Planners Benchmarking Documentation

Shadow Robot Company

Jan 16, 2019

Contents

At Shadow Robot, we use the Universal Robot Arms (UR5 and UR10) with our robot hands to perform experiments in order to improve our robot manipulation capabilities. However, a challenge remains to find accurate and reliable plans to move the arm around our environments even for simple movements. In several cases, the planner algorithms failed to find a plan in a reasonable amount of time or the plan generated makes the robot perform unnecessary movements that might be dangerous to execute for the real arm.

Motion planning is a common problem in robotics and even for the simple problem of finding a path connecting two states while avoiding collisions, there is no efficient solution for the general case. This inspired us to provide the tools to perform rigorous testing of the different planners available and to provide researchers tools to help them find solutions for their specific problems. We use the well-know motion planning framework MoveIt!, that itself does not provide motion planning, but instead is designed to work with planners or planning libraries.

Within MoveIt!, we used the MoveIt! ROS Benchmarks package which provides methods to benchmark motion planning algorithms and aggregate/plot statistics using the OMPL Planner Arena. Here you can follow an example in which they demonstrate how the benchmarking can be run for a Fanuc M-10iA.

We have proposed a series of benchmark problems that are designed to exercise the planners in different scenarios focused specially on common manipulation problems. Different set of queries were defined in each scenario with difficult levels of difficulty with the intention of testing the performance of the planners. Additionally a set of valid random queries were generated in each scenario to validate the results of each planner.

These sets of benchmark problems intend to help the community as a stating point and we encurage researchers to contribute with more scenes and queries that exercise other capabilities of planners.

CHAPTER 1

Contents

1.1 Installing the software

In order to use the Shadow Moveit! Planner Benchmarking, you can use our docker image. Docker is a container framework where each container image is a lightweight, stand-alone, executable package that includes everything needed to run it. It is similar to a virtual machine but with much less overhead. Follow the instructions in the next section to get the latest Docker container up and running.

1.1.1 Hardware specifications

In order to run our software and the ROS software stack you will need to meet some hardware requirements.

- CPU: Intel i5 or above
- RAM: 4GB or above
- Hard Drive: Fast HDD or SSD (Laptop HDD are very slow)
- Graphics Card: Nvidia GPU (optional)
- OS: Ubuntu 18.04, 16.04 Kinetic (Active development)

1.1.2 Create docker container

Pull docker image:

```
docker pull shadowrobot/dexterous-hand:kinetic-sr-benchmarking
```

Create container:

```
docker run -it --privileged --name sr_planner_benchmarking --network=host -e DISPLAY -
↪e QT_X11_NO_MITSHM=1 -e LOCAL_USER_ID=$(id -u) -v /tmp/.X11-unix:/tmp/.X11-unix:rw␣
↪shadowrobot/dexterous-hand:kinetic-sr-benchmarking
```

Note: You don't need to run "docker run" every time as the container is persistent.

To start the container again please execute

```
docker start sr_planner_benchmarking
```

1.2 Motion Planners available in MoveIt!

1.2.1 Currently available in MoveIt!

Currently MoveIt! provide the following planners:

- **Open Motion Planning Library** (OMPL) is a popular choice to solve a motion problem. It is an open-source motion planning library that houses many state-of-the-art sampling based motion planners. OMPL is configured as the default set of planners for MoveIt!. Currently 23 sampling-based motion planners can be selected for use, and they are described here.
- **Stochastic Trajectory Optimization for Motion Planning** (STOMP) is anoptimization-based motion planner. It is designed to plan smooth trajectories for robotic arms, avoiding obstacles, and optimizing constraints. The planner is currently partially supported in MoveIt!
- **Covariant Hamiltonian Optimization for Motion Planning** (CHOMP) this algorithm capitalizes on covariant gradient and functional gradient approaches to the optimization stage to design a motion planning algorithm based entirely on trajectory optimization. Given an infeasible naive trajectory, CHOMP reactsto the surrounding environment to quickly pull the trajectory out of collision while simultaneously optimizing dynamical quantities such as joint velocities and accelerations. It rapidly converges to a smooth collision-free trajectory that can be executed efficiently on the robot. The planner is currently partially supported in MoveIt!
- **Search-Based Planning Library** (SBPL) consists of a set of planners using search-based planning that discretize the space. Integration into latest version of MoveIt! is work in progress.

1.2.2 Selected planners for our benchmarks

Currently only the OMPL planners are well integrated in MoveIt!, so we have only used those in our benchmarks. The following is the set of OMPL planners selected (Description taken form OMPL website):

- Single-query planners
  - Rapidly-exploring Random Trees
    - **RRT**: A well-known Sample Based Planning algorithm, however the plans generated with it are not optimal.
    - **RRT***: A variation of RRT with proven asymptotically optimal property at the cost of execution time and slower path convergence rate.
    - **T-RRT** (Transition-based RRT): It combines the exploration strength of the RRT algorithm that rapidly grow random trees toward unexplored regions of the space, with the efficiency of stochastic optimization methods that use transition tests to accept or to reject a new potential state. It does not give any hard optimality guarantees, but tries to find short, low-cost paths.
    - **RRT Connect**: It works by incrementally building two rapidly-exploring random trees (RRTs) rooted at the start and the goal configurations.
    - **BiTRRT**

  - **EST** (Expansive Space Trees)
  - **BiEST**: Bidirectional version
  - **ProjEST**: Projection-based version
  - **SBL** (Single-query, Bi-directional and Lazy Collision Checking)
  - **KPIECE** (Kinematic Planning by Interior-Exterior Cell Exploration): KPIECE is a tree-based planner that uses a discretization (multiple levels, in general) to guide the exploration of the (continuous) state space. OMPL's implementation is a simplified one, using a single level of discretization: one grid. The grid is imposed on a projection of the state space. When exploring the space, preference is given to the boundary of that part of the grid that has been explored so far. The boundary is defined to be the set of grid cells that have fewer than 2n non-diagonal non-empty neighboring grid cells in an n-dimensional projection space.
  - **BKPIECE** (Bidirectionnal KPIECE)
  - **LBKPIECE** (Lazy BKPIECE)
  - **Fast Marching Tree algorithm (FMT)**: The FMT algorithm performs a "lazy" dynamic programming recursion on a set of probabilistically-drawn samples to grow a tree of paths, which moves outward in cost-to-come space. Unlike all other planners, the numbers of valid samples needs to be chosen beforehand.
  - **Path-Directed Subdivision Trees (PDST)**: PDST is a planner that has entirely removed the dependency on a distance measure, which is useful in cases where a good distance metric is hard to define. PDST maintains a binary space partitioning such that motions are completely contained within one cell of the partition. The density of motions per cell is used to guide expansion of the tree.
  - **Search Tree with Resolution Independent Density Estimation (STRIDE)**: This planner was inspired by EST. Instead of using a projection, STRIDE uses a Geometric Near-neighbor Access Tree to estimate sampling density directly in the state space. STRIDE is a useful for high-dimensional systems where the free space cannot easily be captured with a low-dimensional (linear) projection.

- Multi-query planners
  - **PRM** (Probabilistic roadmap): It takes random samples from the configuration space of the robot, testing them for whether they are in the free space, and use a local planner to attempt to connect these configurations to other nearby configurations. The starting and goal configurations are added in, and a graph search algorithm is applied to the resulting graph to determine a path between the starting and goal configurations.
  - **PRM***: While regular PRM attempts to connect states to a fixed number of neighbors, PRM* gradually increases the number of connection attempts as the roadmap grows in a way that provides convergence to the optimal path.
  - **LazyPRM**: This planner is similar to regular PRM, but checks the validity of a vertex or edge "lazily," i.e., only when it is part of a candidate solution path.
  - **LazyPRM***: A version of PRM* with lazy state validity checking.
  - **SPArse Roadmap Spanner algorithm (SPARS)**: SPARS is a planner that provides asymptotic near-optimality (a solution that is within a constant factor of the optimal solution) and includes a meaningful stopping criterion. Although it does not guarantee optimality, its convergence rate tends to be much higher than PRM*.
  - **SPARS2**: SPARS2 is variant of the SPARS algorithm that works through similar mechanics, but uses a different approach to identifying interfaces and computing shortest paths through said interfaces.

Note: There have been recent updates in MoveIt! that allow the concept of Planning Request Adapters that is used to enable the usage of multiple planning algorithms to be used together in MoveIt. This enables the user to use motion

planning algorithms in a pipeline to produce better trajectories in different situations. It would be nice to test this funcionallity with our benchmarks in the future. More information can be found here.

1.3 Benchmark description

The benchmark problems we define here require the specification of these components:

- Robot model: The particular robot used for the experiments.
- Scenes: The environment to be used for motion planning.
- Queries: A set of initial and goal robot states associated with the scenes.

1.3.1 Robot Model

For our benchmarks, we used the Universal Robots UR10 robot arm. We attached a box as end-effector to simulate the collision issues we will have if a robot hand was attached. The MoveIt! configuration used can be found here.

The robot can be launched using the following command:

```
roslaunch sr_moveit_planner_benchmarking robot.launch
```

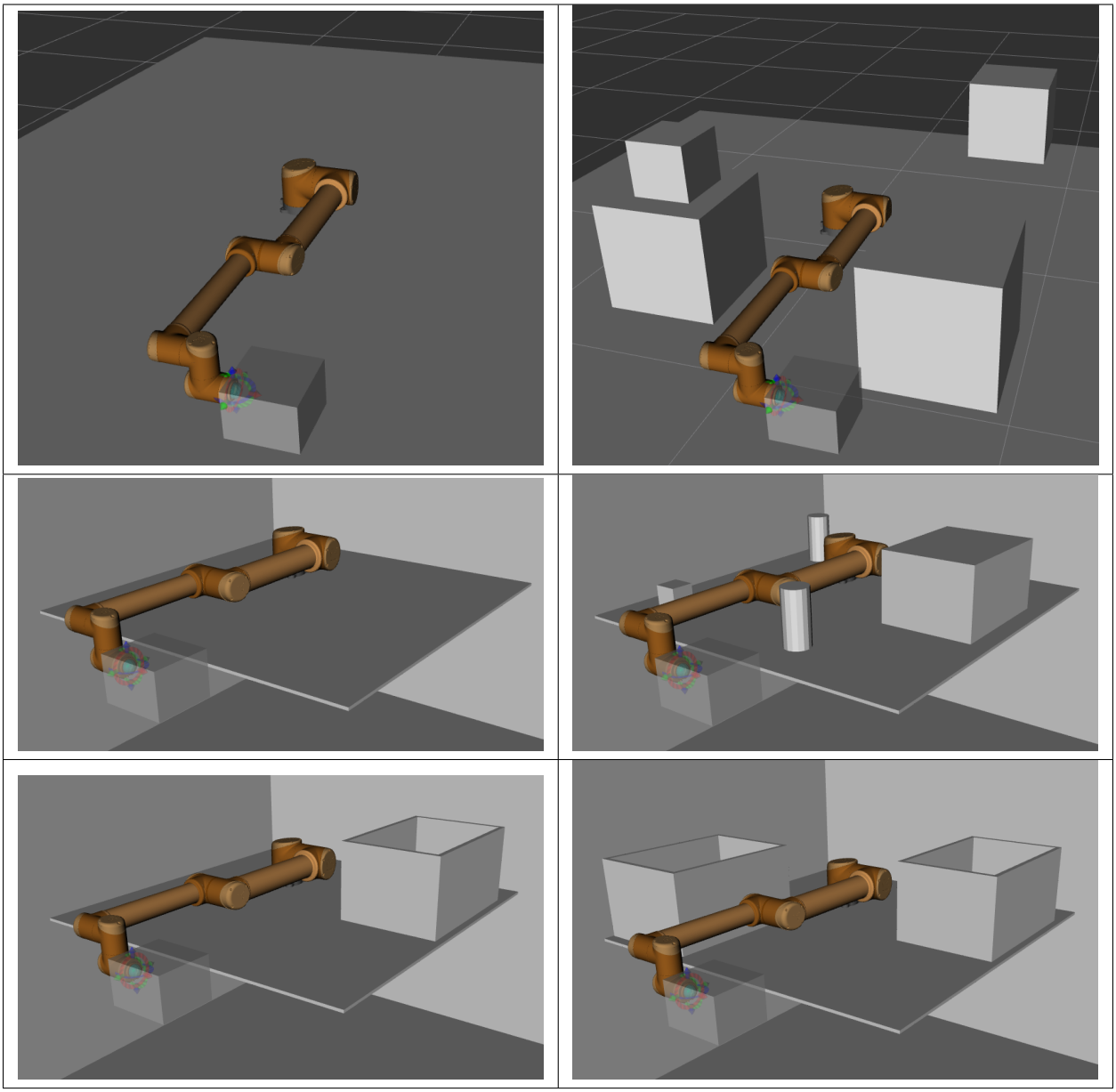
You can add the following arguments:

- initial_z: to move the robot up, such as when it needs to be on the table
- visualization: set this to true to launch rviz, or false to not start rviz
- robot_description: you can change the urdf used in case you want to change your robot or change some urdf values.

1.3.2 Scenes

The scene file format (.scene) is used to define a scene where the robot will be performing the motions. It can be imported, modified and exported using the Motion Planning RViz plugin.

The scenes defined for our benchmarks can be found here and are shown in the following pictures:



1.3.3 Queries

A query is defined as the start and goal state for a robot in a specific scene. We have used Rviz to create a set of queries for each scene with different complexity levels. The queries for each scene can be found here.

1.3.4 Benckmark configuration

Each benchmark has a configuration file in yaml format, containing the following information:

- Details of the warehouse
- Number of runs for each planning algorithm on each request
- The name of the group to plan

- The maximum time for a single run
- The directory to write the output to
- The stored start states in the warehouse to try
- The motion plan queries in the warehouse to try
- The bounds of the workspace the robot plans in.
- A list of planners to benchmark the queries in.

This is an example of the format of a benchmark configurarion yaml file:

```
benchmark_config:
    warehouse:
        host: 127.0.0.1
        port: 33829
        scene_name: scene_ground
    parameters:
        name: scene_ground
        runs: 10
        group: right_arm
        timeout: 10.0
        output_directory: /tmp/moveit_benchmarks/
        queries: .*
    planners:
        - plugin: ompl_interface/OMPLPlanner
          planners:
            - SBLkConfigDefault
            - ESTkConfigDefault
            - LBKPIECEkConfigDefault
            - BKPIECEkConfigDefault
            - KPIECEkConfigDefault
            - RRTkConfigDefault
            - RRTConnectkConfigDefault
            - RRTstarkConfigDefault
            - TRRTkConfigDefault
            - PRMkConfigDefault
            - PRMstarkConfigDefault
```

The configuration for our benchmarks can be found here.

1.3.5 Tools to work with the queries and scenes files

To load the .scene and .queries files into the warehouse, you can use the following methods:

Our high level methods

- To load all our scene and queries to the warehouse:

```
roslaunch sr_moveit_planner_benchmarking load_all_scenes_and_queries_to_db.launch
```

- To load one scene and queries files:

```
roslaunch sr_moveit_planner_benchmarking load_scenes_and_queries_to_db.launch queries_
↪file:=<path_to_file> scene_file:=<path_to_file>
```

- To load just a scene:

```
roslaunch sr_moveit_planner_benchmarking load_scenes_to_db.launch scene_file:=<path_
↪to_file>
```

- To load just a query:

```
roslaunch sr_moveit_planner_benchmarking load_queries_to_db.launch queries_file:=
↪<path_to_file>
```

- To export a scene that is in the warehouse to a .scene text file:

```
roslaunch sr_moveit_planner_benchmarking export_scenes_to_text.launch output_
↪directory:=<path_to_folder_to_save_file>
```

- To export the queries that is in the warehouse to a .queries text file:

```
roslaunch sr_moveit_planner_benchmarking export_queries_to_text.launch output_
↪directory:=<path_to_folder_to_save_file> (group_prefix:=ra) (cartesian:=false)
```

If you want to export the queries in cartesian space (position and orientation) instead of joint space, you can set the cartesian argument to true.

Methods from MoveIt!

The previous methods, internally call a set of MoveIt! methods. You can call the MoveIt! methods directly.

Export to text

An executable is available to allow to export the scene and queries that are available in the Moveit! warehouse to text files:

```
rosrun moveit_ros_warehouse moveit_warehouse_save_as_text
```

It has different parameters that can be specified:

- `help`: Show help message.
- `host`: Host for the DB. Default 127.0.0.1
- `port`: Port for the DB. Default 33829
- `output_directory`: Directory to save the generated files.
- `scene`: Saves the scenes available in the warehouse. It generates .scene files. If it is not specified the queries are saved.
- `cartesian`: Save queries in cartesian space (start and end pose of eef). If not specified they will be saved in joint space by default.
- `eef`: Specify the end effector (Only needed when the cartesian option is set). Default: last link.
- `group_prefix`: Specify the group prefix you'd like to plan with. This is useful if you want to save the queries only for the arm and it has a prefix, so only those joints will be considered (e.g. "ra").

To be able to use this command, the robot should be launched in another terminal first.

Here are a few examples of how to use them:

- Example to export the scenes:

```
rosrun moveit_ros_warehouse moveit_warehouse_save_as_text --scene --output_
↪directory /tmp/scene
```

Example of the exported scene (empty space with ground): example_scene.scene

```
example_scene
* ground__link
1
box
5 5 0.01
-0.2 0.6 0
0 0 0 1
0 0 0 0
.
```

- Example to export the queries in joint space:

```
rosrun moveit_ros_warehouse moveit_warehouse_save_as_text --output_directory /tmp/
↪queries --group_prefix ra
```

Example of exporting a query: example_scene.queries

```
example_scene
example_query
START
ra_shoulder_pan_joint = -0.000150601
ra_shoulder_lift_joint = -0.992381
ra_elbow_joint = 1.73153
ra_wrist_1_joint = 2.46467
ra_wrist_2_joint = -0.000163041
ra_wrist_3_joint = 3.07979
.
GOAL
joint_constraint
ra_shoulder_pan_joint = 2.20941
ra_shoulder_lift_joint = -1.89185
ra_elbow_joint = 1.81825
ra_wrist_1_joint = 0.14342
ra_wrist_2_joint = 2.20853
ra_wrist_3_joint = 0.0420439
.
```

- Example to export the queries in cartesian space:

```
rosrun moveit_ros_warehouse moveit_warehouse_save_as_text --output_directory /tmp/
↪queries --group_prefix ra --cartesian --eef ra_wrist_3_link
```

Example of exporting a query: cartesian_example_scene.queries

```
example_scene
cartesian_query
START
ra_shoulder_pan_joint = 2.64932
ra_shoulder_lift_joint = -1.99619
ra_elbow_joint = -1.29736
ra_wrist_1_joint = 1.87004
ra_wrist_2_joint = -2.11698
```

(continues on next page)

(continued from previous page)

```
ra_wrist_3_joint = 2.63777
.
GOAL
position_constraint
End_effector = ra_wrist_3_link
Position = -0.317122 -0.130014 0.548275
Orientation = 0.0291105 0.158759 0.383943 0.90914
```

Import from text

An executable is available to allow to import .scenes and .queries files to the Moveit! warehouse:

```
rosrun moveit_ros_warehouse moveit_warehouse_import_from_text
```

It has different parameters that can be specified:

- `help`: Show help message
- `host`: Host for the DB. Default 127.0.0.1
- `port`: Port for the DB. Default 33829
- `queries`: Name of file containing motion planning queries.
- `scene`: Name of file containing motion planning scene.

To be able to use this command, the robot should be launched in another terminal first.

Here are a few examples of how to use them:

- Example to import a scene file:

```
rosrun moveit_ros_warehouse moveit_warehouse_import_from_text --scene example_
↪scene.scene
```

- Example to import a query file:

```
rosrun moveit_ros_warehouse moveit_warehouse_import_from_text --queries example_
↪scene.queries
```

Generate random valid queries

An executable is available to allow to the generation of a specified number of valid (collision free) random queries and save them into the Moveit! warehouse:

```
rosrun moveit_ros_warehouse moveit_warehouse_generate_random_queries <name_of_scene>
↪<number_of_random_queries>
```

It has different parameters that can be specified:

- `help`: Show help message.
- `host`: Host for the DB. Default 127.0.0.1
- `port`: Port for the DB. Default 33829
- `limited_joints`: Limit joints from -pi to pi to avoid a lot of impossible queries.

- `group_prefix`: Specify the group prefix you'd like to plan with. This is useful if you want to save the queries only for the arm and it has a prefix, so only those joints will be considered (e.g. "ra").
- `cartesian`: Generate the cartesian space query equivalent to the one generated in joint space.
- `eef`: Specify the end effector (Only needed when the cartesian option is set). Default: last link.
- `clear`: Clears all the random queries for a given scene.

After generating the queries, they are saved in the warehouse. You can use the commands above to export them to text.

Here are a few examples of how to use them:

- Example of generating 10 random queries in joint space in the example_scene:

```
rosrun moveit_ros_warehouse moveit_warehouse_generate_random_queries example_
↪scene 10 --group_prefix ra
```

- Example of generating 10 random queries in joint space in the example_scene limiting the joints:

```
rosrun moveit_ros_warehouse moveit_warehouse_generate_random_queries example_
↪scene 10 --group_prefix ra --limited_joints
```

- Example of generating 10 random queries in cartesian space in the example_scene:

```
rosrun moveit_ros_warehouse moveit_warehouse_generate_random_queries example_
↪scene 10 --group_prefix ra --cartesian --eef ra_wrist_3_link
```

- Example of clearing the random queries in example_scene:

```
rosrun moveit_ros_warehouse moveit_warehouse_generate_random_queries example_
↪scene --clear
```

1.3.6 Benchmark metrics

We use several metrics to perform the benchmarks of the different planners.

Default MoveIt! metrics

These are the metrics defined by default in MoveIt!:

- **Total time (s):** Time taken by the whole process. It is calculated with the following formula: T = plan_time + interpolation_time + simplify_time + process_time *The lower the value, the better performance of the planner.*
- **Solved (%):** Percentage of queries that the planner was able to find a solution for. The higher the value, the better performance of the planner.
- **Correct (%)**: Percentage of solved plans generated by the planner that are correct. Correctness means that path trajectory avoids collisions and all waypoints satisfy given bounds. *The higher the value, the better performace of the planner.*
- **Length (rad):** Calculated by a sum of angles traveled by each of the joints. Formula: L = sumi=0n-1{abs(xi-xi0)}, where: n - number of robot's joints, x - joint's goal position, x0 - joint's initial position. *The lower the value, the better the plan.*
- **Smoothness:** Looks at three consecutive waypoints and the angle formed between them. Value is calculated as a square of sum of all the angles calculated that way. Formula: S = sumi=2n-1{(2 * (pi - arccos((di-2,i-12+di-1,i2-di-2,i2)/(2 * di-2,i-1 * di-1,i))))2}, where: n - number of waypoints on the path, dx,y - distance between waypoints with index x and y. Aligned points result in S = 0. *The lower the value, the smoother plan.*

- **Clearance (m):** Calculated by average distance to nearest invalid state (obstacle) throughout the planned path. Formula: C = (1/n) * sumi=0n-1{cl(si)}, where: n - number of states on path, si - ith state, cl() - distance to the first invalid state. *The higher the value, the better the plan.*

New proposed metrics

We have added two new metrics:

- **Plan quality 1:** Calculated by a weighted sum of angles traveled by each of the joints, giving higher weights to the joints closer to the base of the robot, thus penalizing them as small movements of these joints will result in bigger movements of the end effector. Formula: PQ1 = sumi=0n-1{wi * abs(xi - xi0)}, where: n - number of robot's joints, w - weight specified for each joint, x - joint's goal position, x0 - joint's initial position. *The lower the value, the better the plan.*
- **Plan quality 2:** Measures how different is the total distance traveled by the arm end effector to the most direct path between the start and goal position. Formula: PQ2 = (dt/dl + rt/rl)/2, where: dt - total distance travelled by the end effector, dl - distance travelled by the end effector in a straight line, rt - total rotation done by the end effector, rl - relative rotation of end effector between start and goal pose. *The lower the value, the better the plan with 1 being the best possible score.*

1.4 Running a benchmarking experiment

1.4.1 Executing the benchmarks

After defining the scenes, queries and benchmark configurations, you are ready to execute the benchmarks. To do so, execute the following launch file:

```
roslaunch sr_moveit_planner_benchmarking benchmarking.launch bench_opts:=<path_to_
↪benchmark_config_file>
```

You can also specify the initial_z of the robot to move it on top of the table.

We have created launch files for each of our scenes that you can find here. For example, to run the benchmarks for the scene 'ground_with_boxes', execute this:

```
roslaunch sr_moveit_planner_benchmarking benchmark_ground_with_boxes.launch
```

1.4.2 Generating database files from log files

After the benchmarks are executed, they generate a set of log files (one per query). If you want to convert them to sqlite3 database files that are easy to handle, run the following command:

- All files

```
rosrun sr_moveit_planner_benchmarking sr_moveit_planner_convert_to_db.py -a /tmp/
↪moveit_benchmarks/
```

- Sorted by scenes:

```
rosrun sr_moveit_planner_benchmarking sr_moveit_planner_convert_to_db.py -a /tmp/
↪moveit_benchmarks/ --sort
```

1.5 Visualizing the results

After the benchmarks are executed, the different metrics are computed and written to a logfile. MoveIt! supply a script moveit_benchmark_statistics.py to parse this data and plot the statistics.

```
rosrun moveit_ros_benchmarks moveit_benchmark_statistics.py <path_of_logfile>
```

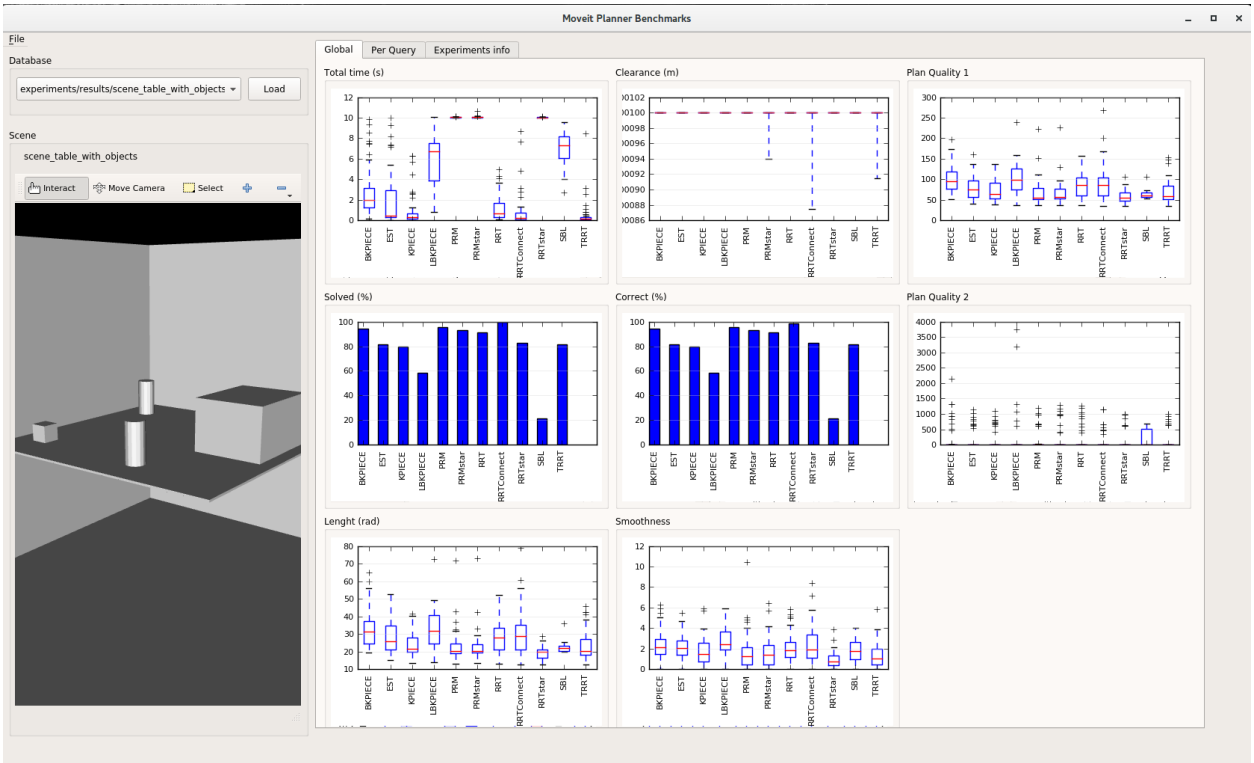
To generate a PDF of plots:

```
python -p <plot_filename> moveit_benchmark_statistics.py <path_of_logfile>
```

Alternatively, you can upload the database file generated by moveit_benchmark_statistics.py to planner arena and interactively visualize the results.

However, that was generated for each experiment and each graph was located in an entire page which make it difficult to compare the results. Therefore, we have improved the visualization of the results with the **Benchmarks Visualizer**, a graphical user interface that:

- Allows you to choose the benchmark database to visualize
- Displays the scene in rviz
- Shows information about the experiments performed on the selected database and also information about the benchmark metrics used in the graphs.
- Displays several graphs in one screen summarizing the results for each planner
- Displays the results per each query selecting a specific planner which will be useful to analyze the output of the different quality metrics



To run the benchmark visulizer, execute the following command:

```
roslaunch sr_moveit_planner_benchmarking benchmarks_visualizer.launch
```