, MOTIONBENCHMAKER has already been used to create a diverse set of datasets suitable for learning-based methods [8], [42], for hyper-parameter tuning methods [43], for planning under uncertainty [44], for planning in partially observable environments [45] and for planning on different abstraction levels [5], [46].

V. EVALUATIONS

In this section, we present two evaluations to showcase the efficacy of MOTIONBENCHMAKER. In Sec. V-A we demonstrate how using few motion planning problems can potentially lead to wrong conclusions, for example when comparing two motion planners. In Sec. V-B we demonstrate that many of the prefabricated datasets are challenging and no specific planner outperforms the other ones.

A. Wrong Hypothesis

Undoubtedly, in any research field, it is necessary to compare the performance of different methods. In motion planning research, it is often the case that a practitioner has a specific robot and target application in mind, which begets the need to manually construct an appropriate benchmark. Creating a benchmark from scratch without the appropriate tools is both time-consuming and challenging since a large number of problems might be required to achieve statistical significance. We note here that the designed experiments highlight the importance of using a large number of problems when comparing different planners and should not be interpreted as an indication of which planner is best. Unless indicated, planners are using default parameters from OMPL [37].

Consider the following hypothetical scenario: a practitioner wants to compare different planners on a picking task. Specifically, the problem of interest involves a UR5 robot tasked with picking a cylinder from a shelf, shown in Fig. 7a. Say the practitioner either samples or chooses specific instances of this scene: in Fig. 7b and Fig. 7c, we present the worst case scenarios for this practitioner (for 5, 10, 50, and 100 problem instances) in terms of drawing conclusions on their planner's performance.

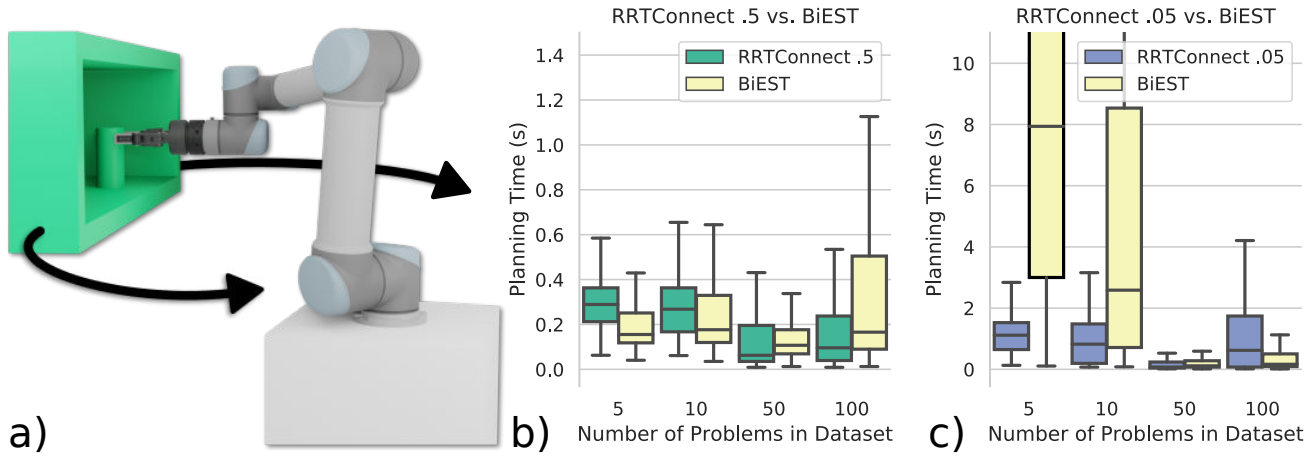

Fig. 7. **a)** One of the 100 sampled problems in a simple shelf-picking dataset for a UR5 robot. Both the relative angular position of the shelf as well as the position of the cylinder (not shown for visual clarity) vary between motion planning problems. **b), c)** Two different adversarial orderings of the same 100 motion planning problems. The Y-AXIS shows the planning time while the X-AXIS is the number of motion planning problems considered. A timeout of 60 seconds was used for all the problems and each motion planning problem was solved 20 times. It is clear that when using only a small number of motion planning problems, the wrong conclusions can be drawn, as the best performing planner can be different when considering all 100 problems.

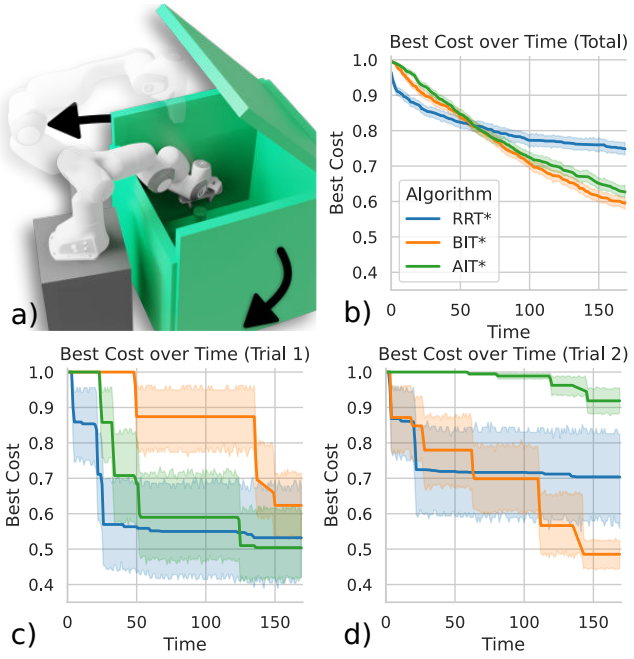

Fig. 8. **a)** One of the 100 sampled in a box-picking dataset with the Panda robot. In this task, both the relative angular position of the box and the position of the object in the box (not shown for visual clarity) vary. **b), c), d)** Show the normalized cost for 3 different optimizing planners in the box-picking task. The median is shown of 5 independent runs for each planner with a 99% confidence interval. Trial 1 and 2 shown in **c), d)** show the convergence plots of RRT* BIT* and AIT* on a single, specific motion planning problem while **b)** shows results considering all 100 problems. Results in aggregate differ between Trial 1 and Trial 2 demonstrating that using a few motion planning problems could potentially lead to incorrect conclusions.

We generated 100 feasible motion planning problems as described in Sec. IV and benchmarked planning time for BIEST [47] and RRTConnect [48] (we use RRTConnect with two different "range" values, 0.05 and 0.5, which controls the $\mathcal{C}$-space expansion step). For each specific problem (an instance of the scene), the problem was solved 20 times (with a 60 seconds timeout), for a total of 2000 data points given 100 scenes. In figures Fig. 7b and Fig. 7c you can see two different adversarial orderings of the data. That is, for both of these plots, we sorted the same motion planning problems in the dataset such that problems early in the dataset have the largest difference in average planning time between the two compared planners. In Fig. 7b BIEST and RRTConnect with range 0.5 are compared. The X-AXIS denotes how many problems from the sorted problems are considered. Here, BIEST is better when considering only 5 or 10 problems, while when considering the entire dataset (100 problems) it is clear that RRTConnect is more performant. In Fig. 7c, the same effect is demonstrated between BIEST and RRTConnect with a range parameter of 0.05, with BIEST faster only after aggregating the results from all 100 problems. This empirically shows the danger in considering only a few problem instances for evaluation. MOTIONBENCHMAKER provides the tools necessary to easily create varied datasets to help avoid this problem.

Beyond planning time, this phenomenon could occur when comparing other planner metrics, e.g., comparing the best cost over time for asymptotically-optimal sampling-based planners, as shown in Fig. 8. Here the experiment entails a Panda robot grasping a cylinder from the box with similar variation as in Fig. 7. In this example cost is defined as joint path length, but different costs suchs as clearance or cartesian length can be specified through the MoveIt [13] interface. We show the median of the best normalized cost found for RRT* [49], BIT* [50], and AIT* [51]. Each planner is run 5 times per problem, with a given 180 seconds planning time. Fig. 8c and Fig. 8d indicate two different conclusions about which planner performs best when considering a single problem instance. As above, incorrect conclusions would be drawn about planner performance in this domain if only based on a specific problem instance—Fig. 8b shows the aggregated results over all 100 problems, which provides a stronger conclusion. Note that in general all datasets are prone to bias, but procedurally generating more instances ameliorates this bias.

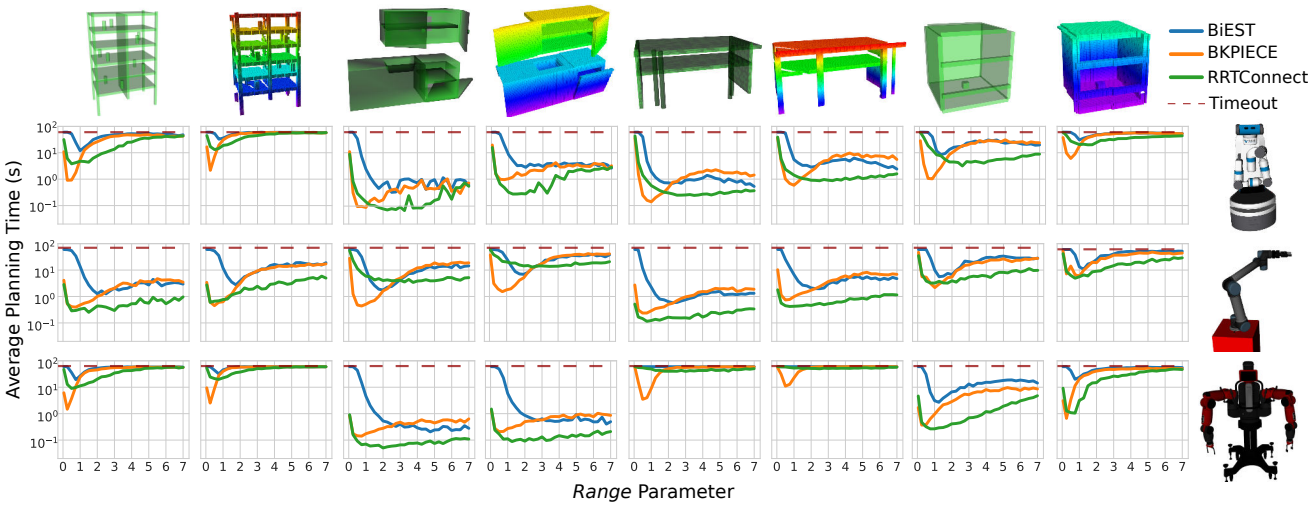

Fig. 9. Timing results for three planners (BKPIECE, RRTConnect, and BIEST) in 4 environments from Fig. 6 (both geometric and sensed) on 3 different robots (the Fetch, UR5, Baxter) for a total of 12 datasets. We plan for both arms of the Baxter in the table environment while for only in the rest. In each plot on this matrix, the value of each of the planner's *range* parameters (which controls $\mathcal{C}$-space expansion) is varied between 0 to 7, in increments of 0.25. The average planning time over the 100 problems in the dataset for each range parameter is shown on a log scale, plotted as a line. Planning timeout was 60 seconds, visualized as a dotted red line—lines touching the red line indicate planning timeout. Note that for many problems, specific tunings of the planners are required to solve the problem, while no planner consistently outperforms all others, indicating their difficulty and diversity.

B. Benchmarking Results of Datasets

In this section, we analyze the results of benchmarking 12 out of the 40 datasets on both the geometric and sensed (octomap) representations to demonstrate the difficulty of the provided datasets. We benchmarked three bidirectional tree-based planners, namely BIEST [52], RRTConnect [48], and BKPIECE [53] for different values of their range parameter as shown on the X-AXIS of each subplot in Fig. 9. We choose these planners, as among sampling-based planners they are typically highly performant in such tasks. Additionally, the *range* parameter (used by each planner to control the rate of expansion in $\mathcal{C}$-space) is empirically known to have a significant influence on planning performance.

Results are shown in Fig. 9. We first note that these problems demonstrate a broad range of planning performance—each of these planners varies in performance according to environment and robot and there is no clear winner across the full spectrum of problems. In several cases, even the most performant planner has more than 1 second of average planning time indicating the difficulty of the datasets. Moreover, note that planner performance is comparable between the geometric and the sensed problems, with a small performance hit in the sensed representation. Finally, we verify that these planners are sensitive to the *range* parameter, as there are clear performance peaks for the planners at specific range values for different problems.

VI. DISCUSSION

In this paper, we have presented MOTIONBENCHMAKER, a new open-source tool to procedurally generate and benchmark motion planning datasets. MOTIONBENCHMAKER supports a robot-agnostic specification of environments, sampling new planning problems from a specified distribution, and can generate "sensed" representations for realistic, challenging problems. Through our experiments, we show the importance of procedurally generating datasets, as using only a few hand-designed problems could potentially lead to incorrect conclusions.

In the future, we would like to continue extending the repository of generated datasets with the help of the community, with more robots and environments as well as supporting sequential motion planning problems, such as in task and motion planning. We would also like to add features that help users profile their dataset with a set of metrics or features, e.g., space expansiveness, to help understand what are the challenging aspects of the proposed problem. Some of the limitations of this work are that SE(3) and *URDF* sampling are only approximations of the variability of the real world, no camera data can be given to the planner for visual planning, and only geometric constraints are considered. We hope to continuously improve this tool and that it will help the community advance the field of motion planning by supporting researchers to design and share benchmarking datasets.

REFERENCES

[1] M. T. Mason, "Toward robotic manipulation," *Annual Review of Control, Robot., and Autom. Syst.*, vol. 1, pp. 1–28, 2018.
[2] J. Leitner, A. W. Tow, N. Sünderhauf, J. E. Dean, J. W. Durham, M. Cooper, M. Eich, C. Lehnert, R. Mangels, C. McCool *et al.*, "The ACRV picking benchmark: A robotic shelf picking benchmark to foster reproducible research," in *IEEE Int. Conf. Robot. Autom.*, 2017, pp. 4705–4712.
[3] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual Review of Control, Robot., and Autom. Syst.*, vol. 1, no. 1, pp. 159–185, 2018.
[4] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual Review of Control, Robot., and Autom. Syst.*, vol. 4, pp. 265–293, 2021.
[5] A. Orthey and M. Toussaint, "Section patterns: Efficiently solving narrow passage problems in multilevel motion planning," *IEEE Trans. Robot.*, pp. 1–15, 2021.
[6] J. F. Canny, *The Complexity of Robot Motion Planning*. MIT Press, 1988.
[7] G. Antonelli, "Robotic research: Are we applying the scientific method?" *Frontiers in Robotics and AI*, vol. 2, no. 13, pp. 1–4, 2015.

[8] C. Chamzas, Z. Kingston, C. Quintero-Peña, A. Shrivastava, and L. E. Kavraki, "Learning Sampling Distributions Using Local 3D Workspace Decompositions for Motion Planning in High Dimensions," in *IEEE Int. Conf. Robot. Autom.*, Jun. 2021.

[9] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *IEEE Int. Conf. Robot. Autom.*, May 2018, pp. 7087–7094.

[10] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song, "Learning to plan in high dimensions via neural exploration-exploitation trees," in *Int. Conf. on Learn. Representations*, 2020.

[11] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[12] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *IEEE ICRA workshop on open source software*, May 2009, pp. 1–6.

[13] S. Chitta, I. Sucan, and S. Cousins, "Moveit! [ros topics]," *IEEE Robot. Autom. Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[14] Z. Kingston and L. E. Kavraki, "Robowflex: Robot motion planning with MoveIt made easy," *arXiv preprint arXiv:2103.12826*, 2021.

[15] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 719–726.

[16] J. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez, "Robot@home, a robotic dataset for semantic mapping of home environments," *Int. J. of Robotics Research*, vol. 36, no. 2, pp. 131–141, 2017.

[17] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *Symposium on Combinatorial Search (SoCS)*, pp. 151–158, 2019.

[18] J. Tani, A. F. Daniele, G. Bernasconi, A. Camus, A. Petrov, A. Courchesne, B. Mehta, R. Suri, T. Zaluska, M. R. Walter *et al.*, "Integrated benchmarking and design for reproducible and accessible evaluation of robotic agents," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2020, pp. 6229–6236.

[19] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2020, pp. 116–121.

[20] 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 Robot. Autom. Letters*, vol. 6, no. 3, pp. 4536–4543, 2021.

[21] A. Toma, H. Hsueh, H. A. Jaafar, R. Murai, P. H. J. Kelly, and S. Saeedi, "Pathbench: A benchmarking platform for classical and learned path planning algorithms," in *The Conference on Robots and Vision (CRV)*, 2021, pp. 79–86.

[22] M. Moll, I. A. Sucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 96–102, 2015.

[23] J. Weisz, Y. Huang, F. Lier, S. Sethumadhavan, and P. Allen, "Robobench: Towards sustainable robotics system benchmarking," in *IEEE Int. Conf. Robot. Autom.* IEEE, 2016, pp. 3383–3389.

[24] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay *et al.*, "Roboturk: A crowdsourcing platform for robotic skill learning through imitation," in *Conf. on Robot Learning*, 2018, pp. 879–893.

[25] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "RLBench: The robot learning benchmark learning environment," *IEEE Robot. Autom. Letters*, vol. 5, no. 2, pp. 3019–3026, 2020.

[26] Z. Liu, W. Liu, Y. Qin, F. Xiang, M. Gou, S. Xin, M. A. Roa, B. Calli, H. Su, Y. Sun, and P. Tan, "OCRTOC: A cloud-based competition and benchmark for robotic grasping and manipulation," *IEEE Robot. Autom. Letters*, vol. 7, no. 1, pp. 486–493, 2022.

[27] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn, "Robonet: Large-scale multi-robot learning," in *Conf. on Robot Learning*. PMLR, 2020, pp. 885–897.

[28] H.-S. Fang, C. Wang, M. Gou, and C. Lu, "Graspnet-1billion: A large-scale benchmark for general object grasping," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 444–11 453.

[29] S. Murray, G. D. Konidaris, and D. J. Sorin, "Roadmap subsampling for changing environments," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pp. 5664–5670.

[30] A. Suarez, V. M. Vega, M. Fernandez, G. Heredia, and A. Ollero, "Benchmarks for aerial manipulation," *IEEE Robot. Autom. Letters*, vol. 5, no. 2, pp. 2650–2657, 2020.

[31] K. Chatzilygeroudis, B. Fichera, I. Lauzana, F. Bu, K. Yao, F. Khadivar, and A. Billard, "Benchmark for bimanual robotic manipulation of semi-deformable objects," *IEEE Robot. Autom. Letters*, vol. 5, no. 2, pp. 2443–2450, 2020.

[32] S. Cruciani, B. Sundaralingam, K. Hang, V. Kumar, T. Hermans, and D. Kragic, "Benchmarking in-hand manipulation," *IEEE Robot. Autom. Letters*, vol. 5, no. 2, pp. 588–595, 2020.

[33] C. Innes and S. Ramamoorthy, "ProbRobScene: A probabilistic specification language for 3D robotic manipulation environments," in *IEEE Int. Conf. Robot. Autom.*, 2021.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[35] B. Wu, W. Chen, Y. Fan, Y. Zhang, J. Hou, J. Liu, and T. Zhang, "Tencent ml-images: A large-scale multi-label image database for visual representation learning," *IEEE Access*, vol. 7, pp. 172 683–172 693, 2019.

[36] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the Yale-CMU-Berkeley object and model set," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[37] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robot. Autom. Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[38] I. Garcia-Camacho, M. Lippi, M. C. Welle, H. Yin, R. Antonova, A. Varava, J. Borras, C. Torras, A. Marino, G. Alenya *et al.*, "Benchmarking bimanual cloth manipulation," *IEEE Robot. Autom. Letters*, vol. 5, no. 2, pp. 1111–1118, 2020.

[39] B. Yang, P. E. Lancaster, S. S. Srinivasa, and J. R. Smith, "Benchmarking robot manipulation with the rubik's cube," *IEEE Robot. Autom. Letters*, vol. 5, no. 2, pp. 2094–2099, 2020.

[40] A. Yershova and S. M. LaValle, "Deterministic sampling methods for spheres and so (3)," in *IEEE Int. Conf. Robot. Autom.*, vol. 4, 2004, pp. 3974–3980.

[41] S. Hart, P. Dinh, and K. Hambuchen, "The affordance template ROS package for robot task programming," in *IEEE Int. Conf. Robot. Autom.*, 2015, pp. 6227–6234.

[42] E. Pairet, C. Chamzas, Y. R. Petillot, and L. E. Kavraki, "Path planning for manipulation using experience-driven random trees," *IEEE Robot. Autom. Letters*, vol. 6, no. 2, p. 3295–3302, Apr. 2021.

[43] M. Moll, C. Chamzas, Z. Kingston, and L. E. Kavraki, "HyperPlan: A framework for motion planning algorithm selection and parameter optimization," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2021.

[44] C. Quintero-Peña, A. Kyrillidis, and L. E. Kavraki, "Robust Optimization-based Motion Planning for high-DOF Robots under Sensing Uncertainty," in *IEEE Int. Conf. Robot. Autom.*, Jun. 2021, pp. 9724–9730.

[45] C. Quintero-Peña, C. Chamzas, V. Unhelkar, and L. E. Kavraki, "Motion Planning via Bayesian Learning in the Dark," in *ICRA: Workshop on Machine Learning for Motion Planning*, Jun. 2021.

[46] A. Orthey, S. Akbar, and M. Toussaint, "Multilevel motion planning: A fiber bundle formulation," 2020, arXiv:2007.09435 [cs.RO].

[47] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," in *Int. Wksp. on the Algorithmic Foundations of Robotics*. Springer, 1998.

[48] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, 2000, pp. 293–308.

[49] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[50] J. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *IEEE Int. Conf. Robot. Autom.*, 2015, pp. 3067–3074.

[51] M. P. Strub and J. D. Gammell, "Adaptively Informed Trees (AIT*): Fast asymptotically optimal path planning through adaptive heuristics," in *IEEE Int. Conf. Robot. Autom.*, 2020, pp. 3191–3198.

[52] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE Int. Conf. Robot. Autom.*, vol. 3. IEEE, 1997, pp. 2719–2726.

[53] I. A. Sucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Int. Wksp. on the Algorithmic Foundations of Robotics*, vol. 57. Springer, 2009, pp. 449–464.