



**US Army Corps
of Engineers®**
Engineer Research and
Development Center



Dambot™

Increasing the Degrees of Freedom on a Robot Arm

Benjamin Christie, Cameron Alred, Mike Paquette, and Garry
Glaspell

November 2023

The US Army Engineer Research and Development Center (ERDC) solves the nation's toughest engineering and environmental challenges. ERDC develops innovative solutions in civil and military engineering, geospatial sciences, water resources, and environmental sciences for the Army, the Department of Defense, civilian agencies, and our nation's public good. Find out more at www.erdclibrary.on.worldcat.org/discovery.

To search for other technical reports published by ERDC, visit the ERDC online library at www.erdclibrary.on.worldcat.org/discovery.

Increasing the Degrees of Freedom on a Robot Arm

Benjamin Christie, Cameron Alred, Mike Paquette, and Garry Glaspell

*US Army Engineer Research and Development Center (ERDC)
Geospatial Research Laboratory (GRL)
7701 Telegraph Road
Alexandria, VA 22315-3864*

Final Technical Report (TR)

DISTRIBUTION A: Approved for public release; distribution is unlimited.

Prepared for Headquarters, US Army Corps of Engineers
Washington, DC 20314-1000

Under PE 096/NA/X/3121/0000 / Project Number 500952 / Task 34

Abstract

This report provides an implementation of the moveit-commander Python module to generate trajectories for custom six- and seven-degrees of freedom (DoF) arms. The moveit_setup_assistant package was used to modify an existing five-DoF OpenManipulator-X model to increase its range of motion. Specifically, additional joints were fabricated and mounted to the physical arm. Also, the Unified Robot Description Format files were modified to account for the additional joints. In order to optimize the solvers, many changes to the MOVEit configuration files were made. The changes documented in this report lay the groundwork for leveraging MOVEit to expand the capabilities of low-DoF arms.

DISCLAIMER: The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. All product names and trademarks cited are the property of their respective owners. The findings of this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN NO LONGER NEEDED. DO NOT RETURN IT TO THE ORIGINATOR.

Contents

Abstract	ii
Figures and Tables.....	v
Preface.....	vi
1 Introduction.....	1
1.1 Background.....	1
1.2 Objectives.....	1
1.3 Approach and Scope	2
2 Increasing the OpenMANIPULATOR-X's Degrees of Freedom.....	3
2.1 Adding Additional Degrees of Freedom	4
2.2 Installation	5
2.3 Launch.....	8
3 Configuration	9
3.1 Inverse Kinematics Solver Configuration	9
3.2 Motion Planner Configuration	9
3.3 Mapping Software Configuration	11
3.3.1 Camera Setup	11
3.3.2 Inspecting with Real-Time Appearance-Based Mapping (RTAB-Map).....	11
4 Package for Custom MOVEit Trajectories.....	14
5 Known Issues.....	16
6 Future Work.....	17
7 Summary.....	18
Bibliography	19
Appendix A: Configuration Files and Nodes	21
A.1 open_manipulator.urdf.xacro.....	21
A.2 open_manipulator_robot.urdf.xacro.....	27
A.3 open_manipulator.transmission.xacro.....	28
A.4 open_manipulator.gazebo.xacro	29
A.5 open_manipulator_controller.cpp	32
A.6 joint_trajectory_controller.yaml	51
A.7 kinematics.yaml.....	52
A.8 chomp_planning.yaml	52
A.9 config/chomp_planning.yaml	52
A.10 config/open_manipulator.srdf	53

A.11	config/ros_controllers.yaml	55
A.12	config/sensors_3d.yaml.....	55
A.13	Orocos_vs_trac_testing.py	56
Appendix B: Launch Files.....		58
B.1	move_group.launch	58
B.2	chomp_planning_pipeline.launch.xml.....	60
B.3	open_manipulator_moveit_controller_manager.launch.xml	61
Appendix C: Custom Package Files		62
C.1	patterns.py	62
C.2	pose_array_nav.py.....	69
C.3	CMakesLists.txt.....	72
C.4	package.xml	77
Abbreviations.....		80
Report Documentation Page (SF 298)		81

Figures

Figures

1. Unmodified OpenMANIPULATOR-X arm with gripper (<i>left</i>) and modified OpenMANIPULATOR-X six-degrees of freedom (DoF) arm without gripper (<i>right</i>) in simulation.	3
2. Unmodified OpenMANIPULATOR-X arm with gripper (<i>left</i>) and modified OpenMANIPULATOR-X seven-DoF arm without gripper (<i>right</i>).	4
3. The 3D-printed 90-degree servo mount (<i>left</i>) and 3D-printed camera holder (<i>right</i>).	5
4. Modified arm links with notches for flexibility.	6
5. Transition planning with different planners: (a) Stochastic Trajectory Optimization for Motion Planning (STOMP); (b) Covariant Hamiltonian Optimization for Motion Planning (CHOMP); (c) Rapidly exploring random tree*(RRT).	10
6. Map generated with kinematic odometry on flexible arm shows significant distortion; it should show a flat surface but is wrapped.	12
7. Map generated with Real-Time Appearance-Based Mapping (RTAB-Map)'s visual / inertial measurement unit (IMU) odometry.	13

Preface

This study was conducted for the US Army Corps of Engineers under Program Element 29785D, Project Number 497149, Task 8, “Dambot™.” The technical monitor was Dr. Anton Netchaev.

The work was performed by the Data Representation Branch of the Topography Imagery and Geospatial Research Division, US Army Engineer Research and Development Center (ERDC), Geospatial Research Laboratory. At the time of publication, Mr. Vineet Gupta was branch chief; Mr. Michael Mailloux was acting division chief; and Dr. Austin Davis was the technical director. The deputy director was Ms. Valerie L. Carney, and the director was Mr. David R. Hibner.

The authors would like to acknowledge the following individuals for their contributions to this project: Mr. Chuck Ellison, Mr. Steven Bunkley, and Mr. Jordan Klein.

COL Christian Patterson was commander of ERDC, and Dr. David W. Pittman was director.

1 Introduction

1.1 Background

MOVEit (formerly known as MoveIt!) is a widely used Robot Operating System (ROS) framework to describe and plan composite tasks (Coleman et al. 2014). Typically, the high-level sequence of actions is fixed and known in advance, but the concrete realizations of these actions need to be adapted to a particular environment. This framework uses custom trajectory planners known as controllers to solve goals. These goals are fully defined within the configuration space (Gorner et al. 2019). Libraries such as the Open Motion Planning Library (OMPL) provide a collection of motion planners that can solve task-space goals. The motion planners assume that a description of the robotic manipulator is provided. This description must be an exact spatial and joint description of the robotic manipulator's mechanics in the form of Unified Robot Description Format (URDF) (ROS 2020) and Semantic Robot Description Format (SRDF) (ROS wiki 2019) files. Common arms such as Panda-Manipulator and OpenMANIPULATOR-X provide these files in their respective public repositories, but custom arms require significant modification or a total rewrite of these files (GmbH 2020; ROBOTIS 2022).

1.2 Objectives

This report provides a description of our package. Specifically, we provide a custom implementation of the moveit-commander Python module to generate trajectories for a custom six-degrees of freedom (DoF) arm. This package was also tested on our custom seven-DoF arm. The trajectories generated by this package are intended to be used to inspect objects of interest on variable scales. The package is also intended to use various custom patterns to accomplish this mapping motion, including rectangular sweeping motions, circular motions, and orbits around an arbitrary centroid. This research supports the “Revolutionize and Accelerate Decision Making” research and development priority, which states,

USACE leverages data science, artificial intelligence, machine learning, robotics, trade-space analytics, geospatial engineering, and advanced models and simulations to revolutionize geospatially informed

decision-making by understanding the operational environment, predicting likely outcomes, decreasing uncertainties by 80 percent, and shaping faster decisions that minimize risk, reduce costs, and accelerate mission delivery by 5× for military, civil works, and disaster response operations.

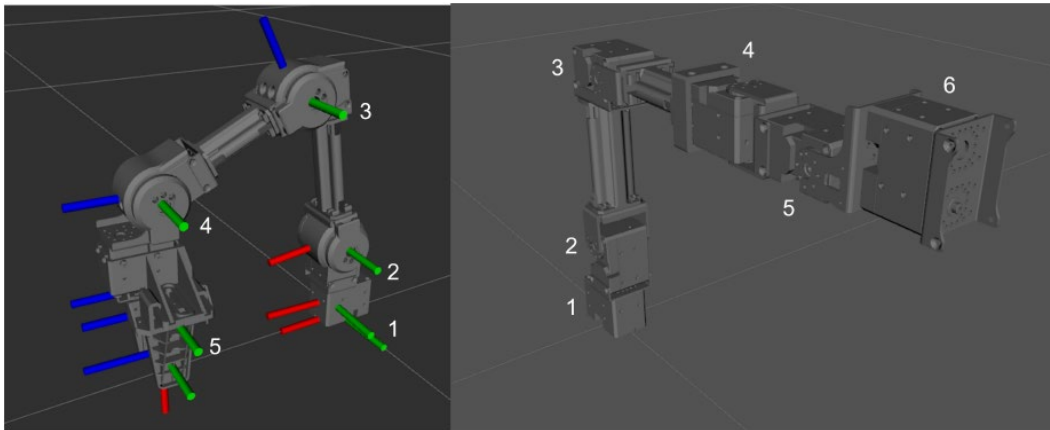
1.3 Approach and Scope

This package uses TRAC Lab's TRAC-IK software (Beeson and Ames 2015) for converting from task-space goals to the joint-space, and the Covariant Hamiltonian Optimization for Motion Planning (CHOMP) motion planner (Ratliff et al. 2009) within the joint space for trajectory planning within an ROS node. The package is also combined with Real-Time Appearance-Based Mapping (RTAB-Map) software (Labbé and Michaud 2019) to produce point cloud maps from a red-green-blue and depth (RGB-D) camera. In our project, the end-effector pose and `rgbd_sync` nodelet messages were used for odometry within the RTAB-Map node. It should also be noted that the cumulative point cloud produced in real time by RTAB-Map was also used as an obstacle avoidance point cloud sensor within MOVEit. Increasing the DoF of OpenMANIPULATOR-X was accomplished using manual edits of dozens of files and using `moveit_setup_assistant` for collision verification, motion planner improvement, and sensor configuration. All the required modifications are provided in Appendix A.

2 Increasing the OpenMANIPULATOR-X's Degrees of Freedom

The OpenMANIPULATOR-X arm by itself has four DoF. The fifth DoF is obtained by adding the gripper, as shown in Figure 1. The critical point of robotic manipulator mechanics occurs when a system reaches six DoF. Barring any collision restraints, a six-DoF arm should be able to satisfy any arbitrary pose goal for the end effector (that is, a specific position and orientation) within the unit Cartesian space. Both physical and software changes are necessary to modify the OpenMANIPULATOR-X to achieve six DoF. At a minimum, at least two more joints must be added to the arm portion of the manipulator to achieve six DoF, minus the gripper portion. In our project, we removed the gripper portion from the arm entirely. Our end effector became the tip of the Dynamixel motor that formerly controlled the gripper. Additionally, another joint was added using a 3D-printed link and a spare Dynamixel motor. This additional motor was added between joints four and five and became the source of pitch rotation for the vertical extremities of the arm.

Figure 1. Unmodified OpenMANIPULATOR-X arm with gripper (*left*) and modified OpenMANIPULATOR-X six-degrees of freedom (DoF) arm without gripper (*right*) in simulation.



SRDF/URDF files contain the robot's joints and links. Joints contain information about each DoF's location, articulation, and range of motion. Links contain information about the robot's segment(s), such as mass properties, visual and collision geometry, and inertial information. A URDF file consists of a sequence of joints and links, where each joint is defined to be the origin of a subsequent link. Joint i 's pose is defined as the

transformation from Joint $i-1$ to Joint i . An example URDF file can be found in Section A.1.

The default planners provided by OMPL and `open_manipulator_ros` were not sufficient for our six-DoF arm to plan properly. As such, we made the following software changes to accommodate our physical modifications to the robot: (1) we modified the Dynamixel ID sequences within the `open_manipulator.cpp` node, and (2) we modified the `open_manipulator` SRDF and URDF files (see Appendix A for more details).

2.1 Adding Additional Degrees of Freedom

For the seven-DoF modification, two 3D-printed fixtures and two additional servos were added, the gripper being fully removed and its servo being repurposed as the final revolute joint. Images of the original and modified arms are shown in Figure 2. An additional servo was inserted between the first two, and another between the elbow and wrist servos. Two 3D-printed parts were used for arm modification, shown in Figure 3. The first printed piece mounted the fifth servo at a 90-degree angle to the fourth joint, and the other printed piece mounted the camera to the final servo. All other modifications were done using parts and motors from a second Open-MANIPULATOR-X.

Figure 2. Unmodified OpenMANIPULATOR-X arm with gripper (*left*) and modified OpenMANIPULATOR-X seven-DoF arm without gripper (*right*).

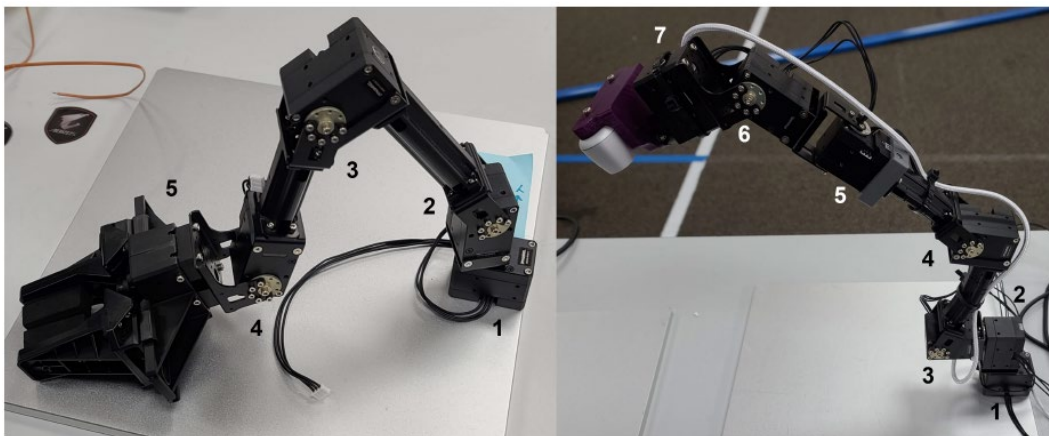


Figure 3. The 3D-printed 90-degree servo mount (*left*) and 3D-printed camera holder (*right*).



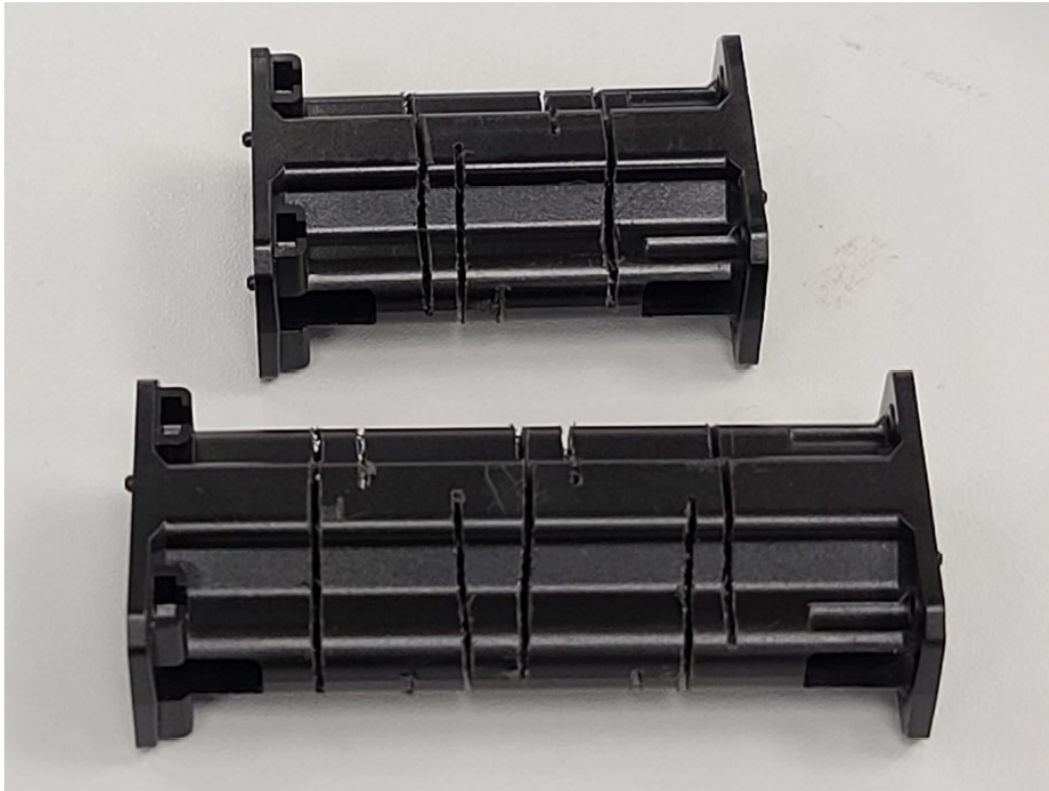
The modifications made to the OpenMANIPULATOR-X significantly altered its kinematic behavior. Whereas the original arm had only four DoF, this modified arm has seven. Because the total number of DoF is at least six, and the joints are properly oriented, all arbitrary end effector poses (both position and orientation) can be achieved. This allows greater freedom to position and orient the depth camera and navigate around obstacles. However, allowable poses are still significantly restricted by the limitations of the joints.

Additionally, as seen in Figure 4, notches were cut in both plastic members that comprise links four and five to make the arm less rigid and introduce sagging and oscillating behavior that would be expected on a larger arm.

2.2 Installation

A multitude of ROS and Python packages are necessary for this package to function. Note that the following instructions assume that the user has a physical implementation of the custom six-DoF arm used in this project. Additionally, note that the adjusted SRDF and URDF files have not been tested in simulation. ROS Melodic was used in this project within Ubuntu 18.04.

Figure 4. Modified arm links with notches for flexibility.



First, install the following ROS packages using apt:

```
$ sudo apt install ros-melodic-desktop-full ros-melodic-
  -RTAB-Map-ros ros-melodic-trac-ik* ros-melodic-
  realsense2-camera ros-melodic-ros-controllers ros-
  melodic-gazebo* ros-melodic-industrial-core swig
  python3-wstool python3-catkin-tools clang-format-10
  python3-rosdep
Installing Nlopt may be necessary for trac-ik to function:
$ cd ~/Downloads/ && git clone https://github.com/
  stevengj/nlopt.git

$ cd nlopt
$ mkdir build && cd build
$ cmake ..
$ make
$ sudo make install
```

Note: If Nlopt does not install properly, change line 171 of nlopt/CMakeLists.txt from

```
set (CMAKE_CXX_FLAGS "-std=c++11 ${CMAKE_CXX_FLAGS}")
```

to

```
set (CMAKE_CXX_FLAGS "-std=c++11 ${CMAKE_CXX_FLAGS} -
    fPIC -fpermssive")
```

and attempt to reinstall. Then, create a new catkin workspace and install `open_manipulator` packages from source:

```
$ mkdir ~/your_ws/src -p && cd ~/your_ws/src
$ git clone https://github.com/ROBOTIS-GIT/DynamixelSDK
.git
$ git clone https://github.com/ROBOTIS-GIT/dynamixel- work-
    bench.git
$ git clone https://github.com/ROBOTIS-GIT/dynamixel- work-
    bench-msgs.git
$ git clone https://github.com/ROBOTIS-GIT/ open_manipulator.git
$ git clone https://github.com/ROBOTIS-GIT/ open_manipula-
    tor_msgs.git
$ git clone https://github.com/ROBOTIS- GIT/ open_manipula-
    tor_simulations.git
$ git clone https://github.com/ROBOTIS-GIT/
    robotis_manipulator.git
```

For CHOMP to function, MOVEit must be built from source:

```
$ cd ~/your_ws
$ source /opt/ros/melodic/setup.bash
$ wstool init src
$ wstool merge -t src https://raw.githubusercontent.com
    /ros-planning/moveit/master/moveit.rosinstall
$ wstool update -t src
$ rosdep install -y --from-paths src --ignore-src
--rosdistro ${ROS_DISTRO}
$ catkin config --extend /opt/ros/melodic --cmake-args
    -DCMAKE_BUILD_TYPE=Release
$ catkin build
```

Finally, paste the files in Appendices A, B, and C into their respective locations and run the following:

```
$ cd ~/your_ws
$ catkin_make
```

If that fails, try running the following:

```
$ cd ~/your_ws
$ catkin build
```

2.3 Launch

To launch the arm controller within MOVEit (if an OpenCR2.0 interface is used), use the following command:

```
$ roslaunch open_manipulator_controllers  
  joint_trajectory_controller.launch sim:=false  
  usb_port:=/dev/ttyACM0
```

If a U2D2 interface is used instead, run the following command:

```
$ roslaunch open_manipulator_controllers joint_trajectory_con-  
  troller.launch sim:=false  
usb_port:=/dev/ttyUSB0
```

Then, launch the `open_nav_pkg` package:

```
$ rosrun open_nav_pkg main.py
```

You can also run the `patterns` file, which does not subscribe to any ROS topics:

```
$ roscd open_nav_pkg/src  
$ python patterns.py
```

You can then publish to the `PoseArray` topic that `open_nav_pkg` listens to:

```
$ rostopic pub /pose_array geometry_msgs/PoseArray <  
  your_pose_msg>
```

Finally, launch the RTAB-Map node:

```
$ roslaunch RTAB-Map_ros data_recorder.launch
```

The complete launch files are provided in Appendix B, and the customized `open_nav_pkg` files are located in Appendix C.

3 Configuration

3.1 Inverse Kinematics Solver Configuration

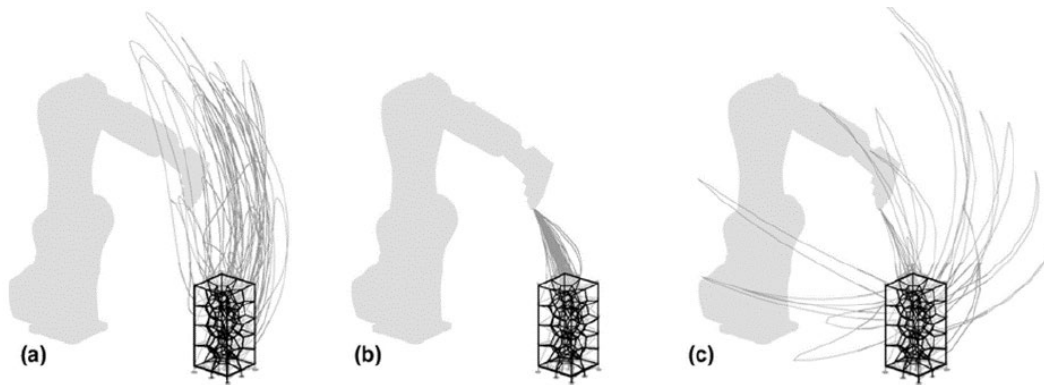
The default inverse kinematics (IK) solver (Orocos Kinematics and Dynamics Library [KDL]) built into MOVEit is not optimized for low-DoF (less than seven DoF) robotic arms (Soetens et al. 2020). One of the issues that were observed over the course of this project included “no IK solutions being found for arbitrary pose goals,” even when one existed. This issue occurred most commonly in robotic manipulators when the DoF was below a certain threshold. However, our custom arms should theoretically be able to achieve these poses. As an example, a script was created that attempted to solve the IK problem for an array of 100 poses scattered throughout the task space. The script limited the IK solver to 5 seconds per pose, which should be sufficient for a solution. In this script, the IK solver was only able to solve poses relatively close to the “zero” and “home” poses, set to be the joint states [0.0 0.0 0.0 0.0 0.0 0.0] and [0.0 - 1.0 0.3 0.0 0.7 0.0] radians, respectively. As a result, we replaced the default inverse kinematics solver with TRAC Lab’s custom IK solver, `trac_ik`. This solver is faster, more precise, and more consistent than the Orocos KDL solver (Beeson and Ames 2015). The `kinematics.yaml` file (Appendix A) was changed to implement this change, but this project used the `moveit_setup_assistant` package. This IK solver did not fail on the six-DoF arm for poses within reach. Additionally, the `trac_ik_python` module was installed for our custom pattern planner node.

3.2 Motion Planner Configuration

While IK solvers find the goal position for any pose in the Cartesian space, the motion planner determines the path from the current state to the goal state. This project tested several different trajectory planners, including RRTConnect (Rapidly exploring random tree Connect), Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE), Lazy Probabilistic Roadmaps (LazyPRM), LazyPRMStar (LazyPRM with the star strategy), Sparse Roadmap Spanner (SPARS), CHOMP, and Stochastic Trajectory Optimization for Motion Planning (STOMP). Note that all planners except CHOMP and STOMP are included within MOVEit by default. LazyPRMStar, LazyPRM, and SPARS take random samples from the robot configuration space and test if they are within free space. Then, the planners attempt to use a local planner to connect these

configuration states to other configurations. The starting and goal configurations are then added and a graph search algorithm is applied using interpolation to determine a path between the starting and goal configurations (Planners Benchmarking 2021). RRTConnect is different; it incrementally uses two rapidly exploring random trees, rooted at the start and goal configurations, using interpolation to connect configurations. STOMP (Williams 2021) uses noisy trajectory generation to explore space around an initial, potentially infeasible trajectory, which is then combined to produce an updated trajectory with lower cost. A cost function based on a combination of obstacle and smoothness cost is optimized in each iteration. However, STOMP appears to be optimized for seven DoF robotic arms and higher. Finally, CHOMP (Ratliff et al. 2009), given a (potentially infeasible) naive trajectory, reacts to the surrounding environment to quickly pull the trajectory out of collision while simultaneously optimizing dynamical quantities such as joint velocities and accelerations. As shown in Figure 5, the CHOMP planner rapidly converges to a smooth, collision-free trajectory that can be executed efficiently on the robot. Additionally, the CHOMP planner was tested on a physical six- and seven-DoF arm during development.

Figure 5. Transition planning with different planners: (a) Stochastic Trajectory Optimization for Motion Planning (STOMP); (b) Covariant Hamiltonian Optimization for Motion Planning (CHOMP); (c) Rapidly exploring random tree*(RRT).



From our testing, the CHOMP motion planner worked the best for our custom six-DoF arm. Although RRTConnect- and PRM-based planners did function on the custom arm, movements were choppy, and trajectory solutions for repeated poses were inconsistent. These distinctions are most likely due to how the motion planners handle the short length of the links, namely links three, four, and five (Figure 1). It should be noted that when the seventh DoF was added, CHOMP did not perform to the degree previously achieved with the six-DoF arm. For the seven-DoF arm,

RRTConnect improved performance, and the choppy movement was eliminated. Therefore, it is believed that for six-DoF arms, CHOMP is superior and should be used. However, when more DoFs are added and some joints become redundant, the OMPL planners can and should be used to improve motion and ease of use.

3.3 Mapping Software Configuration

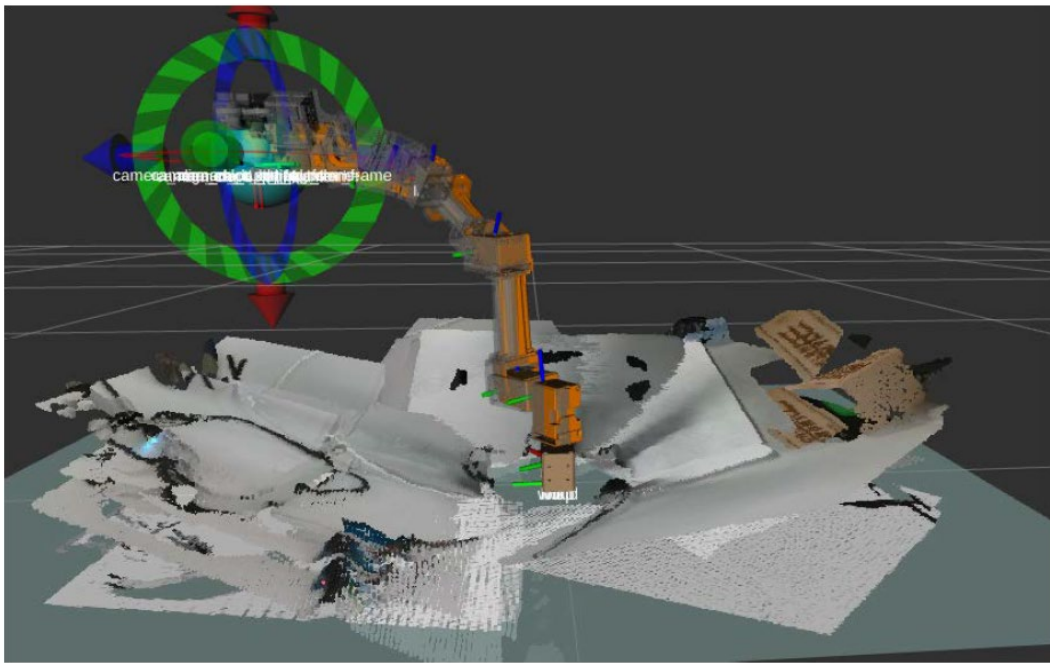
3.3.1 Camera Setup

We attached an Intel RealSense D435i depth camera to the end of the arm. The RealSense camera features a single infrared (IR) laser projector that projects a pattern of dots onto the scene and two IR cameras for stereo depth sensing. It also features a single RGB sensor to color the depth points. By default the camera senses objects up to a minimum distance of 19.5 cm. Given the size of the arm, this minimum distance is fairly restrictive. We alleviated this problem by using custom launch parameters to reduce the camera resolution to 480×270 . This modification reduced the minimum scan distance to approximately 9.5 cm, thus improving our ability to inspect close-up objects. At close distances, this reduction in resolution did not have a significant negative impact on the depth data.

3.3.2 Inspecting with Real-Time Appearance-Based Mapping (RTAB-Map)

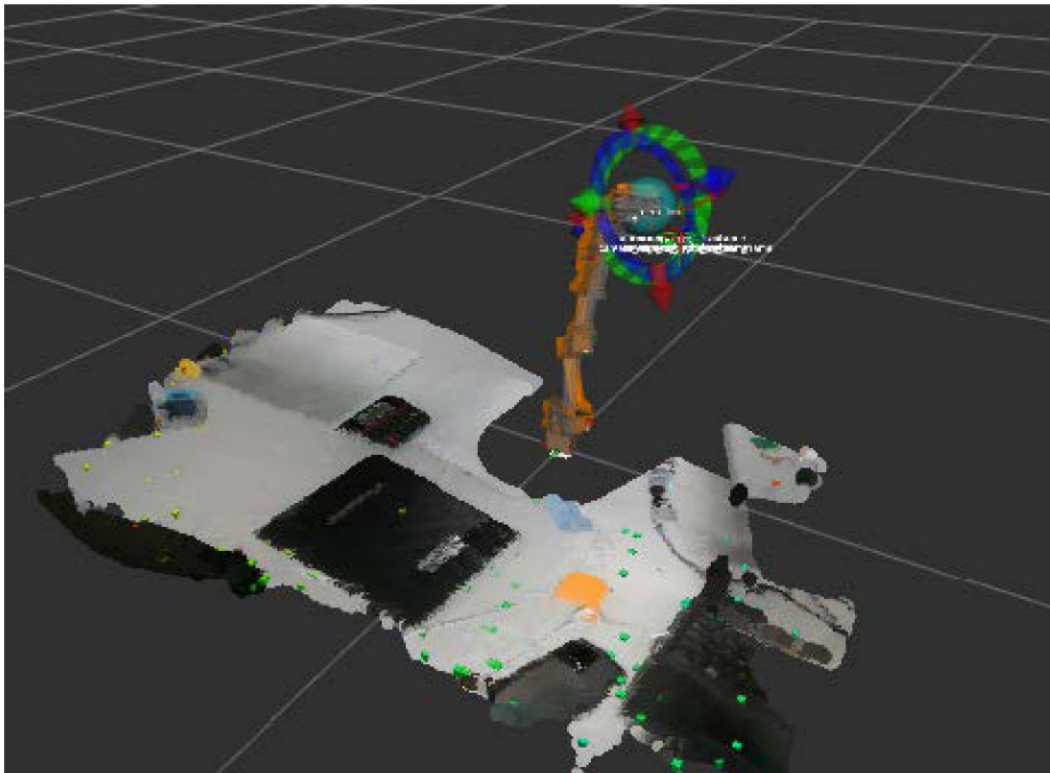
To generate a 3D point cloud of an inspected object, we used RTAB-Map (Labbé and Michaud 2019). RTAB-Map uses loop closure to identify keyframes and can retroactively correct its pose. RTAB-Map also leverages odometry data for pose correction. For testing purposes, RTAB-Map was fed the transformation frame of the camera according to the robot's TF tree (forward kinematics). This worked well for the rigid arm. However, once we incorporated the links (as shown in Figure 4) to mimic the motion of a larger arm, distortion in the scan was evident. This distortion can be observed in Figure 6. The flexible joints made the arm sag and oscillate, causing large errors between the assumed and true camera positions.

Figure 6. Map generated with kinematic odometry on flexible arm shows significant distortion; it should show a flat surface but is wrapped.



To account for sag and oscillation of the flexible joints, we leveraged RTAB-Map's own RGB-D-based visual odometry in conjunction with the RealSense camera's inertial measurement unit (IMU). We also used the Madgwick IMU filter (Madgwick et al. 2011) to generate an orientation estimate before feeding the IMU topic to RTAB-Map under the appropriate topic. A map generated with this approach is shown in Figure 7.

Figure 7. Map generated with visual / inertial measurement unit (IMU) odometry from Real-Time Appearance-Based Mapping (RTAB-Map).



To initialize scanning, the arm is placed in a pose where it does not deflect. This was an upright pose for the test arm but could be any nondeflecting pose for the full-size arm. Once in this pose, a service was called to reset the pose of the odometry camera frame to the camera frame on the robot. This ensures that the map is oriented correctly with respect to the arm. At this point, poses were changed using MOVEit, and a point cloud of the environment was collected.

4 Package for Custom MOVEit Trajectories

MOVEit provides several interfaces for the move group node. This project utilized the Python-based move group interface since it was easier to integrate `trac_ik` into Python than into C++. The documentation of the MOVEit commander namespace is immense and this implementation of the MOVEit commander namespace is not novel. This project used the `MoveGroupCommander` and `RobotCommander` classes in conjunction with `trac_ik_python` to calculate IK solutions. These IK solutions, which were produced by `trac_ik_python`, were then passed to the `move_group` node using `MoveGroupCommander`, where CHOMP was used for motion planning. We hypothesize that this node is usable by any robotic arm with DoF greater than five. An implementation of the rectangle sweep method can be seen in the pseudocode below.

```
for c in range(n):
    go_to_pose_goal(ix+dx, iy+dy, iz-float(c)/float(n)*dz
    , iox, ioy, ioz, iow)
    go_to_pose_goal(ix+dx, iy-dy, iz-float(c)/float(n)*dz
    , iox, ioy, ioz, iow) #
center after for loop
go_to_pose_goal(ix, iy, iz, iox, ioy, ioz, iow)
```

It should be noted that when configured with the move group node, the CHOMP motion planner only accepts joint space (configuration space) goals, not pose goals like the planners of the OMPL. Therefore, implementing an IK solver within the motion planning script was necessary. An implementation of the go to pose goal method can be seen below. This implementation is unique and produces extremely low failure rates (less than 0.5%), especially considering that the arm is low DoF.

```
ik_solver = trac_ik_python.trac_ik.IK("link1", "link7") # ini-
    tialize TRAC_IK solver
seed_state = [0.0] * ik_solver.number_of_joints # should
    be a matrix of N joints
joint_goal = ik_solver.get_ik(seed_state, x, y, z, ox, oy,
    oz, ow)
## Now, we call the planner to compute the plan and execute
    it.
move_group.go(joint_goal, wait=True)
# Calling 'stop()' ensures that there is no residual
    movement
move_group.stop()
```

Finally, by adding a seventh degree of freedom, changing the quaternion of the end effector irrespective of the XYZ position of the end effector was feasible using this `go_to_pose_goal` method. An implementation of an orbital sweep method can be seen in the pseudocode below. Note that this orbital sweep should be achievable within six DoF, but the arm used in this project had extremely short links. This led to an increase of pose goal failures for rotated quaternions with respect to the original orientation of the end effector. This failure rate was lowered by increasing the goal tolerance within `trac_ik` and within the move group node, but this approach was not applied in the final version of this project.

```
for c in range(n):
    go_to_pose_goal(ix+dx, iy+dy, iz-float(c)/float(n)*dz
        , py[0], py[1], py[2], py[3])

    go_to_pose_goal(ix+dx, iy+dy, iz-float(c)/float(n)*dz
        , iox, ioy, ioz, iow)

    go_to_pose_goal(ix+dx, iy-dy, iz-float(c)/float(n)*dz
        , iox, ioy, ioz, iow)

    go_to_pose_goal(ix+dx, iy-dy, iz-float(c)/float(n)*dz
        , ny[0], ny[1], ny[2], ny[3])
    go_to_pose_goal(ix, iy, iz, iox, ioy, ioz, iow)
```

The final contribution of this project was a fake waypoint planner that uses `rospy` and a `go_to_pose_goal` loop that subscribes to an ROS topic that publishes an array of poses. If changes were detected within the subscribed array, the node produced a trajectory that attempted to navigate to the appended pose. The array of poses was designed to be published by an exploration node for autonomous inspection.

5 Known Issues

The custom six-DoF OpenMANIPULATOR-X arm used in this project has many known issues, including the following:

After launching and running MOVEit, the torque controllers stay active on the arm. These motors do not turn off after being disconnected from the computer. Furthermore, this issue persists through relaunching of MOVEit. If a separate package is used, it will not be able to access the Dynamixel units and will not be able to adjust their position or joint angles. The OpenCR2.0 must be physically disconnected from the power source for this issue to be resolved.

There is notable gear slop and goal tolerance that is identified by MOVEit but is not corrected by the joint trajectory controller regardless of what motion planner was used. Even after adjusting tolerances within the `open_nav_pkg` node, this issue persisted. This issue was noticeable when displayed in rviz and made a substantial difference for trajectory planning in cramped environments. This is believed to be due to the fact that the Dynamixels were not programmatically adjusted for the additional torque generated by the manipulator's extremities.

6 Future Work

In the future, software could be developed that marks a voxel grid by the known, unknown, and occupied space. The centroid of the borders of known and unknown voxel space could be pushed to an array of poses, which can then be read by the `open_nav_pkg` node for pseudo-waypoint navigation. This would be a huge development with respect to autonomous inspection within MOVEit.

An additional future adaptation of this project could be the incorporation of a gripper controller for autonomous manipulation. Currently, this package does not support this. Autonomous manipulation combined with autonomous inspection would be a huge step in nondestructive testing, especially when integrated directly into MOVEit.

Finally, more work needs to be done to integrate additional sensors, such as lidar or chemical, biological, radiological, nuclear, and high-yield explosives sensors. This project used the point cloud message produced by the depth camera to avoid obstacles, but these additional sensors allow the soldier to fulfill reconnaissance roles in dangerous environments.

7 Summary

In summary, this project produced a node that can be used as a pattern or pose goal wrapper for a running MOVEit instance. This project provides a novel implementation of `trac_ik` and CHOMP on custom robotic manipulators using modified OpenMANIPULATOR-X hardware. This project also showed that using `moveit_setup_assistant` for custom arms is unnecessary and can result in errors with non-OMPL trajectory planners. It should be noted that the modified arms had substantial joint limits; some joints had angle limits of 180° or less. We believe this affected the OMPL planners' ability to plan trajectories efficiently. It is still unclear why CHOMP performed so well comparatively, aside from the fact that CHOMP is optimized for six-DoF arms. This distinction will be explored further in future projects.

Bibliography

- Beeson, P., and B. Ames. 2015. "TRAC-IK: An open-source library for improved solving of generic inverse kinematics." 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids) (Seoul, South Korea), no. Nov, 928–935. <https://doi.org/10.1109/HUMANOIDS.2015.7363472>.
- Coleman, D., I. A. Sucan, S. Chitta, and N. Correll. 2014. "Reducing the Barrier to Entry of Complex Robotic Software: A MOVEit Case Study." *Journal of Software Engineering for Robotics* 1 (5): 3–16. <https://doi.org/10.6092/JOSER.2014.05.01.p3>.
- Franka Emika GmbH. 2020. *PANDA's Instruction Handbook*. Accessed 27 July 2021. https://dentec.pl/pliki/Artykul/855_dentec-frankainstrukcja-uzytownika.pdf.
- Gorner, M., R. Haschke, H. Ritter, and J. Zhang. 2019. *MOVEit Task Constructor for Task-Level Motion Planning* (May): 190–96. <https://doi.org/10.1109/ICRA.2019.8793898>.
- Labbé, M., and F. Michaud. 2019. "RTAB-Map as an Open-Source Lidar and Visual Simultaneous Localization and Mapping Library for Large-Scale and Long-Term Online Operation." *Journal of Field Robotics* 36 (March): 416–46. <https://doi.org/10.1002/rob.21831>.
- Madgwick, S. O., A. J. Harrison, and R. Vaidyanathan. 2011. "Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm." *IEEE International Conference on Rehabilitation Robotics*, 1–7.
- Planners Benchmarking. 2021. "Motion Planners Available in MoveIt!" Accessed 28 July 2021. https://planners-benchmarking.readthedocs.io/en/latest/user_guide/2_motion_planners.html.
- Ratliff, N., M. Zucker, J. A. Bagnell, and S. Srinivasa. 2009. "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning." 2009 *IEEE International Conference on Robotics and Automation* (Kobe) (May): 489–94. <https://doi.org/10.1109/ROBOT.2009.5152817>.
- ROBOTIS. 2022. "OpenMANIPULATOR-X." Accessed 27 July 2021. <https://emanual.robotis.com/docs/en/platform/openmanipulatorx/overview/>.
- ROS. 2020. "urdf." <https://github.com/ros/urdf>.
- ROS wiki. 2019. "srdf." <https://wiki.ros.org/srdf>.
- Soetens, P., T. Issaris, H. Bruyninckx, S. Joyeux, and R. Smits. 2020. *Orocos Documentation*. <https://docs.oroocos.org/>.

Williams, J. 2021. "STOMP: Stochastic Trajectory Optimization for Motion Planning—Autonomous Motion." Accessed 28 July 2021, Max Planck Institute for Intelligent Systems, <https://is.mpg.de>.

Appendix A: Configuration Files and Nodes

For our custom arm to function within MoveIt, several open manipulator files need to be modified. These can be changed using `moveit_setup_assistant`, but this causes several issues. For example, when implementing a custom inverse kinematics (IK) solver or motion planner, some include and param tags within the `*launch.xacro` files are not modified properly, leading to launch errors that are difficult to trace. Therefore, all files that are modified by hand and are difficult to modify using `moveit_setup_assistant` are included here.

Additionally, a testing script that this project used to compare Orocos Kinematics and Dynamics Library (KDL) and TRAC_IK performance is included in this appendix.

A.1 `open_manipulator.urdf.xacro`

```
<?xml version="1.0"?>
<!-- Open_Manipulator Chain -->
<robot xmlns:xacro="http://ros.org/wiki/xacro">

  <!-- Link 1 -->
  <link name="link1">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package:// open_manipulator_description/meshes/chain_link1.stl"
          scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="grey"/>
    </visual>

    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package:// open_manipulator_description/meshes/chain_link1.stl"
          scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="3.0876154e-04 0.0000000e+00
        -1.2176461e-04" />
      <mass value="7.9119962e-02" />
      <inertia ixx="1.2505234e-05" ixy="0.0" ixz="
        -1.7855208e-07"
```

```

        iyy="2.1898364e-05" iyz="0.0"
        izz="1.9267361e-05" />
    </inertial>
</link>

<!-- Joint 1 -->
<joint name="joint1" type="revolute">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="0.012 0.0 0.017" rpy="0 0 0"/>
    <axis xyz="0 0 1"/>
    <limit velocity="4.8" effort="1" lower="-${pi*0.9}" up-
per=" ${pi*0.9}" />
</joint>

<!-- Link 2 -->
<link name="link2">
    <visual>
        <origin xyz="0 0 0.019" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package:// open_manipulator_descrip-
tion/meshes/chain_link2.stl"
scale="0.001 0.001 0.001"/>
        </geometry>
        <material name="grey"/>
    </visual>

    <collision>
        <origin xyz="0 0 0.019" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package:// open_manipulator_descrip-
tion/meshes/chain_link2.stl"
scale="0.001 0.001 0.001"/>
        </geometry>
    </collision>

    <inertial>
        <origin xyz="-3.0184870e-04 5.4043684e-04 ${0.018
+ 2.9433464e-02}" />
    <mass value="9.8406837e-02" />
        <inertia ixx="3.4543422e-05" ixy="-1.6031095e-08"
ixz="-3.8375155e-07"
            iyy="3.2689329e-05" iyz="2.8511935e-08"
            izz="1.8850320e-05" />
    </inertial>
</link>

<!-- Joint 2 -->
<joint name="joint2" type="revolute">
    <parent link="link2"/>
    <child link="link3"/>
    <origin xyz="0.0 0.0 0.0595" rpy="0 0 0"/>
    <axis xyz="0 1 0"/>

```

```

    <limit velocity="4.8" effort="1" lower="{pi*0.57}" upper="{pi*0.5}" />
  </joint>

  <!-- Link 3 -->
  <link name="link3">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package:// open_manipulator_description/meshes/chain_link3.stl"
          scale="0.001 0.001 0.001"/>
      </geometry>
      <material name="grey"/>
    </visual>

    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package:// open_manipulator_description/meshes/chain_link3.stl"
          scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="1.0308393e-02 3.7743363e-04 1.0170197e-01" />
      <mass value="1.3850917e-01" />
      <inertia ixx="3.3055381e-04" ixy="-9.7940978e-08"
        ixz="-3.8505711e-05"
        iyy="3.4290447e-04" iyz="-1.5717516e-06"
        izz="6.0346498e-05" />
    </inertial>
  </link>

  <!-- Joint 3 -->
  <joint name="joint3" type="revolute">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="0.024 0 0.128" rpy="0 0 0"/>
    <axis xyz="0 1 0"/>
    <limit velocity="4.8" effort="1" lower="{pi*0.3}" upper="{pi*0.44}" />
  </joint>

  <!-- Link 4 -->
  <link name="link4">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package:// open_manipulator_description/meshes/chain_link4.stl"
          scale="0.001 0.001 0.001"/>
      </geometry>

```

```

        <material name="grey"/>
    </visual>

    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package:// open_manipulator_descrip-
tion/meshes/chain_link4.stl"
            scale="0.001 0.001 0.001"/>
        </geometry>
    </collision>

    <inertial>
        <origin xyz="9.0909590e-02 3.8929816e-04
2.2413279e-04" />
        <mass value="1.3274562e-01" />
        <inertia ixx="3.0654178e-05" ixy="-1.2764155e-06"
ixz="-2.6874417e-07"
            iyy="2.4230292e-04" iyz="1.1559550e-08"
            izz="2.5155057e-04" />
    </inertial>
</link>

<!-- Joint 4 -->
<joint name="joint4" type="revolute">
    <parent link="link4"/>
    <child link="link5"/>
    <origin xyz="0.12375 0.0 0.0" rpy="0 0 0"/>
    <axis xyz="0 0 -1"/>
    <limit velocity="4.8" effort="1" lower="-${pi*0.63}" up-
per=" ${pi*0.63}" />
</joint>

<!-- Link 5 -->
<link name="link5">
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package:// open_manipulator_descrip-
tion/meshes/chain_link5.stl"
            scale="0.001 0.001 0.001"/>
        </geometry>
        <material name="grey"/>
    </visual>

    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package:// open_manipulator_descrip-
tion/meshes/chain_link5.stl"
            scale="0.001 0.001 0.001"/>
        </geometry>
    </collision>

    <inertial>

```

```

        <origin xyz="4.4206755e-02 3.6839985e-07
8.9142216e-03" />
        <mass value="1.4327573e-01" />
        <inertia ixx="8.0870749e-05" ixy="0.0" ixz="
-1.0157896e-06"
            iyy="7.5980465e-05" iyz="0.0"
            izz="9.3127351e-05" />
    </inertial>
</link>

<!-- Joint 5 -->
<joint name="joint5" type="revolute">
    <parent link="link5"/>
    <child link="link6"/>
    <origin xyz="0.069 0.0 0.0" rpy="0 0 0"/>
    <axis xyz="0 1 0"/>
    <limit velocity="4.8" effort="1" lower="{pi*0.57}" upper="{pi*0.65}" />
</joint>

<!-- Link 6 -->
<link name="link6">
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package:// open_manipulator_description/meshes/chain_link6.stl"
scale="0.001 0.001 0.001"/>
        </geometry>
        <material name="grey"/>
    </visual>

    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <mesh filename="package:// open_manipulator_description/meshes/chain_link6.stl"
scale="0.001 0.001 0.001"/>
        </geometry>
    </collision>

    <inertial>
        <origin xyz="4.4206755e-02 3.6839985e-07
8.9142216e-03" />
        <mass value="1.4327573e-01" />
        <inertia ixx="8.0870749e-05" ixy="0.0" ixz="
-1.0157896e-06"
            iyy="7.5980465e-05" iyz="0.0"
            izz="9.3127351e-05" />
    </inertial>
</link>

<!-- Joint 6 -->
<joint name="joint6" type="revolute">
    <parent link="link6"/>

```

```

<child link="link7"/>
  <origin xyz="0.064 0.0 0.0" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
  <limit velocity="4.8" effort="1" lower="-${1*pi}" upper="${pi}"
/>
</joint>

<!-- Link 7 -->
<link name="link7">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package:// open_manipulator_descrip-
tion/meshes/chain_link7.stl"
scale="0.001 0.001 0.001"/>
    </geometry>
    <material name="grey"/>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package:// open_manipulator_descrip-
tion/meshes/chain_link7.stl"
scale="0.001 0.001 0.001"/>
    </geometry>
  </collision>

  <inertial>
    <origin xyz="4.4206755e-02 3.6839985e-07
8.9142216e-03" />
    <mass value="1.4327573e-01" />
    <inertia ixx="8.0870749e-05" ixy="0.0" ixz="
-1.0157896e-06"
            iyy="7.5980465e-05" iyz="0.0"
            izz="9.3127351e-05" />
  </inertial>
</link>

<!-- Realsense Camera Joint Position -->
<joint name="camera_joint" type="fixed">
  <axis xyz="1 0 0" />
  <origin xyz="0.0247 0.000 0.000" rpy="0 0 0"/>
  <parent link="link7"/>
  <child link="camera_link"/>
</joint>
<link name="camera_link"/>
<link name="ground_plane_box">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="5.0 5.0 0.1"/>
    </geometry>
    <material name="Grey">

```

```

        <color rgba="0.7 0.7 0.7 1.0"/>
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="5.0 5.0 0.1"/>
      </geometry>
    </collision>
  </link>

  <joint name="ground_plane_box" type="fixed">
    <parent link="world" />
    <child link="ground_plane_box" />
    <origin xyz="0 0 -0.05" rpy="0 0 0" />
  </joint>
</robot>

```

A.2 open_manipulator_robot.urdf.xacro

```

<?xml version="1.0"?>
<!-- Open_Manipulator Chain -->

<robot name="open_manipulator" xmlns:xacro="http://ros.
  org/wiki/xacro">
  <!-- Import Rviz colors -->

  <xacro:include filename="$(find open_manipulator_descrip-
    tion)/urdf/materials.xacro"
  />
  <!-- Import all Gazebo-customization elements,
    including Gazebo colors -->

  <xacro:include filename="$(find open_manipulator_descrip-
    tion)/urdf/open_manipulator.gazebo.xacro" />
  <!-- Import Transmission -->

  <xacro:include filename="$(find open_manipulator_descrip-
    tion)/urdf/open_manipulator.transmission.xacro" />
  <!-- Import URDF -->

  <xacro:include filename="$(find open_manipulator_descrip-
    tion)/urdf/open_manipulator.urdf.xacro" />
  <!-- World -->

  <link name="world"/>
  <!-- World fixed joint-->

  <joint name="world_fixed" type="fixed">

  <origin xyz="0 0 0" rpy="0 0 0"/>
  <parent link="world"/>

  <child link="link1"/>

```

```

</joint>
</robot>

```

A.3 open_manipulator.transmission.xacro

```

<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
<!-- Transmission macro -->

<xacro:macro name="SimpleTransmissionPosition" params=" joint
  n">

<transmission name="tran${n}">

<type>transmission_interface/SimpleTransmission</type>

<joint name="${joint}">

<hardwareInterface>hardware_interface/ PositionJointInter-
  face</hardwareInterface>
</joint>

<actuator name="motor${n}">
<mechanicalReduction>1</mechanicalReduction>
</actuator>

</transmission>

</xacro:macro>
<!-- Transmission macro -->

<xacro:macro name="SimpleTransmissionEffort" params="
joint n">
<transmission name="tran${n}">
<type>transmission_interface/SimpleTransmission</type>
<joint name="${joint}">

<hardwareInterface>hardware_interface/ EffortJointInter-
  face</hardwareInterface>
</joint>

<actuator name="motor${n}">
<mechanicalReduction>1</mechanicalReduction>
</actuator>

</transmission>

</xacro:macro>
<!-- Transmission 1 -->

<xacro:SimpleTransmissionPosition n="1" joint="joint1"
  />
<!-- Transmission 2 -->

```

```

<xacro:SimpleTransmissionPosition n="2" joint="joint2"
/>
<!-- Transmission 3 -->

<xacro:SimpleTransmissionPosition n="3" joint="joint3"
/>
<!-- Transmission 4 -->
<xacro:SimpleTransmissionPosition n="4" joint="joint4"
/>
<!-- Transmission 5 -->

<xacro:SimpleTransmissionPosition n="5" joint="joint5"
/>
<!-- Transmission 6 -->

<xacro:SimpleTransmissionPosition n="6" joint="joint6"
/>
</robot>

```

A.4 open_manipulator.gazebo.xacro

```

<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
<!-- Gazebo Reference macro -->

<xacro:macro name="RefLink" params="ref">

<gazebo reference="${ref}">

<kp>1000000.0</kp>
<kd>100.0</kd>

<mu1>30.0</mu1>

<mu2>30.0</mu2>
<maxVel>1.0</maxVel>
<minDepth>0.001</minDepth>
<material>Gazebo/DarkGrey</material>
</gazebo>

</xacro:macro>
<!-- World -->

<gazebo reference="world"/>
<!-- Link1 -->

<RefLink ref="link1"/>
<!-- Link2 -->

<RefLink ref="link2"/>
<!-- Link3 -->

<RefLink ref="link3"/>

```

```
<!-- Link4 -->

<RefLink ref="link4"/>
<!-- Link5 -->
<RefLink ref="link5"/>
<!-- Link6 -->
<RefLink ref="link6"/>
<!-- end effector link -->

<gazebo reference="end_effector_link">

<material>Gazebo/Red</material>

</gazebo>
<!-- ros_control plugin -->
<gazebo>
<plugin name="gazebo_ros_control" filename=" libgazebo_ros_con-
  trol.so">

<controlPeriod>0.001</controlPeriod>

<robotSimType>gazebo_ros_control/DefaultRobotHWSim</ ro-
  botSimType>

<legacyModeNS>true</legacyModeNS>
</plugin>

</gazebo>
<gazebo>
<!-- loading plugin: mimic joints works in gazebo now
  -->

<plugin name="gazebo_gripper_ros_control" filename="
  libroboticsgroup_gazebo_mimic_joint_plugin.so">
<joint>gripper</joint>
<mimicJoint>gripper_sub</mimicJoint>
<multiplier>1.0</multiplier>
<offset>0.0</offset>
</plugin>

</gazebo>
<!-- camera -->

<gazebo reference="camera_link">
<sensor type="camera" name="intel_realsense_r200">
<update_rate>30.0</update_rate>
<camera name="head">

<horizontal_fov>1.02974</horizontal_fov>
<image>

<width>1920</width>
```

```
<height>1080</height>

<format>R8G8B8</format>

</image>

<clip>
<near>0.02</near>

<far>300</far>
</clip>
<noise>
<type>gaussian</type>

<!-- Noise is sampled independently per pixel on each frame.

That pixel's noise value is added to each of its color channels,

which at that point lie in the range [0,1].
-->

<mean>0.0</mean>

<stddev>0.007</stddev>
</noise>

</camera>
<plugin name="camera_controller" filename=" libgazebo_ros_cam-
  era.so">
<alwaysOn>true</alwaysOn>

<updateRate>0.0</updateRate>
<cameraName>camera/color</cameraName>

<imageTopicName>image_raw</imageTopicName>
<cameraInfoTopicName>camera_info</cameraInfoTopicName>
<frameName>camera_link</frameName>

<hackBaseline>0.07</hackBaseline>

<distortionK1>0.0</distortionK1>

<distortionK2>0.0</distortionK2>

<distortionK3>0.0</distortionK3>
<distortionT1>0.0</distortionT1>
<distortionT2>0.0</distortionT2>
</plugin>

</sensor>

</gazebo>
</robot>
```

A.5 open_manipulator_controller.cpp

```

/*
*****
**
* Copyright 2018 ROBOTIS CO., LTD.
*
* Licensed under the Apache License, Version
* 2.0 (the " License");
*
* you may not use this file except in com-
* pliance with the License.
*
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or
* agreed to in writing, software
*
* distributed under the License is distrib-
* uted on an " AS IS" BASIS,
*
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
* express or implied.
* See the License for the specific lan-
* guage governing permissions and
*
* limitations under the License.
*****
****
*/
/*
Authors: Darby Lim, Hye-Jong KIM, Ryan Shim,
Yong-Ho Na */
#include "open_manipulator_controller/
open_manipulator_controller.h"
using namespace open_manipulator_controller;
OpenManipulatorController::OpenManipulatorCon-
troller( std::string usb_port, std::string
baud_rate)
: node_handle_(""),

priv_node_handle_("~"),

timer_thread_state_(false)

{

/*
*****
** Initialize ROS parameters
*****

```

```

    */
    control_period_ = priv_node_handle_.param<double>("
control_period", 0.010f);
    using_platform_ = priv_node_handle_.param<bool>(" using_plat-
form", false);
    /*
    *****
    ** Initialize variables
    *****
    */

    open_manipulator_.initOpenManipulator(using_platform_,
        usb_port, baud_rate, control_period_);
    if (using_platform_ == true) log::info("Succeeded to
        init " + priv_node_handle_.getNamespace());

    else if (using_platform_ == false) log::info("Ready to simu-
        late " + priv_node_handle_.getNamespace() + " on
        Gazebo");
    /*
    *****

    ** Initialize ROS publishers, subscribers and servers
    *****
    */
    initPublisher(); initSub-
scriber();

    initServer();
}
OpenManipulatorController::~OpenManipulatorController()
{
    timer_thread_state_ = false;

    pthread_join(timer_thread_, NULL); // Wait
        for the thread associated with thread_p
        to complete

    log::info("Shutdown OpenManipulator Control-
ler"); open_manipulator_.disableAllActua-
tor(); ros::shutdown();

}
void OpenManipulatorController::startTimerThread()
{
    //

```

```

////////////////////////////////////
/
/// Use this when you want to increase the priority
    of threads.
//
////////////////////////////////////
/
// pthread_attr_t attr_;
// int error;

// struct sched_param param;
// pthread_attr_init(&attr_);
// error = pthread_attr_setschedpolicy(&attr_, SCHED_RR
);
// if (error != 0) log::error(" pthread_attr_setschedpolicy
    error = ", (double)error
    );

// error = pthread_attr_setinheritsched(&attr_, PTHREAD_EX-
    PLICIT_SCHED);

// if (error != 0) log::error(" pthread_attr_setinheritsched
    error = ", (double) error);
// memset(&param, 0, sizeof(param));

// param.sched_priority = 31; // RT
// error = pthread_attr_setschedparam(&attr_, &param);

// if (error != 0) log::error(" pthread_attr_setschedparam
    error = ", (double)error)
    ;
// if ((error = pthread_create(&this->timer_thread_, & attr_,
    this->timerThread, this)) != 0)
// {
// log::error("Creating timer thread failed!!", (double
    )error);

// exit(-1);
// }
// timer_thread_state_ = true;
//
////////////////////////////////////
/
int error;
if ((error = pthread_create(&this->timer_thread_, NULL, this->timerThread,
    this)) != 0)
{
log::error("Creating timer thread failed!!",
    (double) error);
exit(-1);
}
timer_thread_state_ = true;

```

```

}
void *OpenManipulatorController::timer-
    Thread(void * param)
{
    OpenManipulatorController *controller = ( Open-
        ManipulatorController *) param;

    static struct timespec

    next_time; static struct

    timespec curr_time;

    clock_gettime(CLOCK_MONOTONIC, &next_time);
    while(controller->timer_thread_state_)
    {
        next_time.tv_sec += (next_time.tv_nsec + ((int)( controller-
            >getControlPeriod() * 1000)) * 1000000) /
            1000000000;

        next_time.tv_nsec = (next_time.tv_nsec + ((int)( controller-
            >getControlPeriod() * 1000)) * 1000000) %
            1000000000;
        double time = next_time.tv_sec + (next_time.tv_nsec
            *0.000000001);

        controller->process(time);
        clock_gettime(CLOCK_MONOTONIC, &curr_time);
        /////
        double delta_nsec = controller->getControlPeriod() - ((
            next_time.tv_sec - curr_time.tv_sec) + ((double)(
            next_time.tv_nsec - curr_time.tv_nsec)*0.000000001))
            ;

        //log::info("control time : ", controller-> getCon-
            trolPeriod() - delta_nsec);

        if (delta_nsec > controller->getControlPeriod())
        {
            //log::warn("Over the control time : ", delta_nsec);
            next_time = curr_time;
        }

        else
        clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &
            next_time, NULL);
        /////
    }
    return 0;
}
/*
*****
**

```

```

** Init Functions

*****
****
    */

void OpenManipulatorController::initPublisher()
{
    // ros message publisher

    auto om_tools_name = open_manipulator_.getManipulator()
        ->getAllToolComponentName();
    for (auto const& name:om_tools_name)
    {
        ros::Publisher pb;

        pb = node_handle_.advertise<open_manipulator_msgs:: Kinemat-
            icsPose>(name + "/kinematics_pose", 10);

        open_manipulator_kinematics_pose_pub_.push_back(pb);
    }
    JointState>("joint_states",
        10);
    }

    else

    {
        auto gazebo_joints_name = open_manipulator_. getManipula-
            tor()->getAllActiveJointComponentName();

        gazebo_joints_name.reserve(gazebo_joints_name.size() +
            om_tools_name.size());

        gazebo_joints_name.insert(gazebo_joints_name.end(),
            om_tools_name.begin(), om_tools_name.end());
        for (auto const& name:gazebo_joints_name)
        {
            ros::Publisher pb;

            pb = node_handle_.advertise<std_msgs::Float64>(name + "
                _position/command", 10); gazebo_goal_joint_posi-

            tion_pub_.push_back(pb);

        }
    }

    void OpenManipulatorController::initSubscriber()
    {
        // ros message subscriber

```

```
open_manipulator_option_sub_ = node_handle_.subscribe(" option",
    10, &OpenManipulatorController:: openManipulatorOption-
    Callback, this);
}
void OpenManipulatorController::initServer()
{
goal_joint_space_path_server_ = node_handle_. advertiseServ-
    ice("goal_joint_space_path", & OpenManipulatorControl-
    ler:: goalJointSpacePathCallback, this);

goal_joint_space_path_to_kinematics_pose_server_ = node_han-
    dle_.advertiseService(" goal_joint_space_path_to_kinemat-
    ics_pose", & OpenManipulatorController::
    goalJointSpacePathToKinematicsPoseCallback, this);

goal_joint_space_path_to_kinematics_position_server_ = node_han-
    dle_.advertiseService(" goal_joint_space_path_to_kinemat-
    ics_position", & OpenManipulatorController::
    goalJointSpacePathToKinematicsPositionCallback, this
    );

goal_joint_space_path_to_kinematics_orientation_server_
    = node_handle_.advertiseService("
    goal_joint_space_path_to_kinematics_orientation", & OpenManip-
    ulatorController:: goalJointSpacePathToKinematicsOrientation-
    Callback,
    this);
goal_task_space_path_server_ = node_handle_. advertiseServ-
    ice("goal_task_space_path", & OpenManipulatorControl-
    ler::goalTaskSpacePathCallback
    , this);

goal_task_space_path_position_only_server_ = node_handle_.adver-
    tiseService(" goal_task_space_path_position_only", & OpenMa-
    nipulatorController:: goalTaskSpacePathPositionOnlyCallback,
    this);

goal_task_space_path_orientation_only_server_ = node_handle_.ad-
    vertiseService(" goal_task_space_path_orientation_only", &
    OpenManipulatorController:: goalTaskSpacePathOrientationOn-
    lyCallback, this);

goal_joint_space_path_from_present_server_ = node_handle_.adver-
    tiseService(" goal_joint_space_path_from_present", & OpenMa-
    nipulatorController:: goalJointSpacePathFromPresentCallback,
    this);

goal_task_space_path_from_present_server_ = node_handle_.adver-
    tiseService(" goal_task_space_path_from_present", & OpenManip-
    ulatorController:: goalTaskSpacePathFromPresentCallback,
    this);

goal_task_space_path_from_present_position_only_server_
```

```

        = node_handle_.advertiseService("
goal_task_space_path_from_present_position_only", & Open-
ManipulatorController:: goalTaskSpacePathFromPresentPosi-
tionOnlyCallback, this);

goal_task_space_path_from_present_orientation_only_server_
= node_handle_.advertiseService("
goal_task_space_path_from_present orienta-
tion_only", &OpenManipulatorController::
goalTaskSpacePathFromPresentOrientationOn-
lyCallback, this);
goal_tool_control_server_ = node_handle_. adver-
tiseService("goal_tool_control", & OpenManipu-
latorController::goalToolControlCallback,
this);

set_actuator_state_server_ = node_handle_. ad-
vertiseService("set_actuator_state", & Open-
ManipulatorController::setActuatorStateCallb-
ack,
this);

goal_drawing_trajectory_server_ = node_handle_.
advertiseService("goal_drawing_trajectory",
& OpenManipulatorController:: goalDraw-
ingTrajectoryCallback, this);
}
/*
*****
**
** Callback Functions for ROS Subscribers

*****
****
*/
void OpenManipulatorController:: openManipula-
torOptionCallback(const std_msgs::String
::ConstPtr &msg)
{
if (msg->data == "print_open_manipulator_setting")
open_manipulator_.printManipulatorSetting();
}
/*
*****
**
** Callback Functions for ROS Servers

*****
****
*/
bool OpenManipulatorController::
goalJointSpacePathCallback(open_manipula-
tor_msgs:: SetJointPosition::Request &req,

```

```

open_manipulator_msgs::SetJointPosition::Response &res)
{
    std::vector<double> target_angle;
    for (int i = 0; i < req.joint_position.joint_name.size (); i++)

    target_angle.push_back(req.joint_position.position.at(i));
    if (!open_manipulator_.makeJointTrajectory(target_angle
        ,

    req.path_time))

    res.is_planned =

    false;

    else
    res.is_planned = true;
    return true;
}
bool OpenManipulatorController:: goalJointSpacePathToKinematicsPoseCallback( open_manipulator_msgs::SetKinematicsPose::Request & req,

open_manipulator_msgs::SetKinematicsPose::Response &res
)
{
    KinematicPose target_pose;

    target_pose.position[0] = req.kinematics_pose.pose. position.x;

    target_pose.position[1] = req.kinematics_pose.pose. position.y;

    target_pose.position[2] = req.kinematics_pose.pose. position.z;
    Eigen::Quaterniond q(req.kinematics_pose.pose. orientation.w,

    req.kinematics_pose.pose.orientation.x, req.kinematics_pose.pose.orientation.y, req.kinematics_pose.pose.orientation.z);

    target_pose.orientation = math:: convertQuaternionToRotationMatrix(q);
    if (!open_manipulator_.makeJointTrajectory(req. end_effector_name, target_pose, req.path_time))
    res.is_planned = false;
    else

```

```

    res.is_planned = true;
    return true;
}

bool OpenManipulatorController:: goalJointSpacePathToKinemat-
    icsPositionCallback( open_manipulator_msgs::SetKinemat-
        icsPose::Request & req,

open_manipulator_msgs::SetKinematicsPose::Response &res
    )
{
    KinematicPose target_pose;

    target_pose.position[0] = req.kinematics_pose.pose. position.x;

    target_pose.position[1] = req.kinematics_pose.pose. position.y;

    target_pose.position[2] = req.kinematics_pose.pose. position.z;
    if (!open_manipulator_.makeJointTrajectory(req. end_effec-
        tor_name, target_pose.position, req. path_time))
        res.is_planned = false;
    else
        res.is_planned = true;
    return true;
}

bool OpenManipulatorController:: goalJointSpacePathToKinemat-
    icsOrientationCallback( open_manipulator_msgs::SetKinemat-
        icsPose::Request & req,

open_manipulator_msgs::SetKinematicsPose::Response &res
    )
{
    KinematicPose target_pose;
    Eigen::Quaterniond q(req.kinematics_pose.pose. orientation.w,

    req.kinematics_pose.pose.orientation.x, req.kinemat-

    ics_pose.pose.orientation.y, req.kinematics_pose.pose.orienta-

    tion.z);

    target_pose.orientation = math:: convertQuaternionToRotationMa-
        trix(q);
    if (!open_manipulator_.makeJointTrajectory(req. end_effec-
        tor_name, target_pose.orientation, req.
        path_time))
        res.is_planned = false;
    else
        res.is_planned = true;
    return true;
}

```

```

bool OpenManipulatorController:: goalTaskSpace-
    PathCallback(open_manipulator_msgs:: SetKinematicsPose::Re-
        quest &req,

open_manipulator_msgs::SetKinematicsPose::Response &res
    )
{
    KinematicPose target_pose;

    target_pose.position[0] = req.kinematics_pose.pose. position.x;

    target_pose.position[1] = req.kinematics_pose.pose. position.y;

    target_pose.position[2] = req.kinematics_pose.pose. position.z;
    Eigen::Quaterniond q(req.kinematics_pose.pose. orientation.w,

        req.kinematics_pose.pose.orientation.x, req.kinemat-

        ics_pose.pose.orientation.y,

        req.kinematics_pose.pose.orientation.z);
    target_pose.orientation = math:: convertQuaternionToRotationMa-
        trix(q);
    if (!open_manipulator_.makeTaskTrajectory(req. end_effector_name,
        target_pose, req.path_time))

    res.is_planned = false;
    else
    res.is_planned = true;
    return true;
}

bool OpenManipulatorController:: goalTaskSpacePathPositionOn-
    lyCallback( open_manipulator_msgs::SetKinematicsPose::Re-
        quest & req,

open_manipulator_msgs::SetKinematicsPose::Response &res
    )
{
    Eigen::Vector3d position;

    position[0] = req.kinematics_pose.pose.position.x; posi-

    tion[1]  =  req.kinematics_pose.pose.position.y; posi-

    tion[2] = req.kinematics_pose.pose.position.z;

    if (!open_manipulator_.makeTaskTrajectory(req. end_effector_name,
        position, req.path_time))
    res.is_planned = false;
    else
    res.is_planned = true;
    return true;
}

```

```

    }
    bool OpenManipulatorController:: goalTaskSpacePathOrienta-
        tionOnlyCallback( open_manipulator_msgs::SetKinemat-
            icsPose::Request & req,

    open_manipulator_msgs::SetKinematicsPose::Response &res
        )
    {
        Eigen::Quaterniond q(req.kinematics_pose.pose. orientation.w,

        req.kinematics_pose.pose.orientation.x, req.kinemat-

        ics_pose.pose.orientation.y, req.kinematics_pose.pose.orienta-

        tion.z);

        Eigen::Matrix3d orientation = math:: convertQuaternionToRotation-
            Matrix(q);
        if (!open_manipulator_.makeTaskTrajectory(req. end_effector_name,
            orientation, req.path_time))

        res.is_planned = false;
        else
        res.is_planned = true;
        return true;
    }
    bool OpenManipulatorController:: goalJointSpacePath-
        FromPresentCallback( open_manipulator_msgs::SetJointPosi-
            tion::Request & req,

    open_manipulator_msgs::SetJointPosition::Response &res)
    {
        std::vector <double> target_angle;
        for (int i = 0; i < req.joint_position.joint_name.size (); i
            ++))

        target_angle.push_back(req.joint_position.position.at(i
            ));
        if (!open_manipulator_. makeJointTrajectoryFromPresentPosi-
            tion(target_angle,
                req.path_time))

        res.is_planned = false;

        else
        res.is_planned = true;
        return true;
    }
    bool OpenManipulatorController:: goalTaskSpacePath-
        FromPresentCallback( open_manipulator_msgs::SetKinemat-
            icsPose::Request & req,

```

```

open_manipulator_msgs::SetKinematicsPose::Response &res
)
{
    KinematicPose target_pose;

    target_pose.position[0] = req.kinematics_pose.pose. position.x;

    target_pose.position[1] = req.kinematics_pose.pose. position.y;

    target_pose.position[2] = req.kinematics_pose.pose. position.z;
    Eigen::Quaterniond q(req.kinematics_pose.pose. orientation.w,
    req.kinematics_pose.pose.orientation.x, req.kinemat-
    ics_pose.pose.orientation.y, req.kinematics_pose.pose.orien-
    tation.z);

    target_pose.orientation = math:: convertQuaternionToRotation-
    Matrix(q);
    if (!open_manipulator_.
    makeTaskTrajectoryFromPresentPose(req.planning_group
    , target_pose, req.path_time))
    res.is_planned = false;
    else
    res.is_planned = true;
    return true;
}
bool OpenManipulatorController:: goalTaskSpacePath-
    FromPresentPositionOnlyCallback( open_manipula-
    tor_msgs::SetKinematicsPose::Request & req,

    open_manipulator_msgs::SetKinematicsPose::Response &res
    )
    {
        Eigen::Vector3d position;

        position[0] = req.kinematics_pose.pose.position.x; posi-
        tion[1] = req.kinematics_pose.pose.position.y; posi-
        tion[2] = req.kinematics_pose.pose.position.z;

        if (!open_manipulator_. makeTaskTrajectoryFromPresent-
        Pose(req.planning_group
        , position, req.path_time))

        res.is_planned = false;

```

```

else
res.is_planned = true;
return true;
}
bool OpenManipulatorController:: goalTaskSpacePathFromPresent-
    tOrientationOnlyCallback( open_manipulator_msgs::SetKine-
        maticsPose::Request & req,

open_manipulator_msgs::SetKinematicsPose::Response &res
    )
{
Eigen::Quaterniond q(req.kinematics_pose.pose. orientation.w,

req.kinematics_pose.pose.orientation.x, req.kinemat-

ics_pose.pose.orientation.y, req.kinematics_pose.pose.orienta-

tion.z);

Eigen::Matrix3d orientation = math:: convertQuaternionToRotation-
    Matrix(q);
if (!open_manipulator_. makeTaskTrajectoryFromPresent-
    Pose(req.planning_group
        , orientation, req.path_time)) res.is_planned

= false;

else
res.is_planned = true;
return true;
}
bool OpenManipulatorController::goalToolControlCallback
    (open_manipulator_msgs::SetJointPosition::Request & req,

open_manipulator_msgs::SetJointPosition::Response &res)
{
for (int i = 0; i < req.joint_position.joint_name.size (); i
    ++)
{
if (!open_manipulator_.makeToolTrajectory(req. joint_posi-
    tion.joint_name.at(i), req.joint_position. position.at(i)))

res.is_planned = false;
else
res.is_planned = true;
}
return true;
}
bool OpenManipulatorController:: setActuatorState-
    Callback(open_manipulator_msgs::
SetActuatorState::Request &req,
open_manipulator_msgs::SetActuatorState::Response &res)

```

```
{
  if (req.set_actuator_state == true) // enable actuators
  {
    log::println("Wait a second for actuator enable", "
      GREEN");

    timer_thread_state_ = false; pthread_join(timer_thread_,
      NULL); // Wait for the
      thread associated with thread_p to complete

    open_manipulator_.enableAllActuator(); startTimerThread();

  }
  else // disable actuators
  {
    log::println("Wait a second for actuator disable", "
      GREEN");

    timer_thread_state_ = false; pthread_join(timer_thread_,
      NULL); // Wait for the
      thread associated with thread_p to complete

    open_manipulator_.disableAllActuator(); startTimer-
      Thread();

  }
  res.is_planned = true;
  return true;
}

bool OpenManipulatorController:: goalDrawingTrajecto-
  ryCallback(open_manipulator_msgs
    ::SetDrawingTrajectory::Request &req,

    open_manipulator_msgs::SetDrawingTrajectory::Response & res)
{
  try

  {
    if (req.drawing_trajectory_name == "circle")
    {
      double draw_circle_arg[3];
```

```

draw_circle_arg[0] = req.param[0]; // radius (m) draw_cir-
cle_arg[1] = req.param[1]; // revolution (rev) draw_circle_arg[2]
= req.param[2]; // start angle
    position (rad)

void* p_draw_circle_arg = &draw_circle_arg;
if (!open_manipulator_.makeCustomTrajectory( CUSTOM_TRAJEC-
    TORY_CIRCLE, req.end_effector_name, p_draw_circle_arg,
    req.path_time))

res.is_planned = false;
else
res.is_planned = true;
}
else if (req.drawing_trajectory_name == "line")
{
TaskWaypoint draw_line_arg;

draw_line_arg.kinematic.position(0) = req.param[0]; // x axis
    (m)

draw_line_arg.kinematic.position(1) = req.param[1]; // y axis
    (m)

draw_line_arg.kinematic.position(2) = req.param[2]; // z axis
    (m)

void *p_draw_line_arg = &draw_line_arg;
if (!open_manipulator_.makeCustomTrajectory( CUSTOM_TRAJEC-
    TORY_LINE, req.end_effector_name, p_draw_line_arg,
    req.path_time))

res.is_planned = false;
else
res.is_planned = true;
}
else if (req.drawing_trajectory_name == "rhombus")
{
double draw_rhombus_arg[3];

draw_rhombus_arg[0] = req.param[0]; // radius (m)
draw_rhombus_arg[1] = req.param[1]; // revolution (rev)
draw_rhombus_arg[2] = req.param[2]; // start angle
    position (rad)

void* p_draw_rhombus_arg = &draw_rhombus_arg;
if (!open_manipulator_.makeCustomTrajectory( CUSTOM_TRAJEC-
    TORY_RHOMBUS, req.end_effector_name, p_draw_rhombus_arg,
    req.path_time))

```

```

    res.is_planned = false;
    else
    res.is_planned = true;
    }
    else if (req.drawing_trajectory_name == "heart")
    {
    double draw_heart_arg[3];

    draw_heart_arg[0] = req.param[0]; // radius (m) draw_heart_arg[1]

    = req.param[1]; // revolution (rev) draw_heart_arg[2] =

    req.param[2]; // start angle
        position (rad)

    void* p_draw_heart_arg = &draw_heart_arg;
    if (!open_manipulator_.makeCustomTrajectory( CUSTOM_TRAJEC-
        TORY_HEART, req.end_effector_name, p_draw_heart_arg,
        req.path_time))

    res.is_planned = false;
    else
    res.is_planned = true;
    }
    return true;
    }
    catch ( ros::Exception &e )
    {
    log::error("Creation the custom trajectory is failed!")
        ;
    }
    return true;
    }
    /*
    *****
    **
    ** Callback function for process timer

    *****
    *****
        */

    void OpenManipulatorController::process(double time)
    {
    open_manipulator_.processOpenManipulator(time);
    }
    /*
    *****
    **
    ** Callback function for publish timer

```

```

*****
****
    */
void OpenManipulatorController::publishCallback(const ros::TimerEvent&)
{
    if (using_platform_ == true) publishJointStates();
    else publishGazeboCommand();
    publishOpenManipulatorStates();
    publishKinematicsPose();
}
void OpenManipulatorController::publishOpenManipulatorStates()
{
    open_manipulator_msgs::OpenManipulatorState msg; if
    (open_manipulator_.getMovingState()) msg.open_manip-
    ulator_moving_state = msg.IS_MOVING;

    else
    msg.open_manipulator_moving_state = msg.STOPPED;
    if (open_manipulator_.getActuatorEnabledState(
        JOINT_DYNAMIXEL))
    msg.open_manipulator_actuator_state = msg.
        ACTUATOR_ENABLED;
    else
    msg.open_manipulator_actuator_state = msg.
        ACTUATOR_DISABLED;
    open_manipulator_states_pub_.publish(msg);
}
void OpenManipulatorController::publishKinematicsPose()
{
    open_manipulator_msgs::KinematicsPose msg;

    auto om_tools_name = open_manipulator_.getManipulator()
        ->getAllToolComponentName();
    uint8_t index = 0;

    for (auto const& tools:om_tools_name)
    {
        KinematicPose pose = open_manipulator_.getKinematicPose (tools);
        msg.pose.position.x = pose.position[0]; msg.pose.position.y =
        pose.position[1];

        msg.pose.position.z = pose.position[2];
        Eigen::Quaterniond orientation = math::
        convertRotationMatrixToQuaternion(pose.orientation);

```

```

msg.pose.orientation.w    = orientation.w();

msg.pose.orientation.x    = orientation.x();

msg.pose.orientation.y    = orientation.y();

msg.pose.orientation.z    = orientation.z();

open_manipulator_kinematics_pose_pub_.at(index).publish (msg);
index++;
}

}

void OpenManipulatorController::publishJointStates()
{
    sensor_msgs::JointState msg; msg.header.stamp

= ros::Time::now();

    auto joints_name = open_manipulator_.getManipulator()-> getAllAc-
        tiveJointComponentName();

    auto tool_name = open_manipulator_.getManipulator()-> getAllTool-
        ComponentName();
    auto joint_value = open_manipulator_.
        getAllActiveJointValue();
    auto tool_value = open_manipulator_.getAllToolValue();
    for (uint8_t i = 0; i < joints_name.size(); i++)
    {
        msg.name.push_back(joints_name.at(i));
        msg.position.push_back(joint_value.at(i).position);

        msg.velocity.push_back(joint_value.at(i).velocity);

        msg.effort.push_back(joint_value.at(i).effort);

    }
    for (uint8_t i = 0; i < tool_name.size(); i++)
    {
        msg.name.push_back(tool_name.at(i));
        msg.position.push_back(tool_value.at(i).position);
        msg.velocity.push_back(0.0f); msg.effort.push_back(0.0f);

    }
    open_manipulator_joint_states_pub_.publish(msg);
}

void OpenManipulatorController::publishGazeboCommand()
{
    JointWaypoint joint_value = open_manipulator_. getAllAc-
        tiveJointValue();

```

```

JointWaypoint tool_value = open_manipulator_. getAllToolValue();
for (uint8_t i = 0; i < joint_value.size(); i++)
{
    std_msgs::Float64 msg;

    msg.data = joint_value.at(i).position;
    gazebo_goal_joint_position_pub_.at(i).publish(msg);
}
for (uint8_t i = 0; i < tool_value.size(); i++)
{
    std_msgs::Float64 msg;

    msg.data = tool_value.at(i).position;
    gazebo_goal_joint_position_pub_.at(joint_value.size() + i).publish(msg);
}
}
}
/*
*****
**
** Main
*****
****
    */

int main(int argc, char **argv)
{
    // init
    ros::init(argc, argv, "open_manipulator_controller");
    ros::NodeHandle node_handle("");
    std::string usb_port =

    "/dev/ttyUSB0"; std::string

    baud_rate = "1000000";

    if (argc = 3)
    {
        usb_port =

        argv[1];

        baud_rate =

        argv[2];

        printf("port_name and baud_rate are set to %s, %s \n",
            usb_port.c_str(), baud_rate.c_str());
    }

    else
    {

```

```

log::error("Please set '-port_name' and '-baud_rate'
arguments for connected Dynamixels");
return 1;
}
OpenManipulatorController om_controller(usb_port, baud_rate);
// update om_controller.start-

TimerThread();

ros::Timer publish_timer = node_handle.createTimer(ros
::Duration(om_controller.getControlPeriod()), & OpenManipula-
torController::publishCallback, & om_controller);

ros::Rate loop_rate(100);

while (ros::ok())

{
ros::spinOnce();

loop_rate.sleep();

}
return 0;
}

```

A.6 joint_trajectory_controller.yaml

```

arm_controller:
type: position_controllers/JointTrajectoryController
joints:
  * joint1
  * joint2
  * joint3
  * joint4
  * joint5
  * jo
int6 con-
straints:
  goal_time: 0.6
  stopped_velocity_tolerance: 0.05
  joint1: {trajectory: 0.1, goal: 0.1}
  joint2: {trajectory: 0.1, goal: 0.1}
  joint3: {trajectory: 0.1, goal: 0.1}
  joint4: {trajectory: 0.1, goal: 0.1}
  joint5: {trajectory: 0.1, goal: 0.1}
  joint6: {trajectory: 0.1, goal: 0.1}
stop_trajectory_duration: 0.5
state_publish_rate: 25
action_monitor_rate: 10

```

A.7 kinematics.yaml

```
arm:
  kinematics_solver: trac_ik_kinematics_plugin/
  TRAC_IKKinematicsPlugin kinematics_solver_timeout:
  0.005
  solve_type: Distance
```

A.8 chomp_planning.yaml

```
planning_time_limit: 10.0
max_iterations: 200
max_iterations_after_collision_free: 5
smoothness_cost_weight: 0.1
obstacle_cost_weight: 1.0
learning_rate: 0.01
smoothness_cost_velocity: 0.0
smoothness_cost_acceleration: 1.0
smoothness_cost_jerk: 0.0
ridge_factor: 0.01 use_pseudo_inverse: false pseudo_inverse_ridge_factor: 1e-4
joint_update_limit: 0.1
collision_clearance: 0.2
collision_threshold: 0.07
use_stochastic_descent: true enable_failure_recovery: true
max_recovery_attempts: 5
```

A.9 config/chomp_planning.yaml

```
# joint_limits.yaml allows the dynamics properties
# specified in the URDF to be overwritten or augmented as
# needed
# Specific joint properties can be changed with the keys
# [max_position, min_position, max_velocity, max_acceleration]
# Joint limits can be turned off with [ has_velocity_limits,
# has_acceleration_limits]
joint_limits:
  joint1:
    has_velocity_limits: true max_velocity: 4.8 has_acceleration_limits:
    false max_acceleration: 0
  joint2: has_velocity_limits:
    true max_velocity: 4.8
    has_acceleration_limits: false max_acceleration: 0
  joint3: has_velocity_limits:
    true max_velocity: 4.8
    has_acceleration_limits: false max_acceleration: 0
```

```

joint4: has_velocity_limits:
  true max_velocity: 4.8
  has_acceleration_limits: false max_ac-
    celeration: 0
joint5: has_velocity_limits:
  true max_velocity: 4.8
  has_acceleration_limits: false max_ac-
    celeration: 0
joint6: has_velocity_limits:
  true max_velocity: 4.8
  has_acceleration_limits: false
max_acceleration: 0

```

A.10 config/open_manipulator.srdf

```

<?xml version="1.0" ?>
<!--This does not replace URDF, and is not an extension of
  URDF.
  This is a format for representing semantic information
  about the robot structure.
  A URDF file must exist for this robot as well, where
  the joints and the links that are referenced are de-
  fined
-->
<robot name="open_manipulator">
  <!--GROUPS: Representation of a set of joints and
  links. This can be useful for specifying DOF to plan
  for, defining arms, end effectors, etc-->
  <!--LINKS: When a link is specified, the parent joint
  of that link (if it exists) is automatically included--
  >
  <!--JOINTS: When a joint is specified, the child link
  of that joint (which will always exist) is automatically
  included-->
  <!--CHAINS: When a chain is specified, all the links
  along the chain (including endpoints) are included in the
  group. Additionally, all the joints that are parents to
  included links are also included
  . This means that joints along the chain and the par-
  ent joint of the base link are included in the group--
  >
  <!--SUBGROUPS: Groups can also be formed by referenc-
  ing to already defined group names-->
  <group name="arm">
    <joint name="world_fixed" />
    <joint name="joint1" />
    <joint name="joint2" />
    <joint name="joint3" />
    <joint name="joint4" />
    <joint name="joint5" />
    <joint name="joint6" />
    <joint name="camera_joint" />
  </group>

```

```

    <!--GROUP STATES: Purpose: Define a named state for a par-
    ticular group, in terms of joint values. This
    is useful to define states like 'folded arms'-->
        <group_state name="zero" group="arm">
            <joint name="joint1" value="0" />
<joint name="joint2" value="0" />
<joint name="joint3" value="0" />
            <joint name="joint4" value="0" />
            <joint name="joint5" value="0" />
            <joint name="joint6" value="0" />
        </group_state>
        <group_state name="home" group="arm">
            <joint name="joint1" value="0" />
            <joint name="joint2" value="-1" />
            <joint name="joint3" value="0.3" />
            <joint name="joint4" value="0" />
            <joint name="joint5" value="0.7" />
            <joint name="joint6" value="0" />
        </group_state>
    <!--END EFFECTOR: Purpose: Represent information about
    an end effector.-->
        <end_effector name="End Effector" parent_link=" link7"
        group="arm" parent_group="arm" />
    <!--DISABLE COLLISIONS: By default it is assumed that
    any link of the robot could potentially come into colli-
    sion with any other link in the robot. This tag disables
    collision checking between a specified pair of links. -->
        <disable_collisions link1="ground_plane_box" link2=
        "link1" reason="Adjacent" />
        <disable_collisions link1="ground_plane_box" link2=
        "link2" reason="Never" />
        <disable_collisions link1="ground_plane_box" link2=
        "link3" reason="Never" />
        <disable_collisions link1="link1" link2="link2" reason="Adja-
        cent" />
        <disable_collisions link1="link1" link2="link3" reason="Never"
        />
        <disable_collisions link1="link2" link2="link3" reason="Adja-
        cent" />
        <disable_collisions link1="link3" link2="link4" reason="Adja-
        cent" />
        <disable_collisions link1="link3" link2="link5" reason="Never"
        />
        <disable_collisions link1="link3" link2="link6" reason="Never"
        />
        <disable_collisions link1="link4" link2="link5" reason="Adja-
        cent" />
        <disable_collisions link1="link4" link2="link6" reason="Never"
        />
        <disable_collisions link1="link4" link2="link7"
        reason="Never" />
        <disable_collisions link1="link5" link2="link6" reason="Adja-
        cent" />

```

```

    <disable_collisions link1="link5" link2="link7" reason="Never"
  />
    <disable_collisions link1="link6" link2="link7" reason="Adja-
cent" />
  </robot>

```

A.11 config/ros_controllers.yaml

```

# Simulation settings for using moveit_sim_controllers
moveit_sim_hw_interface:
  joint_model_group: arm
  joint_model_group_pose: zero
# Settings for ros_control_boilerplate control loop ge-
neric_hw_control_loop:
  loop_hz: 300
  cycle_time_error_threshold: 0.01
# Settings for ros_control hardware interface hardware_interface:
joints:
  * joint1
  * joint2
  * joint3
  * joint4
  * joint5
  * joint6
  sim_control_mode: 1 # 0: position, 1: velocity # Publish
all joint states
# Creates the /joint_states topic necessary in ROS
joint_state_controller:
  type: joint_state_controller/JointStateController publish_rate:
  50
controller_list:
  * name: arm_controller
    action_ns: follow_joint_trajectory de-
    fault: True
    type: FollowJointTrajectory
    joints:
      - joint1
      - joint2
      - joint3
      - joint4
      - joint5
      - joint6

```

A.12 config/sensors_3d.yaml

```

# The name of this file shouldn't be changed, or else the
Setup Assistant won't detect it
sensors:
  - filtered_cloud_topic: filtered_cloud max_range:
    5.0
    max_update_rate: 1.0
    padding_offset: 0.1
    padding_scale: 1.0

```

```

    point_cloud_topic: /cloud_map # dependent on RTAB-
Map topic point_sub-
    sample: 1
    sensor_plugin: occupancy_map_monitor/
PointCloudOctomapUpdater

```

A.13 Orocos_vs_trac_testing.py

```

# Import the module
from urdf_parser_py.urdf import URDF
from pykdl_utils.kdl_parser import
    kdl_tree_from_urdf_model
from pykdl_utils.kdl_kinematics import KDLKinematics from
trac_ik_python.trac_ik import IK
from hrl_geom.pose_converter import PoseConv
from random import random
from math import sqrt
from geometry_msgs.msg import Pose #
forwawrd kinematics
def kdl_tester(pose):
    global robot
    kdl_kin = KDLKinematics(robot, "link1", "link7") q_ik
    = kdl_kin.inverse(pose) # inverse kinematics,
    pose needs to be homogenous numpy.mat return
    q_ik != None

def trac_tester(pose):
    global robot
    trac_kin = IK(robot, "link1", "link7") joint_goal =
    trac_kin.get_ik(seed_state, pose[0],
    pose[1], pose[2], pose[3], pose[4], pose[5], pose [6])
    return joint_goal != None
robot = URDF.from_parameter_server()
randquat = [[0, 0, 0, 0]]*1000
idx = 0
trac_count = 0
kdl_count = 0
for idx in length(randquat):
    x = 2*random() - 1
    y = 2*random() - 1
    while (x*x + y*y) >=1:
        x = 2*random() -
        1
        y = 2*random() - 1
    z = x*x + y*y
    # done with step 1
    u = 2*random() - 1
    v = 2*random() - 1
    while (u*u + v*v) >=1:
        u = 2*random() - 1
        v = 2*random() - 1
    w = u*u + v*v
    # done with step
    2 s = sqrt((1-
    z)/w)

```

```
# quat = x, y, su, sv
randquat[idx][0] = x
randquat[idx][1] = y
randquat[idx][2] = s*u
randquat[idx][3] = s*v
# quat matrix now generated, time to call for arbitrary xyz
# want in cube from one corner to another, of length 10 idx =
0
for x in range(10):
    for y in
        range(10):
            for z in
                range(10): x =
                    x-5
                    y = y-5
                    z = z-5
                    p = [x, y, z, randquat[idx][0], randquat[idx][1],
randquat[idx][2], randquat[idx][3]]
                    idx+=1
                    mat = PoseConv.to_homo_mat(p)
                    if kdl_tester(mat):
                        kdl_count+=1
                    if trac_tester(p):
                        trac_count+=1
print("KDL Count : " + kdl_count)
print("TRAC Count : " + trac_count)
```

Appendix B: Launch Files

This appendix includes the launch files that needed to be modified to accommodate the motion planner changes, IK changes, and hardware changes during the course of this project.

B.1 move_group.launch

```
<launch>
<!-- GDB Debug Option -->
<arg name="debug" default="false" />
<arg unless="$(arg debug)" name="launch_prefix" value="" />
<arg if="$(arg debug)" name="launch_prefix" value="gdb -x
$(find
open_manipulator_moveit_config)/launch/gdb_settings. gdb --
ex run --args" />

<!-- Verbose Mode Option -->
<arg name="info" default="$(arg debug)" />
<arg unless="$(arg info)" name="command_args" value=" " />
<arg if="$(arg info)" name="command_args" value="--
debug" />

<!-- move_group settings -->
<arg name="pipeline" default="chomp" />
<arg name="allow_trajectory_execution" default="true" />
<arg name="fake_execution" default="false"/>
<arg name="execution_type" default="interpolate"/> <!--
-- set to 'last point' to skip intermediate trajectory
in fake execution -->
<arg name="max_safe_path_cost" default="1"/>
<arg name="jiggle_fraction" default="0.05" />
<arg name="publish_monitored_planning_scene" default="true"/>

<arg name="capabilities" default=""/>
<arg name="disable_capabilities" default=""/>
<!-- load these non-default MoveGroup capabilities (
space seperated) -->
<!--
<arg name="capabilities" value="
a_package/ AwesomeMotionPlanningCapabil-
ity
another_package/GraspPlanningPipeline "
/>

-->

<!-- inhibit these default MoveGroup capabilities ( space
seperated) -->
```

```

<!--
<arg name="disable_capabilities" value="
    move_group/MoveGroupKinematicsService
    move_group/ClearOctomapService
    " />

-->

<arg name="load_robot_description" default="true" />
<!-- load URDF, SRDF and joint_limits configuration
-->
<include file="$(find open_manipulator_moveit_config)
/launch/planning_context.launch">
    <arg name="load_robot_description" value="$(arg load_robot_de-
scription)" />
</include>

<!-- Planning Functionality -->
<include ns="move_group" file="$(find open_manipula-
tor_moveit_config)/launch/ planning_pipeline.launch.xml">
    <arg name="pipeline" value="$(arg pipeline)" />
    <param name="capabilities" value="$(arg capabili-
ties)"/>
    <param name="disable_capabilities" value="$(arg disable_capa-
bilities)"/>
</include>

<!-- Trajectory Execution Functionality -->
<include ns="move_group" file="$(find open_manipula-
tor_moveit_config)/launch/ trajectory_execution.launch.xml"
if="$(arg allow_trajectory_execution)">
    <arg name="moveit_manage_controllers" value="true"
/>
    <arg name="moveit_controller_manager" value=" open_manipu-
lator" unless="$(arg fake_execution)"/>
    <arg name="moveit_controller_manager" value="fake"
if="$(arg fake_execution)"/>
    <arg name="execution_type" value="$(arg execution_type)"
/>
</include>

<!-- Sensors Functionality -->
<include ns="move_group" file="$(find open_manipula-
tor_moveit_config)/launch/ sensor_manager.launch.xml"
if="$(arg allow_trajectory_execution)">
    <arg name="moveit_sensor_manager" value=" open_manipula-
tor" />
</include>

<!-- Start the actual move_group node/action server
-->
<node name="move_group" launch-prefix="$(arg launch_prefix)"
pkg="moveit_ros_move_group" type=" move_group" respawn="false"
output="screen" args="$( arg command_args)">

```

```

    <!-- Set the display variable, in case OpenGL code is used
internally -->
    <env name="DISPLAY" value="$(optenv DISPLAY :0)" />

    <param name="allow_trajectory_execution" value="$( arg al-
low_trajectory_execution)"/>
    <param name="max_safe_path_cost" value="$(arg
max_safe_path_cost)"/>
    <param name="jiggle_fraction" value="$(arg jiggle_frac-
tion)" />

    <!-- Publish the planning scene of the physical robot so
that rviz plugin can know actual robot -->
    <param name="planning_scene_monitor/ publish_plan-
ning_scene" value="$(arg publish_monitored_planning_scene)"
/>
    <param name="planning_scene_monitor/ publish_geometry_up-
dates" value="$(arg publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/
publish_state_updates" value="$(arg
publish_monitored_planning_scene)" />
    <param name="planning_scene_monitor/ publish_transforms_up-
dates" value="$(arg publish_monitored_planning_scene)" />
</node>
<!-- Load kinematics yaml file, this is necessary due to
stupid moveit issues with loading param server from
private nodes -->
<group ns="move_group">
    <rosparam command="load" file="$(find open_manipula-
tor_moveit_config)/config/kinematics. yaml"/>
</group>
</launch>

```

B.2 chomp_planning_pipeline.launch.xml

```

<launch>
<!-- CHOMP Plugin for MoveIt! -->
<arg name="planning_plugin" value="chomp_interface/ CHOMPPPlanner"
/>

<!-- define capabilities that are loaded on start ( space
seperated) -->
<arg name="capabilities" default=""/>

<!-- inhibit capabilities (space seperated) -->
<arg name="disable_capabilities" default=""/>

<arg name="start_state_max_bounds_error" value="0.1"
/>

<!-- The request adapters (plugins) used when planning.
ORDER MATTERS -->
<arg name="planning_adapters" value="default_planner_re-
quest_adapters/
AddTimeParameterization

```

```

        default_planner_request_adapters/ ResolveCon-
straintFrames
        default_planner_request_adapters/ FixWorkspace-
Bounds
        default_planner_request_adapters/
FixStartStateBounds
FixStartStateCollision
        default_planner_request_adapters/ FixStartState-
PathConstraints"
    />

<param name="planning_plugin" value="$(arg planning_plugin)"
/>
<param name="request_adapters" value="$(arg planning_adapt-
ers)" />
<param name="start_state_max_bounds_error" value="$( arg
start_state_max_bounds_error)" />
<param name="capabilities" value="$(arg capabilities) " />
<param name="disable_capabilities" value="$(arg disable_capabili-
ties)" />
    <rosparam command="load" file="$(find open_manipula-
tor_moveit_config)/config/ chomp_planning.yaml" />
</launch>

```

B.3 open_manipulator_moveit_controller_manager.launch.xml

```

<launch>
  <!-- loads moveit_controller_manager on the parameter server
        which is taken as argument
        if no argument is passed, moveit_simple_controller_man-
ager will be set -->
  <arg name="moveit_controller_manager" default=" moveit_sim-
ple_controller_manager/ MoveItSimpleControllerManager" />
  <param name="moveit_controller_manager" value="$(arg
moveit_controller_manager)"/>
  <arg name="execution_type" default="interpolate"/>
  <param name="execution_type" value="$(arg execution_type)"/>
  <!-- loads ros_controllers to the param server -->
  <rosparam file="$(find open_manipulator_moveit_config
)/config/ros_controllers.yaml"/>
</launch>

```

Appendix C: Custom Package Files

This appendix includes the source, launch, and other package files that belong to the custom `open_nav_pkg` package. These files can be copied into the catkin workspace and built using the commands detailed above.

C.1 `patterns.py`

```
#!/usr/bin/env python
# Python 2/3 compatibility imports

from future import print_function

from six.moves import input

import traceback

import tf

import sys

import

copy im-

port rospy

import moveit_commander

import moveit_msgs.msg

import geometry_msgs.msg

# trac_ik import for transform from task space to joint
from trac_ik_python.trac_ik import IK
# useful for quaternion math

from tf.transformations import euler_from_quaternion,
    quaternion_from_euler
try:
from math import pi, tau, dist, fabs, cos ex-

cept: # For Python 2 compatibility from math

import pi, fabs, cos, sqrt

tau = 2.0*pi
def
dist(p, q):
```

```

return sqrt(sum((p_i - q_i)**2.0 for p_i, q_i in zip(p, q)))

from std_msgs.msg import String

from moveit_commander.conversions import pose_to_list
def main():

    global move_group, robot, scene
    moveit_commander.roscpp_initialize(sys.argv)

    rospy.init_node('open_manipulator_nav', anonymous=True)

    ## Instantiate a 'RobotCommander' object. Provides information
    ## such as the robot's

    ## kinematic model and the robot's current joint states robot =
    moveit_commander.RobotCommander()

    ## Instantiate a 'PlanningSceneInterface' object. This provides
    ## a remote interface

    ## for getting, setting, and updating the robot's

    internal understanding of the ## surrounding world:

    scene = moveit_commander.PlanningSceneInterface()
    ## Instantiate a 'MoveGroupCommander' object. This object
    ## is an interface

    ## to a planning group (group of joints). In this tutorial
    ## the group is the primary

    ## arm joints in the Panda robot, so we set the group's name
    ## to "panda_arm".

    ## If you are using a different robot, change this value
    ## to the name of your robot

    ## arm planning group.

    ## This interface can be used to plan and execute motions:

    group_name = "arm"

    move_group = moveit_commander.MoveGroupCommander(
        group_name)

    init_pose = move_group.get_current_pose().pose

    print("Initial Pose : \n
        ")

```

```
print(init_pose)

inp = input("Sweep? [y/n/L] :").lower()

if inp == "y":

    n = int(input("Enter n : "))

    x = float(input("Enter x : "))
    y = float(input("Enter y : "))
    z = float(input("Enter z : "))

    rectangle_sweep(n, x, y, z) elif

inp == "n":

    x = float(input("Enter x : "))
    y = float(input("Enter y : "))
    z = float(input("Enter z : "))

    rectangle(x, y, z)

else:
    n = int(input("Enter n : "))
    x = float(input("Enter x : "))
    y = float(input("Enter y : "))
    z = float(input("Enter z : "))

    rectangle_corner_sweep(n, x, y, z)

## NOTE: All paths assume that the bot starts from the home
    position

#xyz = forwards, left, up

# rectangle is four points, rectangle goes CW def

rectangle(dx=0, dy=0.15, dz=0.15): global

move_group

init_pose = move_group.get_current_pose().pose

ix = init_pose.position.x

iy = init_pose.position.y
iz = init_pose.position.z
iox = init_pose.orientation.x
```

```
ioy = init_pose.orientation.y

ioz = init_pose.orientation.z

iow = init_pose.orientation.w

#p1
go_to_pose_goal(ix+dx, iy+dy, iz, iox, ioy, ioz, iow)
print("p1")
#p2
go_to_pose_goal(ix+dx, iy-dy, iz, iox, ioy, ioz, iow)
print("p2")
#p3
go_to_pose_goal(ix+dx, iy-dy, iz-2*dz, iox, ioy, ioz, iow)

print("p3")
#p4
go_to_pose_goal(ix+dx, iy+dy, iz-2*dz, iox, ioy, ioz, iow)

print("p4")
#center
go_to_pose_goal(ix, iy, iz, iox, ioy, ioz, iow)
# this path follows a rectangle corner-wise, but sweeps
  horizontally every n lines incrementally (verti-
  cally)

def rectangle_sweep(n=5, dx=0, dy=0.1, dz=0.08): global
move_group

init_pose = move_group.get_current_pose().pose
ix = init_pose.position.x

iy = init_pose.position.y

iz = init_pose.position.z

iox = init_pose.orientation.x

ioy = init_pose.orientation.y

ioz = init_pose.orientation.z

iow = init_pose.orientation.w for c in range(n):

go_to_pose_goal(ix+dx, iy+dy, iz-float(c)/float(n)*dz, iox,
  ioy, ioz, iow)

go_to_pose_goal(ix+dx, iy-dy, iz-float(c)/float(n)*dz, iox,
  ioy, ioz, iow)

# center after for loop
```

```

go_to_pose_goal(ix, iy, iz, iox, ioy, ioz, iow)
# for a given rectangle, we want to be able to sweep
  around corners.
def rectangle_corner_sweep(n=5, dx=0, dy=0.1, dz=0.1):
global move_group
init_pose = move_group.get_current_pose().pose
ix = init_pose.position.x

iy = init_pose.position.y

iz = init_pose.position.z

iox = init_pose.orientation.x

ioy = init_pose.orientation.y

ioz = init_pose.orientation.z

iow = init_pose.orientation.w

(iroll, ipitch, iyaw) = euler_from_quaternion([iox, ioy, ioz,
iow])

py = quaternion_from_euler(iroll, ipitch, iyaw-pi/2) # positive
  y quaternion, head facing in negative y direction

ny = quaternion_from_euler(iroll, ipitch, iyaw+pi/2) # negative
  y quaternion, head facing in positive y direction

print(init_pose)
print(py)

print(ny)

for c in range(n):

go_to_pose_goal(ix+dx, iy+dy, iz-float(c)/float(n)*dz, py[0],
  py[1], py[2], py[3])

go_to_pose_goal(ix+dx, iy+dy, iz-float(c)/float(n)*dz,
iox, ioy, ioz, iow)
go_to_pose_goal(ix+dx, iy-dy, iz-float(c)/float(n)*dz, iox,
  ioy, ioz, iow)

go_to_pose_goal(ix+dx, iy-dy, iz-float(c)/float(n)*dz, ny[0],
  ny[1], ny[2], ny[3])

go_to_pose_goal(ix, iy, iz, iox, ioy, ioz, iow)
# assumes original quaternion orientation is preserved #@param

du: the minimum incremental difference

def go_along_line(x, y, z, du):

```

```
global move_group

init_pose = move_group.get_current_pose().pose
ix = init_pose.position.x

iy = init_pose.position.y

iz = init_pose.position.z

iox = init_pose.orientation.x

ioy = init_pose.orientation.x

ioz = init_pose.orientation.x

iow = init_pose.orientation.x

# current xyz
cx = ix
cy = iy

cz = iz
total_pose_count = max([abs((x-ix)/du), abs((y-iy)/du),
                        abs((z-iz)/du)])

print(total_pose_count)

pose_count = 0

# est incremental difference for each direction
dx = (x - ix) / total_pose_count

dy = (y - iy) / total_pose_count

dz = (z - iz) / total_pose_count

print(dx)

print(dy)

print(dz)

# while not within range +-du of xyz

# same as (cx > x + du or cx < x - du) or (cy > y + du or
      cy < y - du) or (cz > z + du or cz < z - du)

while pose_count < total_pose_count: go_to_pose_goal(cx+dx,

cy+dy, cz+dz, iox, ioy, ioz, iow
```

```

    )
    cx+=dx

    cy+=dy

    cz+=dz

    pose_count+=1
    def go_to_pose_goal(x, y, z, ox, oy, oz, ow):
    # Copy class variables to local variables to make the
    # web tutorials more clear.

    # In practice, you should use the class variables directly
    # unless you have a good

    # reason not to.

    ## BEGIN_SUB_TUTORIAL plan_to_pose

    global move_group

    ## Planning to a Pose Goal

    ## ~~~~~

    ## We can plan a motion for this group to a desired pose
    # for the

    ## end-effector:

    #pose_goal = geometry_msgs.msg.Pose()

    ik_solver = IK("link1", "link7") # initialize TRAC_IK
    solver

    seed_state = [0.0] * ik_solver.number_of_joints # should
    # be a matrix of N joints

    joint_goal = ik_solver.get_ik(seed_state, x, y, z, ox, oy,
    oz, ow)

    ## Now, we call the planner to compute the plan and exe-
    # cute it.

    move_group.go(joint_goal, wait=True)

    # Calling 'stop()' ensures that there is no residual movement

    move_group.stop()
    # this is useful for checking for deviations
    pose_goal = geometry_msgs.msg.Pose()

```

```

pose_goal.orientation.w = ow pose_goal.orien-
tation.x = ox pose_goal.orientation.y = oy
pose_goal.orientation.z = oz pose_goal.posi-
tion.x = x pose_goal.position.y = y
pose_goal.position.z = z

current_pose = move_group.get_current_pose().pose
return all_close(pose_goal, current_pose, 0.01)
def all_close(goal, actual, tolerance):
    all_equal = True

    if type(goal) is list:
        for index in range(len(goal)):
            if abs(actual[index] - goal[index]) > tolerance: return
                False

    elif type(goal) is geometry_msgs.msg.PoseStamped: return
        all_close(goal.pose, actual, tolerance)

    elif type(goal) is geometry_msgs.msg.Pose:
        return all_close(pose_to_list(goal), pose_to_list(
            actual), tolerance)
    return True

if name == " main ":
    try:

        while True:

            main()

    except rospy.ROSInterruptException:
        traceback.print_exc() ex-

    cept KeyboardInterrupt:

        exit()

```

C.2 pose_array_nav.py

```

#!/usr/bin/env python
# Python 2/3 compatibility imports

from future import print_function

```

```

from six.moves import input
import traceback
import tf
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
# trac_ik import for transform from task space to joint from
trac_ik_python.trac_ik import IK
# useful for quaternion math
from tf.transformations import euler_from_quaternion,
    quaternion_from_euler

try:
    from math import pi, tau, dist, fabs, cos
except:
    # For Python 2 compatibility
    from math import pi, fabs, cos, sqrt
    tau = 2.0*pi
    def dist(p, q):
        return sqrt(sum((p_i - q_i)**2.0 for p_i, q_i in zip(p,q)))
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list

def main():
    global move_group, ik_solver, robot, scene,
    allPoses
    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('open_manipulator_nav', anonymous=
    True)
    ## Instantiate a 'RobotCommander' object. Provides information
    such as the robot's
    ## kinematic model and the robot's current joint states
    robot = moveit_commander.RobotCommander()

    ## Instantiate a 'PlanningSceneInterface' object.
    This provides a remote interface
    ## for getting, setting, and updating the robot's internal
    understanding of the
    ## surrounding world:
    scene = moveit_commander.PlanningSceneInterface()

    ## Instantiate a 'MoveGroupCommander' object.
    This object is an interface
    ## to a planning group (group of joints). In this tutorial
    the group is the primary
    ## arm joints in the Panda robot, so we set the group's
    name to "panda_arm".
    ## If you are using a different robot, change this value
    to the name of your robot
    ## arm planning group.

```

```

    ## This interface can be used to plan and execute mo-
    tions:
    group_name = "arm"
    move_group = moveit_commander.MoveGroupCommander(
        group_name)
    init_pose = move_group.get_current_pose().pose all-
    Poses = []
    ik_solver = IK("link1", "link7")

    rospy.init_node("open_manipulator_nav", anonymous=
    True)
    rospy.Subscriber("/pose_array", geometry_msgs.msg.
    PoseArray, callback)
    rospy.spin()

def
    callback(dat
    a): global
    allPoses
    # want to find poses that are in data but not in allPoses
    for pose in data.poses: if
        pose in allPoses:
            pas
        s else:
            allPoses.append(pose) go_to_pose_goal(pose.posi-
            tion.x, pose.
            position.y, pose.position.z, pose.orientation.x, pose.orienta-
            tion.y, pose.orientation.z,
            pose.orientation.w)
            # end go_to_pose_goal statement

def go_to_pose_goal(x, y, z, ox, oy, oz, ow):
    # Copy class variables to local variables to make the
    web tutorials more clear.
    # In practice, you should use the class variables di-
    rectly unless you have a good
    # reason not to.
    ## BEGIN_SUB_TUTORIAL plan_to_pose
    global move_group, ik_solver
    ## Planning to a Pose Goal
    ## ~~~~~
    ## We can plan a motion for this group to a desired pose
    for the
    ## end-effector:
    #pose_goal = geometry_msgs.msg.Pose()
    seed_state = [0.0] * ik_solver.number_of_joints #
    should be a matrix of N joints
    joint_goal = ik_solver.get_ik(seed_state, x, y, z,
    ox, oy, oz, ow)
    ## Now, we call the planner to compute the plan and execute
    it.
    move_group.go(joint_goal, wait=True)
    # Calling 'stop()' ensures that there is no residual
    movement

```

```

    move_group.stop()
    # this is useful for checking for deviations
    pose_goal = geometry_msgs.msg.Pose()
    pose_goal.orientation.w = ow pose_goal.orientation.x = ox pose_goal.orientation.y = oy
    pose_goal.orientation.z = oz pose_goal.position.x = x pose_goal.position.y = y
    pose_goal.position.z = z
    current_pose = move_group.get_current_pose().pose return

    all_close(pose_goal, current_pose, 0.01)

def all_close(goal, actual, tolerance): all_equal
    = True
    if type(goal) is list:
        for index in range(len(goal)):
            if abs(actual[index] - goal[index]) > tolerance:

                return False

        elif type(goal) is geometry_msgs.msg.PoseStamped: return all_close(goal.pose, actual, tolerance)

    elif type(goal) is geometry_msgs.msg.Pose: return all_close(pose_to_list(goal),
pose_to_list(actual), tolerance) re-

    turn True

if __name__ == "__main__":
    try:

main()

```

C.3 CMakeLists.txt

```

cmake_minimum_required(VERSION 3.0.2)
project(open_nav_pkg)
## Compile as C++11, supported in ROS Kinetic and newer #
add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages find_package(catkin
REQUIRED COMPONENTS
    geometry_msgs open_manipula-
    tor_controllers rospy
    std_msgs
    trac_ik_python

)

```

```

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)

## Uncomment this if the package has a setup.py. This macro
  ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/
  setup_dot_py.html
# catkin_python_setup()

#####
## Declare ROS messages, services and actions ##
#####

## To declare and build messages, services or actions from
  within this
## package, follow these steps:
## * Let MSG_DEP_SET be the set of packages whose message
  types you use in
## your messages/services/actions (e.g. std_msgs, ac-
  tionlib_msgs, ...).
## * In the file package.xml:
## * add a build_depend tag for "message_generation" ## *
add a build_depend and a exec_depend tag for
each package in MSG_DEP_SET
  ## * If MSG_DEP_SET isn't empty the following dependency
    has been pulled in
  ## but can be declared for certainty nonetheless: ##
  * add a exec_depend tag for "message_runtime" ## * In
  this file (CMakeLists.txt):
  ## * add "message_generation" and every package in
    MSG_DEP_SET to
  ## find_package(catkin REQUIRED COMPONENTS ...) ## *
add "message_runtime" and every package in
  MSG_DEP_SET to
  ## catkin_package(CATKIN_DEPENDS ...)
  ## * uncomment the add_*_files sections below as needed
  ## and list every .msg/.srv/.action file to be pro-
    cessed
  ## * uncomment the generate_messages entry below ## *
add every package in MSG_DEP_SET to
  generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#     Mes-
sage1.msg #
Message2.msg #
)

## Generate services in the 'srv' folder
# add_service_files(
#   FILES

```

```

#         Ser-
vicel.srv #
Service2.srv #
)

## Generate actions in the 'action' folder #
add_action_files(
# FILES
#     Action1.ac-
tion #     Ac-
tion2.action # )

## Generate added messages and services with any depend-
encies listed here
# generate_mes-
sages( # DEPEND-
ENCIES
#     geometry_msgs#     std_msgs
# )
#####
## Declare ROS dynamic reconfigure parameters ##
#####

## To declare and build dynamic reconfigure parameters
within this
## package, follow these steps:
## * In the file package.xml:
## * add a build_depend and a exec_depend tag for " dynamic_re-
configure"
## * In this file (CMakeLists.txt):
## * add "dynamic_reconfigure" to
##   find_package(catkin REQUIRED COMPONENTS ...) ## *
uncomment the "
    generate_dynamic_reconfigure_options" section below
##   and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder
# generate_dynamic_reconfigure_options( #     cfg/DynRe-
conf1.cfg
#     cfg/DynReconf2.cfg
# )

#####
## catkin specific configuration ##
#####
## The catkin_package macro generates cmake config files
for your package
## Declare things to be passed to dependent projects ##
INCLUDE_DIRS: uncomment this if your package
contains header files
## LIBRARIES: libraries you create in this project that dependent
projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also
need

```

```

## DEPENDS: system dependencies of this project that dependent
    projects also need
catkin_package(
# INCLUDE_DIRS include # LIBRARIES
open_nav_pkg
# CATKIN_DEPENDS geometry_msgs
open_manipulator_controllers rospy std_msgs
trac_ik_python
# DEPENDS system_lib
)
#####
## Build ##
#####
## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories( # in-
clude
    ${catkin_INCLUDE_DIRS}

)
## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/open_nav_pkg.cpp # )
## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}
    _EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single
    CMake context
## The recommended prefix ensures that target names across
    packages don't collide
# add_executable(${PROJECT_NAME}_node src/ open_nav_pkg_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names,
    the following renames the
## target back to the shorter version for ease of user use
## e.g. "roslaunch someones_pkg node" instead of "roslaunch
    someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES
OUTPUT_NAME node PREFIX "")
## Add cmake target dependencies of the executable ##
same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EX-
    PORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target
    against

```

```

# target_link_libraries(${PROJECT_NAME}_node # ${catkin_LIBRAR-
IES}
# )
#####
## Install ##
#####

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv\_user\_guide/variables.html

## Mark executable scripts (Python etc.) for instal-
    lation
## in contrast to setup.py, you can choose the des-
    tination
# catkin_install_python(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION} # )

## Mark executables for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building\_executables.html
# install(TARGETS ${PROJECT_NAME}_node
#   RUNTIME DESTINATION ${
#     CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark libraries for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building\_libraries.html
# install(TARGETS ${PROJECT_NAME}
#   ARCHIVE DESTINATION ${
#     CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${
#     CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION
#   }
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION} #
FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and
    bag files, etc.)
# in-
stall(FILES
#   #
myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION} # )

```

```
#####
## Testing ##
#####

## Add gtest based cpp test target and link libraries #
catkin_add_gtest(${PROJECT_NAME}-test test/
  test_open_nav_pkg.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

C.4 package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>open_nav_pkg</name>
  <version>0.0.0</version>
<description>The open_nav_pkg package</description>
<!-- One maintainer tag required, multiple allowed, one
  person per tag -->
<!-- Example: -->
<!-- <maintainer email="jane.doe@example.com">Jane
  Doe</maintainer> -->
<maintainer email="benc00@vt.edu">benjaminchristie</
  maintainer>

<!-- One license tag required, multiple allowed, one li-
  cense per tag -->
<!-- Commonly used license strings: -->
<!-- BSD, MIT, Boost Software License, GPLv2, GPLv3
  , LGPLv2.1, LGPLv3 -->
<license>TODO</license>

<!-- Url tags are optional, but multiple are allowed, one
  per tag -->
<!-- Optional attribute type can be: website, bugtracker,
  or repository -->
<!-- Example: -->
<!-- <url type="website">http://wiki.ros.org/
  open_nav_pkg</url> -->

<!-- Author tags are optional, multiple are allowed, one
  per tag -->
<!-- Authors do not have to be maintainers, but could be -->
<!-- Example: -->
<!-- <author email="jane.doe@example.com">Jane Doe</ au-
  thor> -->

<!-- The *depend tags are used to specify dependencies -
  ->
<!-- Dependencies can be catkin packages or system de-
  pendencies -->
```

```

<!-- Examples: -->
<!-- Use depend as a shortcut for packages that are both
  build and exec dependencies -->
<!-- <depend>roscpp</depend> -->
<!-- Note that this is equivalent to the following:
-->
<!-- <build_depend>roscpp</build_depend> -->
<!-- <exec_depend>roscpp</exec_depend> -->
<!-- Use build_depend for packages you need at compile
  time: -->
<!-- <build_depend>message_generation</build_depend
  > -->
<!-- Use build_export_depend for packages you need in order to
  build against this package: -->
<!-- <build_export_depend>message_generation</ build_export_de-
  pend> -->
<!-- Use buildtool_depend for build tool packages: --
  >
<!-- <buildtool_depend>catkin</buildtool_depend> --
  >
<!-- Use exec_depend for packages you need at runtime: -
  ->
<!-- <exec_depend>message_runtime</exec_depend> -->
<!-- Use test_depend for packages you need only for
  testing: -->
<!-- <test_depend>gtest</test_depend> -->
<!-- Use doc_depend for packages you need only for
  building documentation: -->
<!-- <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>geometry_msgs</build_depend>
<build_depend>open_manipulator_controllers</ build_de-
  pend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>trac_ik_python</build_depend>
<build_export_depend>geometry_msgs</ build_export_de-
  pend>
<build_export_depend>open_manipulator_controllers</ build_ex-
  port_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<build_export_depend>trac_ik_python</ build_export_de-
  pend>
<exec_depend>geometry_msgs</exec_depend>
<exec_depend>open_manipulator_controllers</ exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>trac_ik_python</exec_depend>
<!-- The export tag contains other, unspecified, tags
-->
<export>

```

```
<!-- Other tools can request additional information be placed
here -->
</export>
</package>
```

Abbreviations

CHOMP	Covariant Hamiltonian Optimization for Motion Planning
DoF	Degrees of freedom
IK	Inverse kinematics
IMU	Inertial measurement unit
IR	Infrared
KPIECE	Kinematic Planning by Interior-Exterior Cell Exploration
LazyPRM	Lazy Probabilistic Roadmap Planner
LazyPRMstar	Lazy Probabilistic Roadmap Planner using the star strategy
OMPL	Open Motion Planning Library
RGB-D	Red-green-blue and depth
ROS	Robot Operating System
RRT	Rapidly exploring random tree
RTAB-Map	Real-Time Appearance-Based Mapping
SPARS	Sparse Roadmap Spanner
SRDF	Semantic Robot Description Format
STOMP	Stochastic Trajectory Optimization for Motion Planning
URDF	Unified Robot Description Format

REPORT DOCUMENTATION PAGE

1. REPORT DATE November 2023		2. REPORT TYPE Final Technical Report		3. DATES COVERED	
				START DATE FY20	END DATE FY22
4. TITLE AND SUBTITLE Increasing the Degrees of Freedom on a Robot Arm					
5a. CONTRACT NUMBER		5b. GRANT NUMBER		5c. PROGRAM ELEMENT 622146	
5d. PROJECT NUMBER AT9		5e. TASK NUMBER 1		5f. WORK UNIT NUMBER	
6. AUTHOR(S) Benjamin Christie, Cameron Alred, Mike Paquette, and Garry Glaspell					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Engineer Research and Development Center (ERDC) Geospatial Research Laboratory (GRL) 7701 Telegraph Road Alexandria, VA 22315-3864				8. PERFORMING ORGANIZATION REPORT NUMBER ERDC/GRL TR-23-4	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Headquarters, US Army Corps of Engineers Washington, DC 20314-1000			10. SPONSOR/MONITOR'S ACRONYM(S) HQUSACE		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT This report provides an implementation of the moveit-commander Python module to generate trajectories for custom six- and seven-degrees of freedom (DoF) arms. The moveit_setup_assistant package was used to modify an existing five-DoF OpenManipulator-X model to increase its range of motion. Specifically, additional joints were fabricated and mounted to the physical arm. Also, the Unified Robot Description Format files were modified to account for the additional joints. In order to optimize the solvers, many changes to the MOVEit configuration files were made. The changes documented in this report lay the groundwork for leveraging MOVEit to expand the capabilities of low-DoF arms.					
15. SUBJECT TERMS Computer programs--Evaluation; Hydraulic structures--Inspection; Robots--Control systems; Robots--Kinematics					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	SAR		89
19a. NAME OF RESPONSIBLE PERSON				19b. TELEPHONE NUMBER (include area code)	