Appears in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2022

Robowflex: Robot Motion Planning with *MoveIt* Made Easy

Zachary Kingston Lydia E. Kavraki

***Abstract*— *Robowflex* is a software library for robot motion planning in industrial and research applications, leveraging the popular MoveIt library and Robot Operating System (ROS) middleware. *Robowflex* provides an augmented API for crafting and manipulating motion planning queries within a single program, making motion planning with MoveIt easy. *Robowflex*'s high-level API simplifies many common use-cases while still providing low-level access to the MoveIt library when needed. *Robowflex* is particularly useful for 1) developing new motion planners, 2) evaluating motion planners, and 3) complex problems that use motion planning as a subroutine (e.g., task and motion planning). *Robowflex* also provides visualization capabilities, integrations to other robotics libraries (e.g., DART and Tesseract), and is complementary to other robotics packages. With our library, the user does not need to be an expert at ROS or MoveIt to set up motion planning queries, extract information from results, and directly interface with a variety of software components. We demonstrate its efficacy through several example use-cases.**

I. Introduction

A core component of any autonomous system is *motion planning* [1]–[3], which finds feasible motions that satisfy task requirements (e.g., reaching the goal, satisfying some motion constraint, etc.). There are many motion planning software system for general manipulators; a popular library for motion planning is MoveIt [4], which is built on top of the ubiquitous Robot Operating System (ROS) framework [5]. MoveIt has four key advantages: it is widely adopted in industry and research, it is easy to setup for new robots and over 150 robots are already available [6], it is easy to integrate with a ROS system, and it has a large and vibrant open source community. However, due to MoveIt's massive scope and abstract architecture, many tasks are challenging for both engineers and researchers. For example, it can be difficult to evaluate and develop planning algorithms, extend a planner's functionality, extract low-level information from planners, or use a planner within the scope of a broader planning algorithm, e.g., task and motion planning [7].

This paper introduces *Robowflex*, a software library designed to simplify the use of MoveIt for industrial and research applications of motion planning. *Robowflex* is a high-level API to easily manipulate robots, collision environments, planning requests, and motion planners. *Robowflex* "wraps" the underlying MoveIt library within a C++ interface that provides many utilities that simplify the use and evaluation of motion planners. Moreover, *Robowflex* provides direct access to the implementation (that is, not through ROS messaging). The key advantage of this approach is the ability to 1) develop self-contained scripts to evaluate motion planning, 2) retain the capability of easy system integration through ROS when necessary, and 3) implement integrated algorithms that use motion planning extensively (e.g., task and motion planning). *Robowflex* also integrates other libraries, e.g., the Open Motion Planning Library (OMPL) [8], DART [9], ROS Industrial's Tesseract [10], and visualization with Blender [11]. For example, these integrations enable comparison of MoveIt's RRTConnect against Tesseract's TrajOpt on the same scene in a single, short script. We demonstrate the usefulness of *Robowflex* in several example use-cases, such as benchmarking for motion planning experiments and an industry-focused use-case of evaluating Robonaut 2 walking from NASA. *Robowflex* is open-source[1], documented online[2], and has already been used in a number of publications [12]–[22].

Department of Computer Science, Rice University, Houston, TX, 77005, USA, {zak,kavraki}@rice.edu. This work was supported in part by NSF 1718478, NSF 2008720, NSTRF 80NSSC17K0163, and Rice University Funds.

II. Background

Robowflex is built around MoveIt [4], a widely-used[3] software library designed to provide motion planning to ROS [5] enabled robots. MoveIt has been successfully used with many robots, such as the PR2 [23], Fetch [24], and NASA's Robonaut 2 [25]. MoveIt also provides a setup assistant to easily configure new robots for motion planning [6]. MoveIt provides default motion planning plugins such as sampling-based motion planning [1]–[3] through OMPL [8] and trajectory optimization (e.g., [26], [27]).

Typically[4], users interact with MoveIt through the provided MoveGroup program, which leverages MoveIt's plugin-based architecture to provide a flexible, configurable motion planning service. While convenient for basic motion planning, MoveGroup falls short when used outside this scope. For more advanced applications of motion planning, such as profiling or evaluating detailed aspects of a motion planner, changing parameters or components of a planner, or extracting more information from planners, it is insufficient. To edit or improve the capabilities of MoveIt, users must either edit or create plugin classes, which can be complicated for those who are unfamiliar with MoveIt's internal structure. Moreover, if users just want to use MoveIt as a library, there are many parameters to load and configure (e.g., URDF, SRDF, joint limits, kinematics plugins, planner configurations, etc.). *Robowflex* provides a MoveGroup-like interface as a high-level API, while also providing full access to underlying

[1] https://github.com/KavrakiLab/robowflex
[2] https://kavrakilab.github.io/robowflex/

data, enabling access to these underlying structures to either modify or inspect[5].

Robowflex aims to support motion planning research, such as developing new *sampling-based algorithms* [1]–[3]. A popular motion planning library used by many frameworks is the Open Motion Planning Library (OMPL) [8], and support of OMPL is paramount for many benchmarking applications [28]. However, augmenting or customizing OMPL is difficult for real robotic systems, due to the difficulty of accessing the internals of the library through interfaces such as MoveIt. *Robowflex* provides low-level access to OMPL, which is essential for rapid development and testing of novel motion planning techniques. Moreover, it is possible to develop novel planners within the *Robowflex* framework that are not tied to MoveIt or OMPL.

Additionally, *Robowflex* was designed to support algorithms that use motion planning as a subroutine. For example, motion planning is used within *task and motion planning* (TAMP), e.g., [7], [29], [30]. TAMP combines AI task planning with motion planning, generating a sequence of valid actions the robot should accomplish, each requiring a motion plan. There has been work in integrating TAMP-like approaches in MoveIt, e.g., [31], however this approach only finds motions for predetermined sequences of actions. Critically, TAMP requires both evaluating many motion plans to validate actions as well as using feedback from motion planning to inform task planning. These requirements involve a deeper introspection into motion planning, which is at odds with the design of MoveGroup.

Beyond MoveIt, there are many excellent motion planning frameworks that are compatible with ROS through wrappers. OpenRAVE [32] is a fully featured motion planning framework. However, it is not actively developed and does not have the level of community support available in MoveIt and ROS. Similarly, KLAMP't [33] is an all-in-one toolbox that specializes in handling contact, providing its own environment with ROS adapters. The Robotics Library [34] is an all-in-one library for real-time planning, but lacks the ability to connect to the broader robotics ecosystem through ROS. As stated above, the benefit of using MoveIt over the above libraries is that MoveIt is easy to integrate with new robots, easy to integrate with ROS, and is actively developed with a large community.

III. The *Robowflex* Library

Robowflex is intended for use in research, education, and industry. *Robowflex* is informed by the following design goals.

1) *Clarity of Interface*: Provide an easy to understand interface that meshes with intuitive understanding of the concepts involved in motion planning.
2) *Minimize* ROS: Encapsulate ROS as much as possible such that it is easy to write independent programs.
3) *Leverage* MoveIt*'s Ubiquity*: By being based on MoveIt, many robots are supported in *Robowflex* out of the box. The use of MoveIt also enables *Robowflex* scripts to effortlessly connect to the greater ROS system.
4) *Unrestricted Access and Integration*: While providing a high-level API, give access to underlying libraries so users are not hampered by *Robowflex* in any way. A key advantage *Robowflex* provides is that all the underlying data structures used in MoveIt, OMPL, or other libraries can be accessed and modified. *Robowflex* provides a common interface and adapters to convert between the representations used by each library.
5) *Consistency Across Versions*: *Robowflex* provides adapters such that *Robowflex* code is not tied to a specific version of ROS or MoveIt. *Robowflex* supports all versions from ROS Indigo to ROS Noetic.

At a high-level, *Robowflex* provides wrappers around core MoveIt concepts so that it is easy to create and manage robots, scenes, and planners within a script. The primary building blocks of *Robowflex* are the robot's kinematics, the collision environment, and the motion planner. Although other facades to MoveIt exist, *Robowflex* provides a higher level of abstraction that allows users who are unfamiliar with MoveIt to still achieve complex programming tasks. Moreover, these core features are supported by a suite of utilities, such as input-and-output helpers for a variety of formats, benchmarking tools, visualization within RViz, and more. Beyond the core library are auxiliary modules for other robotics libraries, and support seamless integration between *Robowflex* and native formats.

A. Input and Output

Many complicated robotic problems require large amounts of file input and output (IO), e.g., for configuration, loading problems and scenes, and more. *Robowflex* provides many IO helpers for common file-types used in robotics, such as:

a) Files and ROS *Packages:* It is common to access files that exist in ROS packages that are specified with package URIs—*Robowflex* provides helper functions to resolve file paths and to either open or create said file.

b) XML *Files:* Many configuration files in robotics are written in Extensible Markup Language (XML) or in the macro XML language XACRO. *Robowflex* provides helper functions to load these files (including unprocessed XACRO) as well as inject changes on the fly.

c) YAML *Files:* It is common to save and load ROS messages as YAML files. However, there is no easy way to load or save ROS messages in YAML in C++. *Robowflex* has broad YAML support, and can load YAML from ROS Python and ROS topics. Many *Robowflex* classes can load ROS messages, e.g., motion planning requests, robot states, and more. This makes it easy to load problems using YAML files.

[3] https://moveit.ros.org/moveit/ros/2020/07/24/moveit-research-roundup.html

[4] https://ros-planning.github.io/moveit_tutorials/

[5] Note that *MoveIt*CPP (https://ros-planning.github.io/moveit_tutorials/doc/moveit_cpp/moveitcpp_tutorial.html) and https://github.com/PickNikRobotics/moveit_boilerplate provide simple interfaces to load MoveIt structures internally, but lack the abstraction, isolation, IO support, and support modules provided by *Robowflex*.

```
namespace rx = robowflex;

auto wam7 = //
  std::make_shared<rx::Robot>("wam7");
wam7→initialize( //
  "package://barrett_model/robots/
      wam_7dof_wam_bhand.urdf.xacro",  // urdf
  "package://barrett_wam_moveit_config/config/
      wam7_hand.srdf",      // srdf
  "package://barrett_wam_moveit_config/config/
      joint_limits.yaml",  // joint limits
  "package://barrett_wam_moveit_config/config/
      kinematics.yaml"      // kinematics
);
```

Fig. 1. Loading a robot (here, a Barrett WAM® arm) in *Robowflex*.

d) ROS Parameters: Many ROS programs rely on the parameter server, a distributed key-value store available in ROS. As a result, it is sometimes difficult to have multiple programs running simultaneously that require similar parameters, leading to issues with managing namespaces. By default, *Robowflex* uses an anonymous namespace so that many instances of *Robowflex* code can simultaneously run. Moreover, there is support to load YAML files onto the parameter server, which is typically only available through ROS launch, making it easy to have scripts load their parameters.

There is also support for ROS bag files, HDF5 files, various transformation representations, and others.

B. Robot Kinematics

Commonly, the kinematics and geometry of a robot are described using the Universal Robot Description Format (URDF). Moreover, MoveIt requires a Semantic Robot Description Format (SRDF) file, which describes additional properties such as what links are allowed to collide with each other, what groups of joints should be used for planning, and what parts of the robot are the end-effector. There are also additional configuration files MoveIt requires, such as YAML files that describe kinematics plugins and additional joint limits. Typically, most robots are configured through the MoveIt setup assistant wizard which generates a ROS package containing all necessary configuration files. Part of this configuration are complicated ROS launch files that contain parameters necessary to start MoveGroup, and in order to modify the behavior of MoveIt, both the program and launch file need to be modified.

Robowflex takes care of all the necessary legwork to load and configure a robot without the need for a launch file. Moreover, *Robowflex* enables loading multiple robots within the same program, and duplicating robots if necessary—with default MoveIt, this process can become convoluted. An example of loading a robot is shown in Figure 1.

C. Collision Environment

The robot can be used to initialize a planning scene, which contains the collision geometry of the environment. The scene can be used for adding and moving collision objects and computing collisions and distance to collision. For example, scenes can be loaded from YAML files, which encode full planning scenes:

```
auto scene = std::make_shared<rx::Scene>(wam7);
scene→fromYAMLFile( //
  "package://robowflex_library/yaml/scene.yml");
```

Scenes also support adding collision objects programatically:

```
auto scene = std::make_shared<rx::Scene>(wam7);
auto geometry = //
  rx::Geometry::makeCylinder(0.025, 0.1);
auto pose = //
  rx::TF::createPoseXYZ( //
    -0.268, -0.826, 1.313,     // position
        0.,     0.,    0.));  // XYZ Euler
scene→updateCollisionObject( //
  "cylinder", geometry, pose);
```

Note that many scenes can be loaded simultaneously, can be copied and modified, and saved and loaded to and from disk.

D. Motion Planner

MoveIt uses a plugin-based system to load motion planning pipelines[6], which consist of adapters that filter and process both the planning request and output trajectory found by a motion planner. *Robowflex* provides an implementation to access any pipeline, and helpers for common plugins such as the default OMPL planning pipeline plugin. To specify a planning request, a helper class is provided which simplifies the design of complex goal and path constraints, as well as setting start and goal states. This helper class can also save and load motion planning requests to YAML files, for later evaluation or setup. Many planners can be loaded simultaneously and used in tandem. Moreover, *Robowflex* supports inserting new planners, either through MoveIt's API or its own.

E. Example Script

Figure 2 shows a simple script for motion planning with *Robowflex* using the Fetch robot. The robot is loaded on line 2—*Robowflex* comes with some preconfigured robots. An empty planning scene for the robot is created on line 6. The standard OMPL planner for the Fetch is created and initialized in lines 9 to 12. A simple request, which unfurls the Fetch's arm from the stow position to an extended position, is created in lines 15 to 31. Finally, motion planning occurs on line 34. A version of this script is available in the repository[7].

This simple script is akin to basic planning using the MoveIt's `MoveGroupInterface` class[8]. The critical difference is that, rather than having to use `roslaunch` to run an instance of the MoveGroup program and then communicate plans over ROS messages, all of MoveIt's

[6] https://ros-planning.github.io/moveit_tutorials/doc/motion_planning_pipeline/motion_planning_pipeline_tutorial.html

[7] https://github.com/KavrakiLab/robowflex/blob/master/robowflex_library/scripts/fetch_test.cpp

[8] https://ros-planning.github.io/moveit_tutorials/doc/move_group_interface/move_group_interface_tutorial.html

```cpp
// Create a default Fetch robot.
auto fetch = std::make_shared<rx::FetchRobot>();
fetch->initialize();

// Create an empty scene.
auto scene = std::make_shared<rx::Scene>(fetch);

// Create the default planner for the Fetch.
auto planner = //
  std::make_shared< //
    rx::OMPL::FetchOMPLPipelinePlanner>(fetch);
planner->initialize();

// Create a motion planning request.
rx::MotionRequestBuilder request( //
  planner, "arm_with_torso");

// Set the start state.
fetch->setGroupState("arm_with_torso",
  {0.05, 1.32, 1.40, -0.2, 1.72, 0, 1.66, 0});
request.setStartConfiguration( //
  fetch->getScratchState());

// Set the goal state.
fetch->setGroupState("arm_with_torso",
  {0.27, 0.5, 1.28, -2.27, 2.24, -2.77, 1, -2});
request.setGoalConfiguration( //
  fetch->getScratchState());

// Set the desired planner.
request.setConfig("RRTConnect");

// Do motion planning!
auto result = planner->plan( //
                scene, request.getRequest());
```

Fig. 2. A code snippet demonstrating basic motion planning on a Fetch robot [24], from a "stow" position of the arm to an "unfurled" position. Note this does not use any ROS messages, similar to the internals of MOVEIT's MOVEGROUP program.

internal structures are loaded in the *Robowflex* program[9]. Providing access to these structures within a single program is a key benefit of *Robowflex*. The scripting paradigm offered by *Robowflex* is more amenable to rapid testing and scripting than MOVEGROUP, which is designed primarily to be a "live" component of a ROS system extant in some world.

F. Integrations

Robowflex also provides a number of auxiliary modules that enable compatibility with different robotics libraries.

a) OMPL *Integration:* The *Robowflex* OMPL module provides deeper access for motion planning to the default MOVEIT OMPL motion planning plugin. This includes extracting the underlying OMPL setup for a given planning problem, which enables users to modify the behavior of OMPL planning without having to recompile either the MOVEIT planning plugin or OMPL. An example script for how to extract and customize the underlying OMPL planner used by MOVEIT is shown in Figure 3. With *Robowflex*, it easy to use a custom planner or feature in OMPL, as compared to

[9]See https://ros-planning.github.io/moveit_tutorials/doc/motion_planning_api/motion_planning_api_tutorial.html for how this could be done without *Robowflex*. Also, see footnote 5 for other helper classes that can set up planning without *Robowflex*.

```cpp
// Create an OMPL planner
auto planner = //
  std::make_shared< //
    rx::OMPL::OMPLInterfacePlanner>(fetch);

// Extract underlying OMPL structures
auto context = //
  planner->getPlanningContext(scene, request);
auto ss = context->getOMPLSimpleSetup();

// Customize OMPL planner
ss->setPlanner(...);
auto space = ss->getStateSpace();
space->setStateSamplerAllocator(...);
```

Fig. 3. A code snippet demonstrating how to use the *Robowflex* OMPL integration to access internal OMPL features.

integrating a new planner into MOVEIT (e.g., this was done in [12]–[14]).

b) DART *Integration:* The *Robowflex* DART module provides an alternative to MOVEIT, by modeling robots and scenes in the DART [9] framework with bidirectional conversion to/from MOVEIT constructs. The module provides an implementation of motion planning through OMPL, including motion planning with manifold constraints [15]. Moreover, this module provides an easy way to plan for multi-robot systems, allowing arbitrary composition of MOVEIT enabled robots. This capability was used by the multi-robot task-motion planning framework discussed in subsection IV-B. An example script demonstrating the DART module is given shown in Figure 4.

c) Tesseract Integration: The *Robowflex* Tesseract module provides access to the ROS Industrial Consortium's planning framework [10], which includes an implementation of the TrajOpt planner [27]. Similar to the DART module, methods for converting data (e.g., scenes and plans) back and forth are provided. *Robowflex*'s Tesseract module was used in [16] to access and modify TrajOpt for comparison against OMPL planners.

d) MOVEGROUP *Integration:* The *Robowflex* MOVEGROUP module provides an easy connection to a live instance of MOVEIT's MOVEGROUP. This connection can be used to obtain the current collision environment, publish plans to be executed, and generally synchronize information between MOVEGROUP and a *Robowflex* script.

e) Visualization with Blender: The *Robowflex* visualization module makes it easy to render robots within Blender [11], a tool for 3D modeling and animation. An example rendered still is shown in Figure 5. Moreover, it is easy to animate motion plans generated by *Robowflex* to create appealing visualizations and videos[10]. This module has also been used to generate figures in [12], [13], [15].

IV. EXAMPLE USE-CASES

Crucial to the particular use-cases listed here as well as many others, *Robowflex* simplifies the pipeline of creating,

[10]https://kavrakilab.github.io/robowflex/md__home_runner_work_robowflex_robowflex_robowflex_visualization_README.html

```cpp
namespace rd = robowflex::darts;

// Convert MoveIt robot to Dart
auto fetch1 = std::make_shared<rd::Robot>(fetch);

// Copy the Fetch for multi-robot planning
auto fetch2 = fetch1->cloneRobot("other");
fetch2->setDof(4, 1); // Offset on X-axis

// Combine kinematic structures into a world
auto world = std::make_shared<rd::World>();
world->addRobot(fetch1);
world->addRobot(fetch2);

// Plan using planning groups from both robots
rd::PlanBuilder builder(world);
builder.addGroup("fetch", "arm_with_torso");
builder.addGroup("other", "arm_with_torso");

// Set start configuration for both robots
builder.setStartConfiguration({ //
  0.05, 1.32, 1.4, -0.2, 1.72, 0, 1.66, 0,
  0.05, 1.32, 1.4, -0.2, 1.72, 0, 1.66, 0
});
builder.initialize();

// Set goal configuration and setup planning
auto goal = builder.setGoalConfiguration({ //
  0.27, 0.5, 1.28, -2.27, 2.24, -2.77, 1, -2,
  0.27, 0.5, 1.28, -2.27, 2.24, -2.77, 1, -2
});
builder.setGoal(goal);
builder.setup();

// Solve using OMPL
auto result = builder.ss->solve(30.0);
```

Fig. 4. A code snippet demonstrating *Robowflex*'s DART module for multi-robot motion planning. Here, the Fetch robot that was loaded in the prior script (Figure 2) is converted to DART and copied, created a multi-robot system. A plan is generated in the composite space of both robots.

editing, inspecting, saving, and loading motion planning problems (both collision environments and motion planning requests). These capabilities are essential for many research and industrial tasks that rely on reproducibility, experimentation, and debugging.

A. Benchmarking Motion Planners

A core use-case for *Robowflex* is benchmarking motion planners in a variety of planning scenes. The task of evaluating motion planners on realistic robots in many environments, extracting detailed planner information, and collating all collected data is difficult. *Robowflex* provides a benchmarking tool that enables easy benchmarking of different planners, scenes, and requests. The tool enables configurable benchmark output in a number of formats. For example, a default format provided is the OMPL benchmark log output, so output is compatible with the standard OMPL benchmarking suite and tools [28]. In addition, recall that *Robowflex* supports loading both environments and requests from disk, making it easy to craft datasets for evaluation. As new planners are developed, targeted benchmarking can be done for many planner properties, a contribution to the motion planning community due to the difficulty of setting up consistent benchmarking criteria. For example, the following

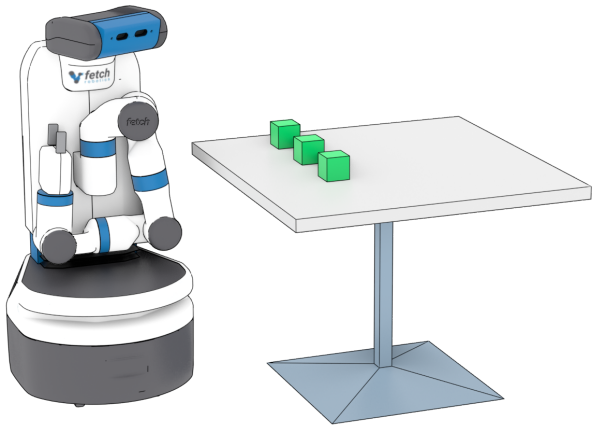

Fig. 5. A Fetch robot [24] and planning scene rendered in Blender using *Robowflex*'s visualization module.

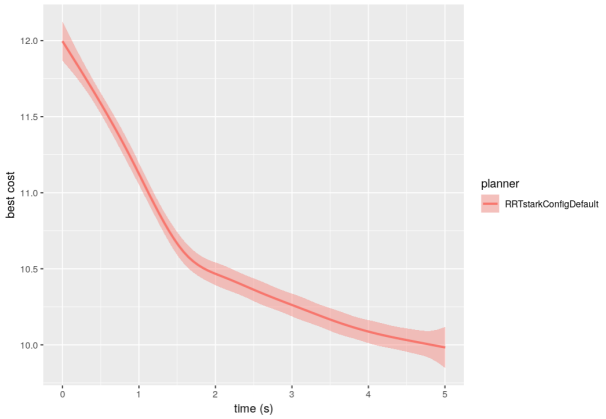

Fig. 6. Example benchmarking results from 200 runs of RRT* [35], an asymptotically optimal algorithm, on the Fetch. The "best cost" path (here, shortest path length) is displayed over time. This progress property of the planner is captured by *Robowflex*'s benchmarking, using the planner from the OMPL module. This plot was generated with Planner Arena[12] [28], which accepts OMPL benchmark output for interactive plotting.

could be added after line 31 of Figure 2 to benchmark the motion planning request:

```cpp
rx::Profiler::Options options;
rx::Experiment experiment( //
  "example",  // Name of experiment
  options,    // Options for internal profiler
  60.0,       // Query timeout
  100);       // Number of trials

experiment.addQuery("planner",
  scene, planner, request->getRequest());

auto *dataset = experiment.benchmark();
rx::OMPLPlanDataSetOutputter output( //
  "benchmark_example");
output.dump(*dataset);
```

Moreover, benchmarking can capture *progress properties* of a motion planner, if properly exposed. These properties are important for profiling the performance of asymptotically (near-)optimal motion planning algorithms, such as RRT* [35]. An example is shown in Figure 6. *Robowflex*'s

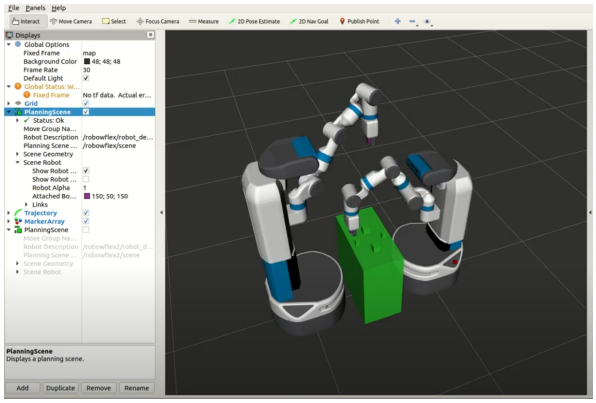

Fig. 7. Two Fetch robots [24] displayed in RViz executing a task and motion plan (TAMP) to rearrange blocks. The motion planning component of this TAMP problem is done through the DART module of *Robowflex*. *Robowflex* enables the TAMP algorithm to have integrated access to the motion planner, which allows for trying many queries simultaneously, extracting collision information, and more. Image courtesy of Tianyang Pan.

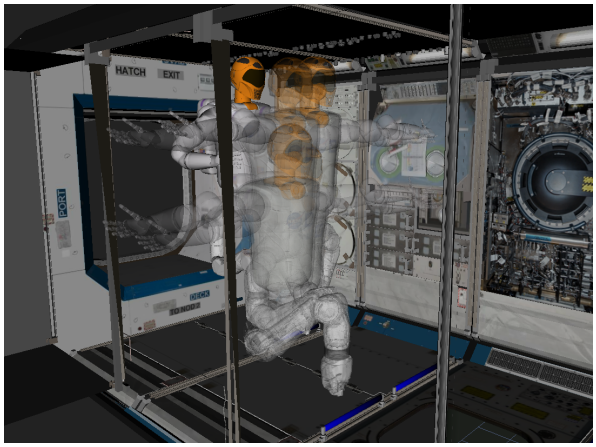

Fig. 8. NASA's Robonaut 2 inside of a module of the International Space Station, visualized in RViz. For a given step between handrails, many possible configurations are evaluated through *Robowflex*. Image courtesy of Misha Savchenko and NASA.

benchmarking was used in [12]–[17], [19].

Compared to MOVEIT's built in benchmarking capabilities[13], *Robowflex* provides a self-contained means of benchmarking that is more easily extendable. MOVEIT's benchmarking requires use of ROS Warehouse for constructing benchmark datasets as opposed to *Robowflex*'s simple file storage, and does not easily support planning scenes with obstacle variation, which is important for learning-based methods. For example, [13], [19] take advantage of these two features by running *Robowflex* benchmarking instances in containers federated over many machines. Moreover, *Robowflex* enables creation of custom metrics with access to underlying planner results (e.g,. progress properties are not available in MOVEIT, properties of a particular method), and can be run as a single script.

B. Task and Motion Planning

One of the strengths of *Robowflex* is motion planning in isolation. That is, being able to use many different instances of robots, scenes, and motion planners all within the same script. This is essential to efficient task and motion planning (TAMP), as a TAMP algorithm will evaluate many different motion plans in a variety of scenes to find a feasible task-and-motion plan. *Robowflex* has been used as the motion planning component in a few TAMP algorithms (e.g., [18], [21]), one of which is shown in Figure 7. Here, *Robowflex* and *Robowflex*'s DART module are leveraged to provide the motion planning components necessary for a multi-robot TAMP algorithm. Crucial to TAMP is the evaluation of many possible scene configurations—*Robowflex* allows for many copies of the collision environment to be considered in parallel. Moreover, *Robowflex* enables information on planning progress to be extracted from the underlying planner, which is used to inform the task planner. Finally, the DART module allows for multi-robot planning, as shown in Figure 4.

C. Robonaut 2 and NASA

Robowflex has also been used by NASA for motion planning for Robonaut 2. Robonaut 2 is a highly dexterous system, with many degrees of freedom. One of the many motion planning challenges Robonaut 2 faces is climbing across handrails in the International Space Station, as shown in Figure 8. To this end, *Robowflex* was used to evaluate potential handrail grasps and the difficulty of motion planning between different grasps to automate walking across the station. *Robowflex* provides the means to use, inspect, and evaluate custom inverse kinematics solvers for Robonaut 2 and to benchmark the variety of handrail grasp configurations and scenes. Additionally, *Robowflex* was used for the Robonaut 2 experiments and figure in [15]. As demonstrated by the examples presented here, the affordances provided by *Robowflex* are general and broadly useful to different members of the robotics community.

V. DISCUSSION

We have presented *Robowflex*, a C++ library that enables the use of MOVEIT in an easier, more flexible way for the creation of advanced robot software for industry, research, and education. The core advantage that *Robowflex* provides over the default distribution of MOVEIT is the ability to easily access and modify core data structures within the program itself, rather than through ROS messages to the provided MOVEGROUP program. This also enables the use of motion planning within more complex algorithms, such as task and motion planning approaches. Moreover, *Robowflex* provides a high-level API, enabling many use-cases such as benchmarking and motion planning without any ROS or MOVEIT expertise required. *Robowflex* also has a number of auxiliary modules that provide access to other robotics libraries and visualization tools, such as OMPL, DART,

[12] http://plannerarena.org/

[13] https://ros-planning.github.io/moveit_tutorials/doc/benchmarking/benchmarking_tutorial.html

Tesseract, and Blender. Beyond motion planning software development, we hope that *Robowflex* will enable broader application and adoption of motion planning algorithms and raise the level of experimental evaluation in comparisons.

Acknowledgments

We thank Constantinos Chamzas, Carlos Quintero-Peña, Andrew Wells, Wil Thomason, Bryce Willey, Juan David Hernández, Mark Moll, and the other members of the Kavraki Lab. We also thank Misha Savchenko, Julia Badger, and the rest of the Robonaut 2 team at NASA Johnson Space Center.

References

[1] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, 2005.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] L. E. Kavraki and S. M. LaValle, *Springer Handbook of Robotics*, 2nd ed. Springer, 2016, ch. Motion Planning, pp. 139–162.

[4] I. A. Şucan and S. Chitta. (2011) *MoveIt!* [Online]. Available: http://moveit.ros.org

[5] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *Wksp. on Open Source Software at ICRA*, vol. 3, 2009.

[6] D. Coleman, I. A. Şucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a MoveIt! case study," *J. of Software Engineering for Robotics*, vol. 5, no. 1, pp. 3–16, 2014.

[7] N. T. Dantam, Z. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *Int. J. of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[8] 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.

[9] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *J. of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[10] ROS-Industrial. (2017) Tesseract. [Online]. Available: https://github.com/ros-industrial-consortium/tesseract

[11] Blender Online Community. (2020) Blender—a 3D modelling and rendering package. Blender Foundation. [Online]. Available: http://www.blender.org

[12] C. Chamzas, A. Shrivastava, and L. E. Kavraki, "Using local experiences for global motion planning," in *IEEE Int. Conf. Robot. Autom.*, 2019, pp. 8606–8612.

[13] 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.*, 2021, pp. 1283–1289.

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

[15] Z. Kingston, M. Moll, and L. E. Kavraki, "Exploring implicit spaces for constrained sampling-based planning," *Int. J. of Robotics Research*, vol. 38, no. 10–11, pp. 1151–1178, 2019.

[16] C. Quintero-Peña, A. Kyrillidis, and L. E. Kavraki, "Robust optimization motion planning for high-DoF robots under sensing uncertainty," in *IEEE Int. Conf. Robot. Autom.*, 2021.

[17] J. D. Hernández, M. Moll, and L. E. Kavraki, "Lazy evaluation of goal specifications guided by motion planning," in *IEEE Int. Conf. Robot. Autom.*, 2019, pp. 944–950.

[18] A. M. Wells, Z. Kingston, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Finite horizon synthesis for probabilistic manipulation domains," in *IEEE Int. Conf. Robot. Autom.*, 2021.

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

[20] S. Sobti, R. Shome, S. Chaudhuri, and L. E. Kavraki, "A sampling-based motion planning framework for complex motor actions," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2021.

[21] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki, "A general task and motion planning framework for multiple manipulators," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2021, pp. 3168–3174.

[22] C. Chamzas, C. Quintero-Peña, 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 Robot. Autom. Letters*, vol. 7, no. 2, pp. 882–889, 2022.

[23] K. A. Wyrobek, E. H. Berger, H. M. Van der Loos, and J. K. Salisbury, "Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot," in *IEEE Int. Conf. Robot. Autom.*, 2008, pp. 2165–2170.

[24] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and Freight: Standard platforms for service robot applications," in *Wksp. on Autom. Mobile Service Robots*, 2016.

[25] W. Baker, Z. Kingston, M. Moll, J. Badger, and L. E. Kavraki, "Robonaut 2 and you: Specifying and executing complex operations," in *IEEE Wksp. on Advanced Robot. and its Social Impacts*, 2017.

[26] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning," *Int. J. of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[27] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *Int. J. of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

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

[29] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *IEEE Int. Conf. Robot. Autom.*, 2011, pp. 1470–1477.

[30] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE Int. Conf. Robot. Autom.*, 2014, pp. 639–646.

[31] M. Görner, R. Haschke, H. Ritter, and J. Zhang, "MoveIt! task constructor for task-level motion planning," in *IEEE Int. Conf. Robot. Autom.*, 2019, pp. 190–196.

[32] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, 2010.

[33] K. Hauser, "Robust contact generation for robot simulation with unstructured meshes," in *Int. Symp. on Robotics Research*. Springer, 2016, pp. 357–373.

[34] M. Rickert and A. Gaschler, "Robotics Library: An object-oriented approach to robot applications," in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2017, pp. 733–740.

[35] 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.