EMD

Release alpha

Glenn Tan

Jun 30, 2021

ABOUT

A modular and easy to deploy ROS2 manipulation pipeline that integrates perception elements to establish an end-to-end pick and place task.

This package was tested with the easy_perception_deployment ROS2 package, but any other perception system that provides the same ROS2 message in the right topic can work with this package as well.

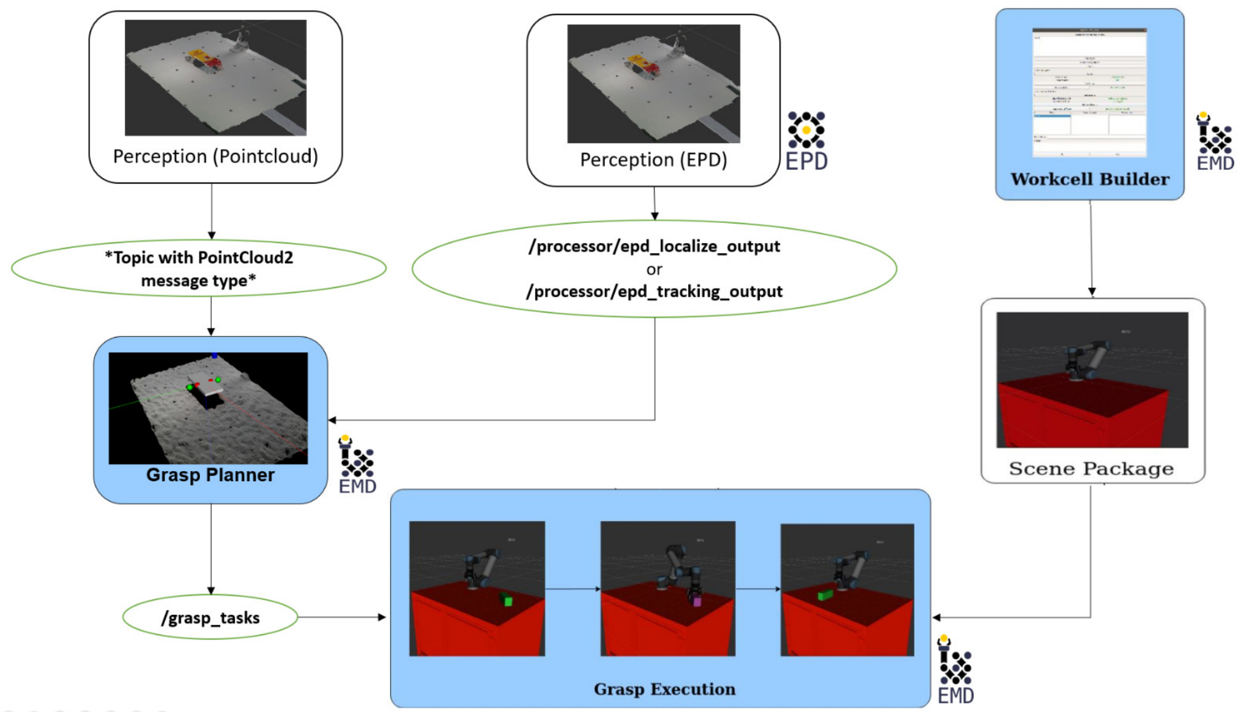
CHAPTER

ONE

OVERVIEW

1.1 Manipulation Pipeline

To preserve the modularity of this package, the manipulation pipeline can be broken down into three main aspects, each of which can function separately, or together as an end to end pipeline. Each component has its own documentation and tutorials which are linked in the headers.



1.1.1 Workcell Builder

The Workcell Builder provides an easy to use GUI that allows users to create a representation of their robot task space to provide robot simulation and to provide the initial state for trajectory planning using frameworks such as Moveit2

1.1.2 Grasp Planner

The Grasp Planner subscribes to a topic published by a perception source and outputs an End-effector specific grasp pose for the end effector using a novel, algorithmic depth-based method. This pose is then published to a ROS2 topic.

1.1.3 Grasp Execution

The Grasp Execution component subscribes to the output published by the Grasp Planner, and uses Moveit2 to develop a collision free trajectory for the robot to navigate to the required grasp object.

Next step: *Download Instructions*

CHAPTER

TWO

FREQUENTLY ASKED QUESTIONS

2.1 Workcell Builder

2.1.1 *How many Robots are supported in workcell generation?*

The current Workcell Builder only supports **one robot and one end effector**. Future plans may involve supporting multiple grippers and robots. You can still manually add them in the scene URDF/SRDF.

2.1.2 *I have my own object_description folders for existing objects that I want to to load into the scene. How do I load it into the Workcell Builder?*

It is currently not possible to load your own objects into the scene. You need to create it in the GUI. Object loading features will be included in the future versions of this package.

2.1.3 *Can I create my own robot and end effector from the Workcell builder?*

The current version of the Workcell builder does not support robot and end effector creation. There are many existing repositories of robots from major robot vendors such as Universal Robots , ABB Fanuc and end effectors from vendors such as Robotiq

2.1.4 *How do I visualize the workspace during editing using the GUI*

Currently, you are not able to visualize the workcell during editing, but rather, using the demo.launch. Future improvements may include a real time visualization of the scene as you change the GUI parameters.

2.2 Grasp Planner

2.2.1 *Can I use my own perception system with this package*

Yes! While it is highly recommended to use the easy_perception_deployment package for seamless integration, you can use your own perception system, but make sure to follow the *Grasp Planner Input Message Types*

2.2.2 *Is it possible to do a side grasp rather than a top down grasp?*

Yes, but your camera would then be required to then face the side which you want to grasp.

CHAPTER

THREE

FUTURE PLANS

Beyond the alpha release, there are some future plans that will be implemented, depending on the time and resources available. These are some of the proposed features that may be included in the beta release in the second release.

3.1 Workcell Builder

3.1.1 Loading of custom environment objects

This feature allows users to upload already existing object_description folders available online, rather than having to always create objects from scratch whenever initializing a scene.

3.1.2 Multi robot support

Future updates will support adding multiple robots into the scene

3.1.3 Real Time Visualization of workcell building

3D visualization of the workcell will be included in the future for users to be able to visualize real time the changes made while in the GUI.

3.2 Grasp Planner

3.2.1 Cross gripper ranking system

Grasp planner will output the best rank across different gripper types

3.3 Grasp Execution

3.3.1 Eye In Hand support

Grasp Execution will support eye-in-hand configuration (camera attached to robot arm)

CHAPTER

FOUR

COMMON CONCEPTS

The following highlights some common concepts and definitions that might be useful to know for new users

Contents

4.1 YAML

YAML (YAML Ain't Markup Language) is a human-readable format for storing data. This is highly used in this package to provide a more understandable description of the scene that can be parsed into URDF files and to be reloaded into the GUI when needed.

More information about the YAML format

4.2 URDF

A URDF (Universal Robot Description Format) file is a file that describes the physical attributes of a robot which will be used by the package

More information about the URDF format

URDF tutorials

4.3 SRDF

A SRDF (Semantic Robot Description Format) file represents the semantic information of the robot that is not typically included in the robot URDF file

More information about the SRDF format

4.4 Moveit_Config Folders

This is a folder that is typically generated by the Moveit Setup Assistant which includes many files that are required for integration with Moveit. This is also where the srdf is stored. These folders are needed for Robots and End Effectors

For a robot named <robot_name> the folder should be named *<robot_name>_moveit_config*

This is the general struction for a typical moveit_config folder

```
|--robot_moveit_config
___|--config
______|--fake_controllers.yaml
______|--robot.srdf.xacro
______|.....other files
___|--launch
.....other files
```

4.5 Description Folders

This folder typically contains the URDF files and the 3D meshes of the object. A description folder is needed not just for robots and end effectors, but also for other static models in the scene.

An object with the name <object_name> will have the description folder named <object_name>_description.

This is the general struction for a typical description folder

```
|--object_name_description
___|--urdf
______|--object_name.urdf.xacro
___|--meshes
______|--collision
_________|--object_collision.stl
______|--visual
_________|--object_visual.stl
```

4.6 Workcell

An environment that involves a robot doing tasks

4.7 Scenes

The visual description of a workcell. This term essentially is interchangeable with a Workcell. However, Scenes and Workcell provides a more all encompassing and understandable term.

4.8 Assets

Represents all visual elements in the scene. This can be split into Robots, End effectors, and Environment Objects

4.8.1 Robots

Any robotic manipulator required for the tasks.

4.8.2 End Effectors

Any end effector attached to the robot. Currently the Grasp Planner only supports a single suction cup gripper and a 2 finger gripper.

4.8.3 Environment Objects

Any other objects that are required in the scene. This includes any tables, boxes, etc that you need added to the scene as static obstacles.

CHAPTER

FIVE

DOWNLOAD INSTRUCTIONS

One of the key features of this package it is semi-modular.

ROS projects generally revolve around usage of URDFs to setup an environment of a robotic workcell. Workcell Builder eases this process by implementing a GUI to generate your desired workcell environment and can be used for other projects as well!

Grasp Planner and Grasp Execution pipeline work hand in hand to provide a seamless pick and place solution with the workcell environment created by Workcell Builder.

Note: Grasp Execution requires a robotic workcell to be set up, so do install the whole EMD package if a pick and place solution is what you're looking for!

5.1 Installing a perception package

The Easy Manipulation Deployment(EMD) package, specifically Grasp Planner, subscribes to either a topic with PointCloud2 message type *OR* topics from easy_perception_deployment(EPD) package.

Warning: EMD is dependent on a perception source. If your camera driver does not provide any PointCloud2 message type topics, do check out the EPD package!

Here *Grasp Planner Input Message Types* is a link for more information to determine which perception input fits best for your requirements.

5.2 Installing complete Easy Manipulation Deployment suite

5.2.1 Installing Easy Manipulation Deployment dependencies

Moveit2

Follow this link Moveit2 to build Moveit2 from source.

Note: The following *Major* EMD dependencies do not require to be built from source and will be installed with rosdep install in the steps below.

Pointcloud Library (PCL) | version: 1.10

The Flexible Collision Library (FCL) | version: 0.5

5.3 Installing only the Workcell Builder

```
mkdir -p ~/workcell_ws/src

cd ~/workcell_ws/src

git clone https://github.com/ros-industrial/easy_manipulation_deployment.git

mv easy_manipulation_deployment/assets/ .

mv easy_manipulation_deployment/scenes/ .

mv easy_manipulation_deployment/easy_manipulation_deployment/workcell_builder ./easy_
↪manipulation_deployment

find ./easy_manipulation_deployment -mindepth 1 ! -regex '^./easy_manipulation_
↪deployment/workcell_builder\(/.*\)?' -delete

cd ~/workcell_ws

source /opt/ros/foxy/setup.bash

rosdep install --from-paths src --ignore-src -yr --rosdistro "${ROS_DISTRO}"

colcon build

source install/setup.bash
```

5.4 Installing entire Easy Manipulation Deployment package

```
mkdir -p ~/workcell_ws/src

cd ~/workcell_ws/src

git clone https://github.com/ros-industrial/easy_manipulation_deployment.git

mv easy_manipulation_deployment/assets/ .

mv easy_manipulation_deployment/scenes/ .

mv easy_manipulation_deployment/easy_manipulation_deployment/workcell_builder ./easy_
↪manipulation_deployment
```

(continues on next page)

(continued from previous page)

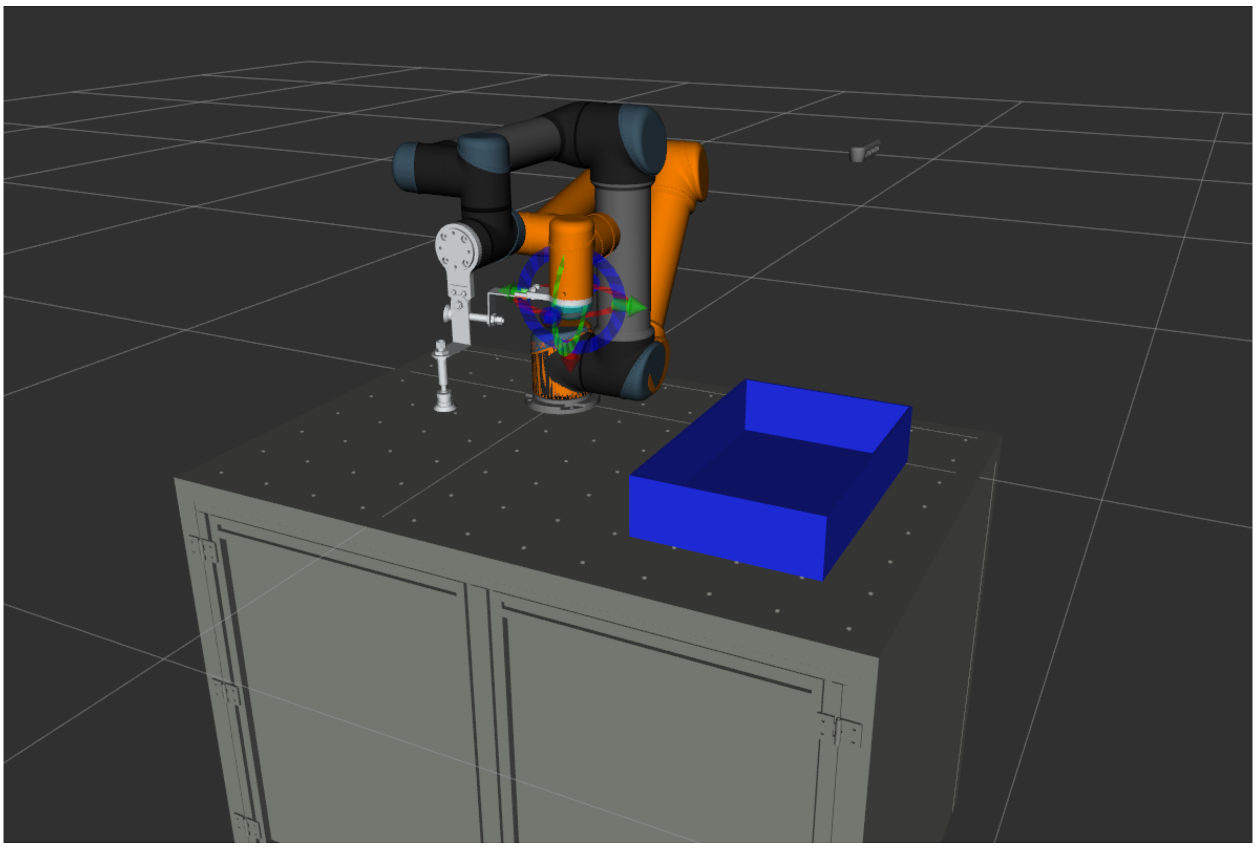
```
cd ~/workcell_ws

source /opt/ros/foxy/setup.bash

rosdep install --from-paths src --ignore-src -yr --rosdistro "${ROS_DISTRO}"

source ~/ws_moveit2/install/setup.bash

colcon build

source install/setup.bash
```

CHAPTER

SIX

WORKCELL BUILDER

This ROS package provides an easy to use Graphical User Interface for generation of a robotic workcell in RViz which serves as the first step in the pipeline for a pick and place task

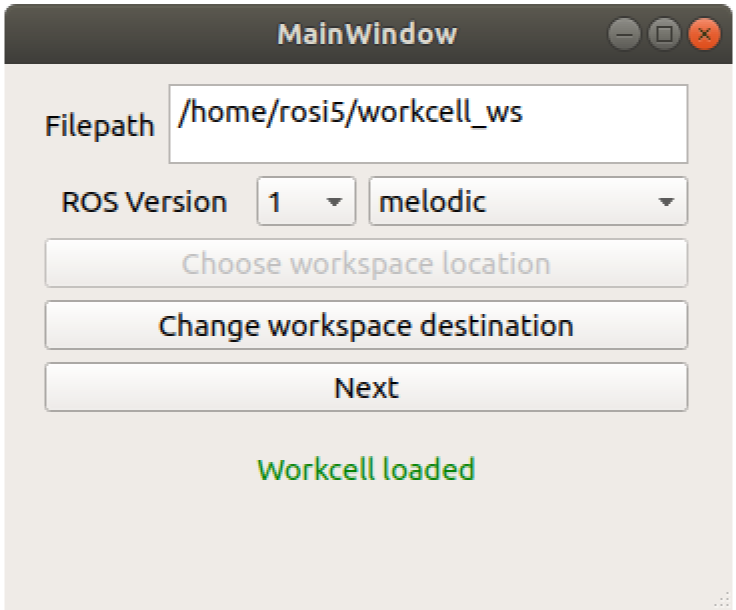


6.1 Workcell Initialization

6.1.1 Folder structure

It is (Highly) recommended to select the same workspace location as the workspace that you lodaded this package in.

From the main window, choose your workspace location. Ensure that the folder selected is the **main workspace folder**, i.e catkin_ws or colcon_ws



Once you see the confirmation message that the workcell is loaded, you can then check that folder using your file explorer and you will see the various folders required to store your assets created. Choose the ROS version and distribution required and click next to be directed to the scene select window

Folder Structure for Assets

This package requires a standardized folder structure in order for the workcell builder to function well. This serves as good practice as well for users to store their files in a logical and standardized format. The following is how your folder should be structured

```
|--workcell_ws
___|--src
______|--scenes
______|--assets
_________|--robots
____________|--robot_brand
_______________|--robot_model_description
_______________|--robot_model_moveit_config
_________|--end_effectors
____________|--end_effector_brand
_______________|--end_effector_model_description
_______________|--end_effector_model_moveit_config
_________|--environment_objects
____________|--environment_objects_description
```

The rest of the documentation will highlight how it should be populated.

6.1.2 Generating Moveit Config packages

One key feature of this package is to generate workcell simulations that is compatible with path planning frameworks. The current version of the workcell builder is designed to be compatible with Moveit, a popular open source motion planning framework

The moveit configuration packages are required if you want to control your robot with the moveit (and the grasp execution component of easy_manipulation_deployment). It is recommended to use the Moveit Setup Assistant it is recommended to use this to generate the package rather than to do it yourself. Below are some existing moveit_config folders

UR Robots

ABB Robots

FOR ROS 2

Note that the setup wizard only generates ROS1 packages for now, so if you are using ROS2, **please convert the moveit_config packages to ROS2 before starting**.

ROS2 Examples (ur5_moveit_config)

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.10.2)
project(ur5_moveit_config)
find_package(ament_cmake REQUIRED)

install(DIRECTORY launch DESTINATION "share/${PROJECT_NAME}")
install(DIRECTORY config DESTINATION "share/${PROJECT_NAME}")
ament_package()
```

package.xml

```
<?xml version="1.0"?>
<package format="3">
  <name>ur5_moveit_config</name>
  <version>0.6.4</version>
  <description>Resources used for MoveIt! testing</description>

  <author email="isucan@willowgarage.edu">Ioan Sucan</author>
  <author email="acorn@willowgarage.edu">Acorn Pooley</author>

  <maintainer email="dave@dav.ee">Dave Coleman</maintainer>

  <license>BSD</license>
  <url type="website">http://moveit.ros.org</url>
  <url type="bugtracker">https://github.com/ros-planning/moveit-resources/issues</url>
  <url type="repository">https://github.com/ros-planning/moveit-resources</url>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <exec_depend>joint_state_publisher</exec_depend>
  <exec_depend>robot_state_publisher</exec_depend>
```

(continues on next page)

(continued from previous page)

```
  <export>
      <build_type>ament_cmake</build_type>
  </export>
</package>
```

6.1.3 Uploading Relevant Assets

Before generating a scene, you need to make sure you have the assets you need for the scene, especially for the robot and end effector.

Robots

For increased reusability and ease of visualization, we will create separate folders for separate vendors of Robots. For example, we will create a folder to store UR robots

```
$ cd workcell_ws/src/assets/robots

$ mkdir universal_robot
```

Copy over the moveit_config folder and description folders of the relevant robot models you want to add, ensuring that the folder names and file names follow the naming *Conventions*

End Effector

Simlarly for End Effectors, we will create a separate folder for each End Effector Vendor. For example, we will create a folder to store Robotiq Grippers

```
$ cd workcell_ws/src/assets/end_effectors

$ mkdir robotiq
```

Copy over the moveit_config folder and description folders of the relevant end_effector models you want to add, ensuring that the folder names and file names follow the naming *Conventions*

Environment Objects

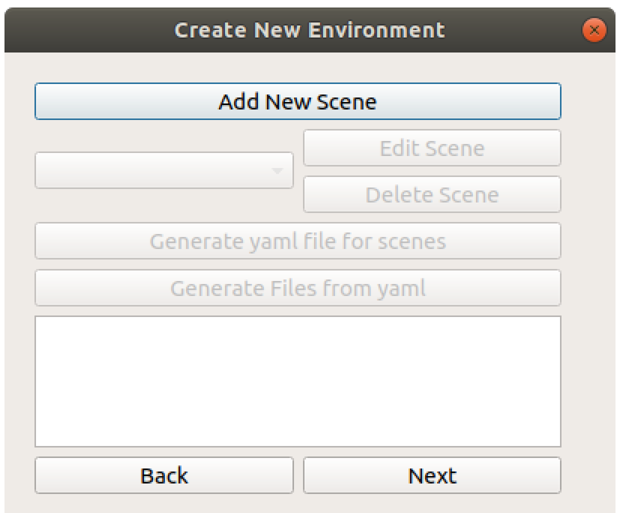
For objects that is part of the environment that will be used as static collision objects, it should be stored in the **workcell_ws/src/assets/environment folder**.

***Current version of the GUI does not support loading of existing environment objects. For simple environment objects, consider creating a copy of the environment objects with the gui instead**

Next step: *Create A Scene*

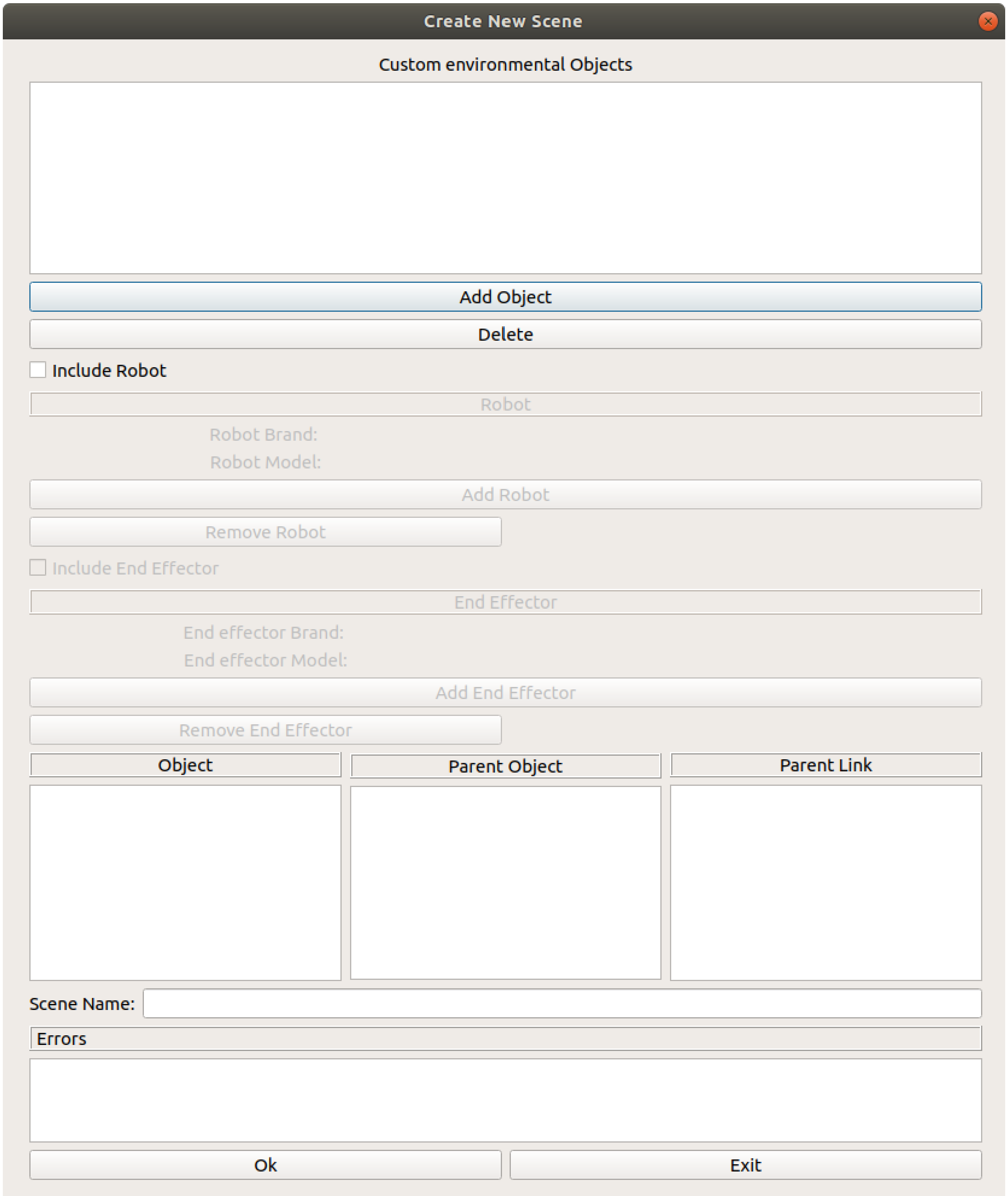
6.2 Create A Scene

Contents



If there are currently no scenes in the "scenes" folder, you need need to add a new scene. Click **Add New Scene** .

You should be shown the window below, which is the start of scene creation.



At this point, there are a few things you can do to populate the scene:

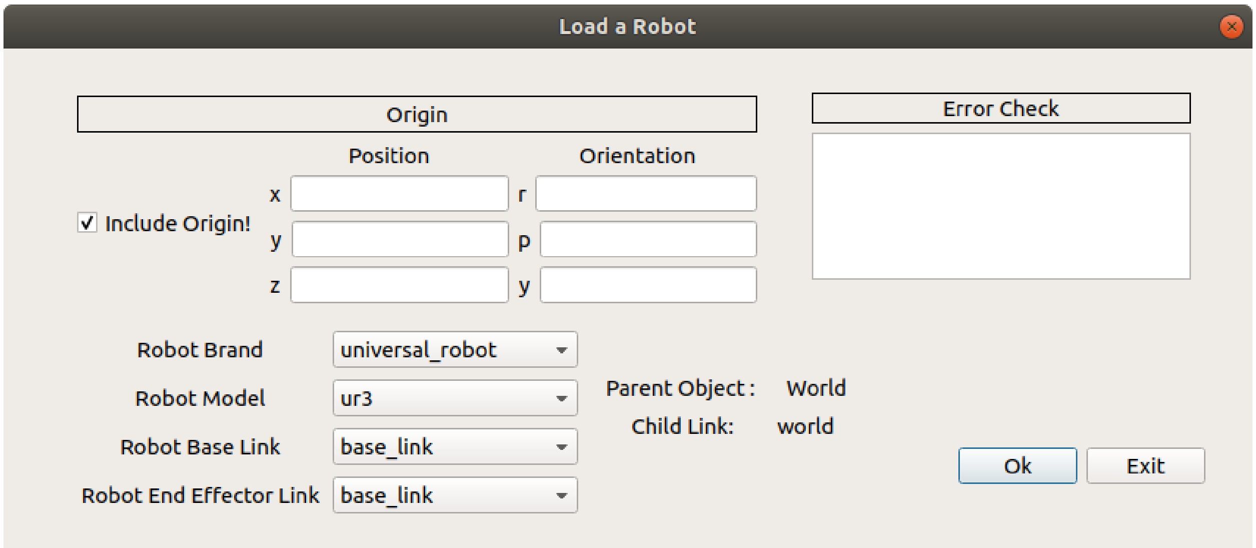
6.2.1 Adding a Robot into scene

To add the robot into the workspace, check the **include robot** box and the add robot button

Current implementation of this GUI assumes that the Robot is connected to the World link. Manual editing of the world link can be done through the URDF.

If the error " No robot is detected in the workcell folder" is seen in the Robot Brand and Robot Model Fields, it means that the robot description folder and moveit_config is not properly loaded. Refer to " Uploading Relevant Assets " in *Workcell Initialization*

Otherwise, you will see the window below



In the dropdown menu, select the robot brand and model you would like to include in the work space

Origin

The Origin is the positional and orientation coordinates values of the robot's **base link with respect to the World Link**. Unchecking the box defaults the XYZ and RPY coordinates to 0

Robot Base Link

This is the link of the robot that will be connected to the World Link

Robot End Effector Link

This is the link of the robot that will be connected to the End Effector Base Link

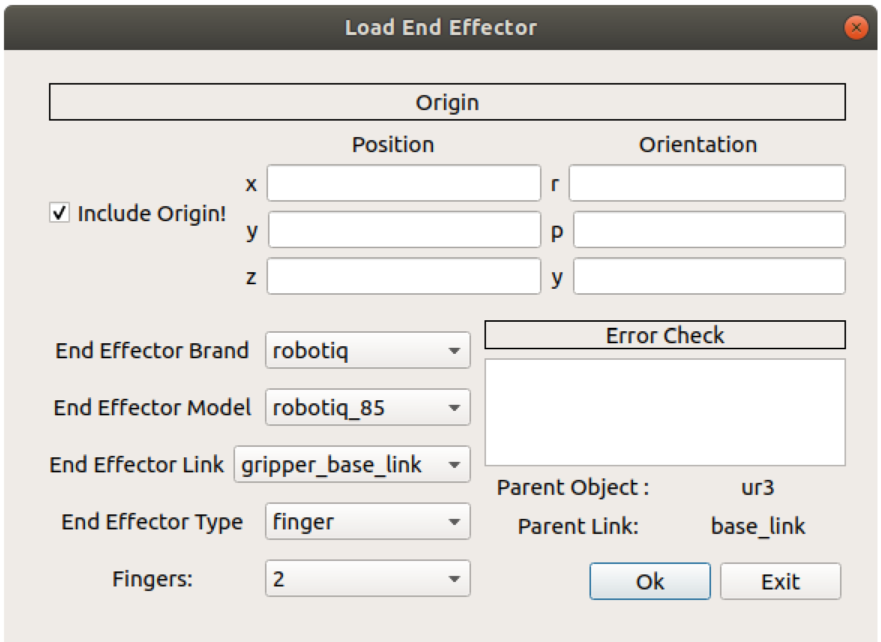
6.2.2 Adding an End Effector into scene

To add the end effector into the workspace, check the **include end effector** box and the **add end effector** button

Note that the end effector can only be included if the Robot is successfully loaded into the scene.

If the error " No end effector is detected in the workcell folder" is seen in the End Effector Brand and End Effector Model Fields, it means that the end effector description folder and moveit_config folder is not properly loaded. Refer to [Uploading Relevant Assets](#uploading-relevant-assets)

Otherwise, you will see the window below



Origin

The Origin is the positional and orientation coordinates values of the **end effector's base link with respect to the Robot's end effector link**. Unchecking the box defaults the XYZ and RPY coordinates to 0

End Effector Link

This is the link of the end effector (Usually the base link of the end effector) that will be connected to the Robot's End Effector Link (Usually the tip of the robot).

End Effector Type

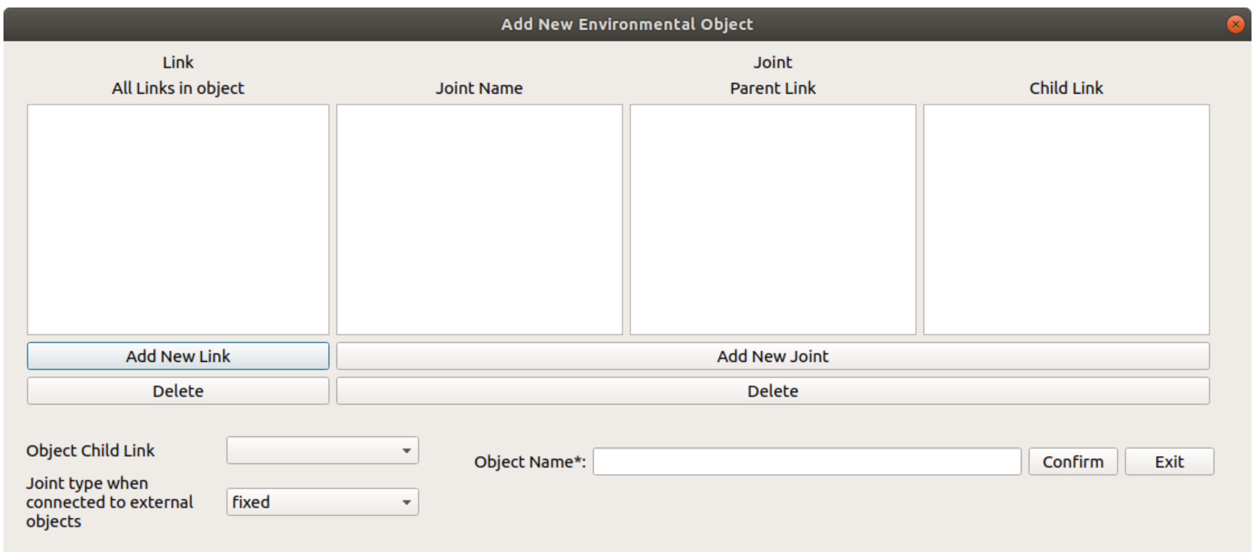
Currently, the end effector types supported will be only for **2 Finger Grippers** and **Single Suction Cup**. Thus the attribute options cannot be selected at this moment.

6.2.3 Adding Objects into scene

Create an object

Contents

An environment object can be used to create a visual representation of an object in a real robotic workcell, or additionally to act as a collision object to be taken into account during path planning



An object is required to have minimally one link that will be used to connect to the external world. This link will be selected under the *Object Child Link* Field. The type of joint you would want to connect to the world is also specifed in the field below

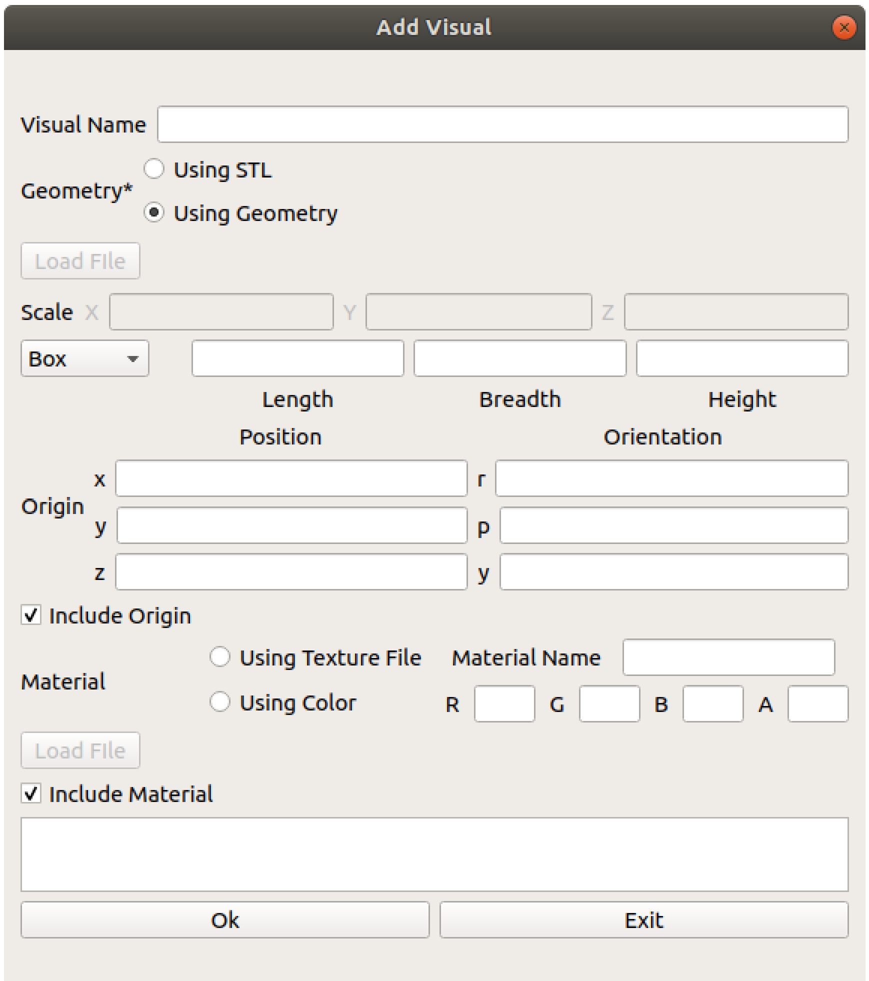
Adding Links

To find out more about each component, Check out this link that describes the various aspects of a Link

For an object, there should be at least one link that connects this object to the external world

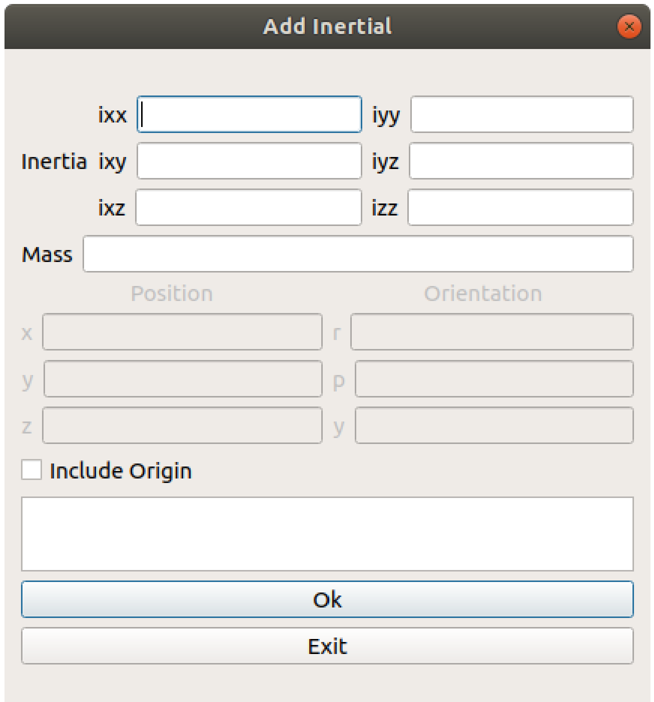
Adding Visual Component

The visual properties of the link. This element specifies the shape of the object (box, cylinder, etc.) for visualization purposes.



Adding Inertial Component

This window allows you to add the inertial properties of the link. This aspect is __optional__ and will default to zero mass and zero inertia if it is not specified.



Inertia

The 3x3 rotational inertia matrix, represented in the inertia frame. Because the rotational inertia matrix is symmetric, only 6 above-diagonal elements of this matrix are specified here, using the attributes ixx, ixy, ixz, iyy, iyz, izz.

Origin

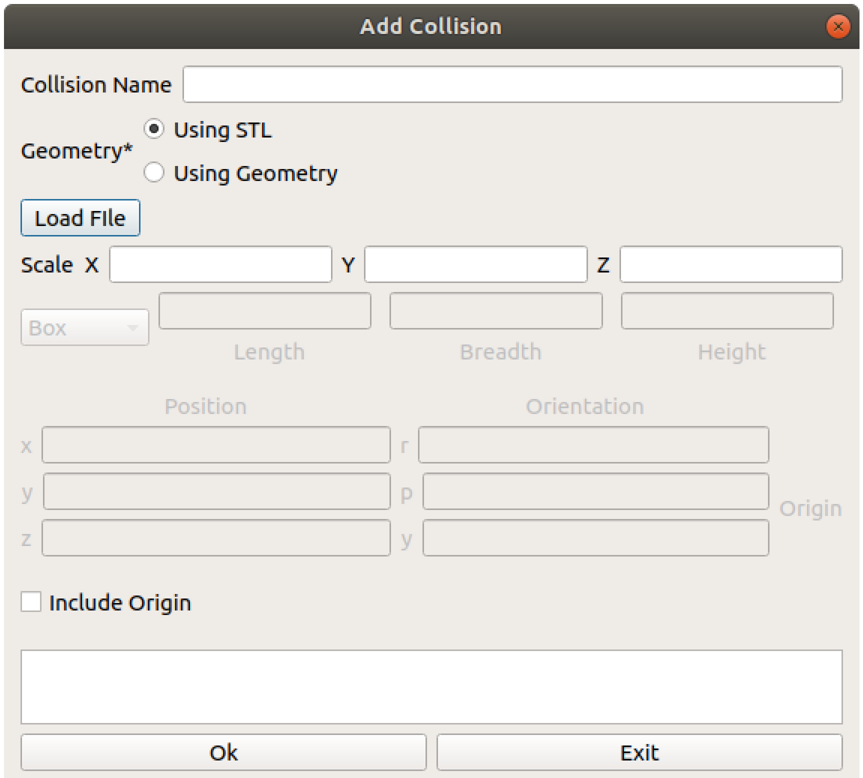
Refer to this section

Mass

Mass of the Link

Adding Collision Component

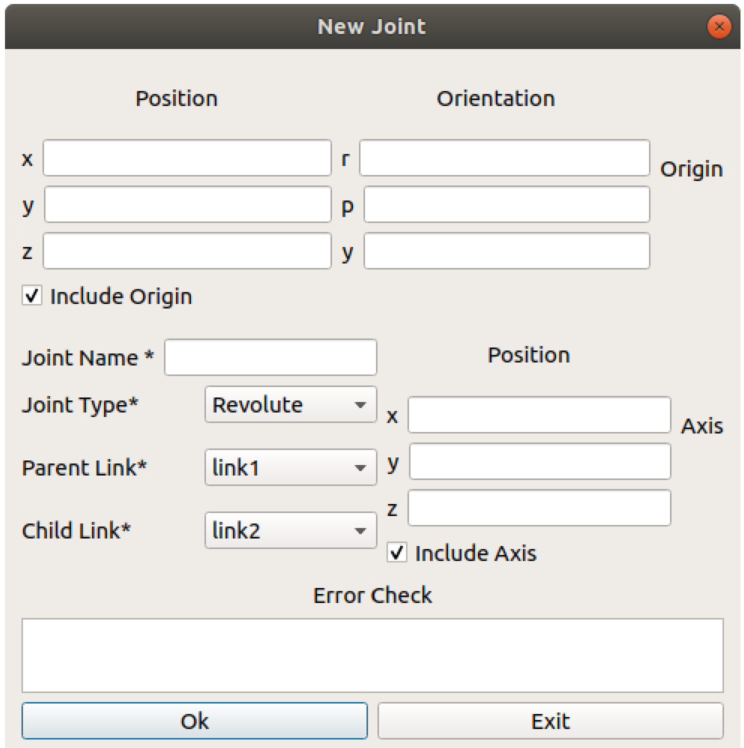
This allows you to describe the collision properties of the link. **To reduce computation time, simpler collision models can be used to describe the object compared to the visual components**



Adding Joints

To find out more about each component, Check out this link that described the various aspects of a Joint

Note that as of this current implementation, only simple joint attributes are included. Other attributes like calibration, dynamics, limits, mimic, safety_controller, will be added in future iterations



Inheritance

When creating a joint for two links in an object, note that it is not possible for a link to be a parent of another link that is higher on the inheritance hierarchy

For example,

```
Link A is a parent of Link B who is a parent of Link C
A > B > C

Link B is also a parent of Link D
B > D

By inheritance rules, Link D cannot be the parent of Link A (Because Link A is the␣
↪parent of Link B)
```

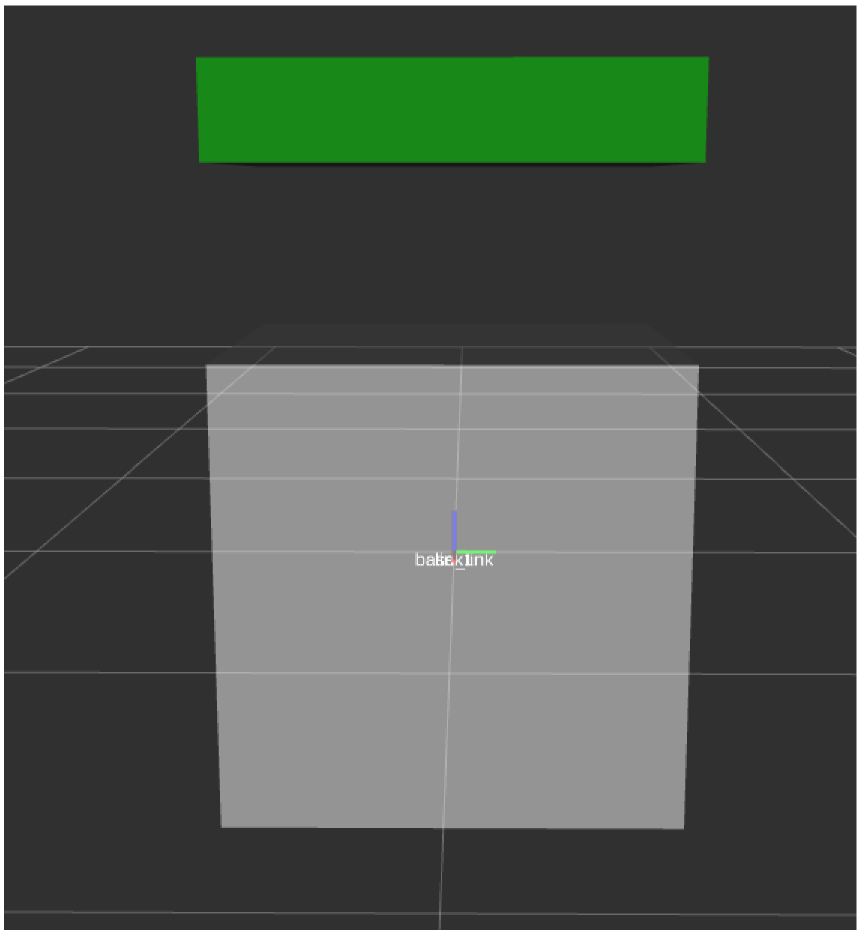
Axis

Represents the joint axis specified in the joint frame. Note that this field is disabled for **fixed and floating joints**.

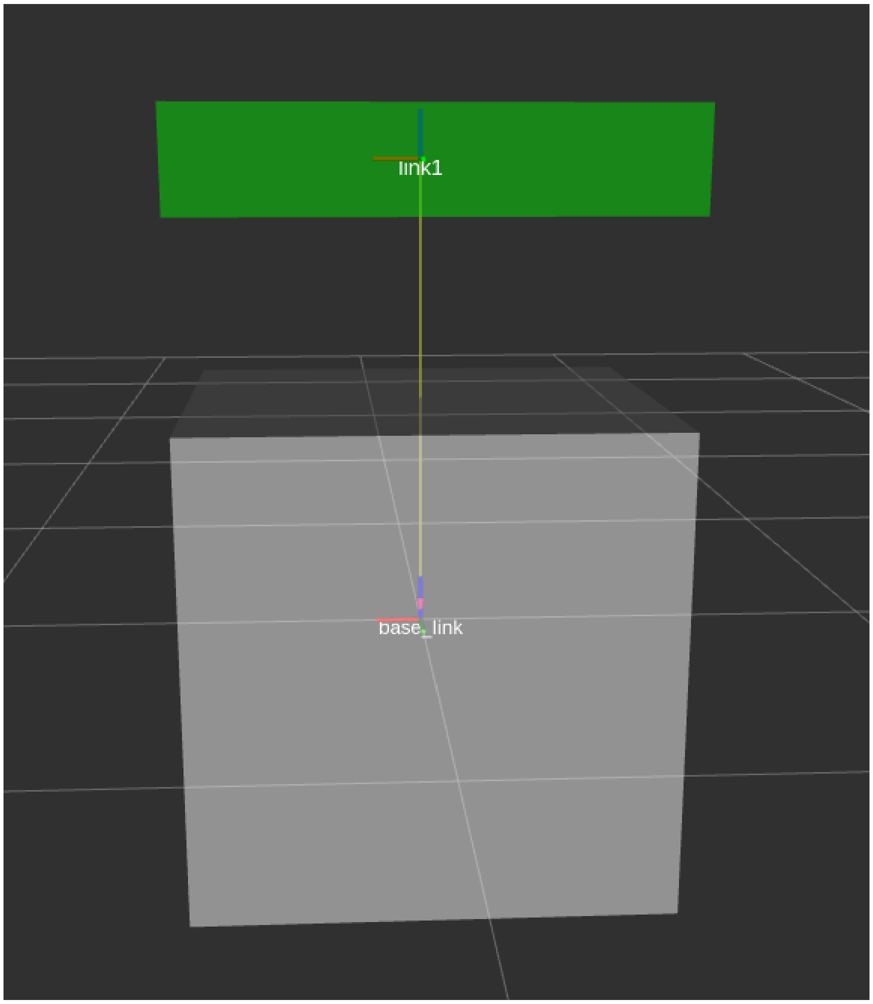
Origin Explanations

Note that there are many different origin sources for the Visual, Collision, Inertia and Joint aspects of the object. For each aspect of the link, all three (visual, collision and inertia) origins will be taken with respect to the **reference frame of the link**.

To find out where this reference frame is, we need to look at the **joint origin** . The following example shows, generally, how the joint origin relates to the link origin.



This configuration shows a `link` connected to the `base_link` with a joint origin of `0,0,0` and the visual mesh of `link` has an origin of `0,0,1`. As you can see, `link`'s tf is at the `0,0,0` of `base_link`, while the **visual** component of `link` is at `0,0,1` from the **tf** of `link`



In this configuration however, shows a joint origin of `0,0,1` and the visual mesh of `link` has an origin of `0,0,0`. As you can see, `link`'s **tf** is at the `0,0,1` of `base_link`, while the **visual** component of `link` is at `0,0,0` from the **tf** of `link`

6.2.4 Complete Scene

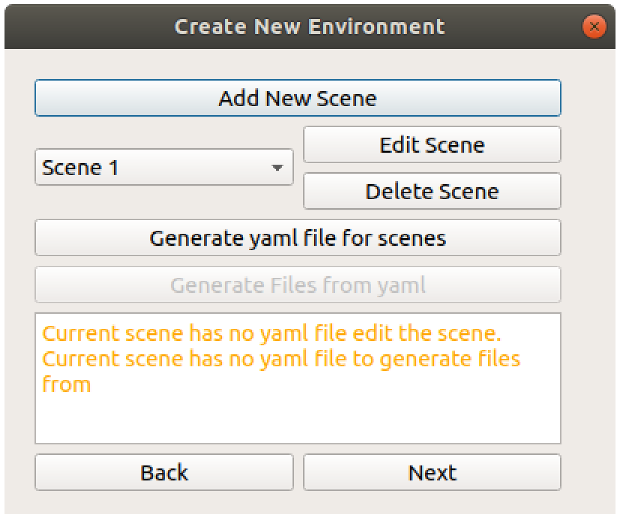
Before you exit the Scene, ensure that a scene name is entered then click the OK button. You should be redirected back to the scene select window

Next step: *Generate Files and Folders*

6.3 Generate Files and Folders

The next step after creating a scene is to generate the relevant files and folders required to create the simulations

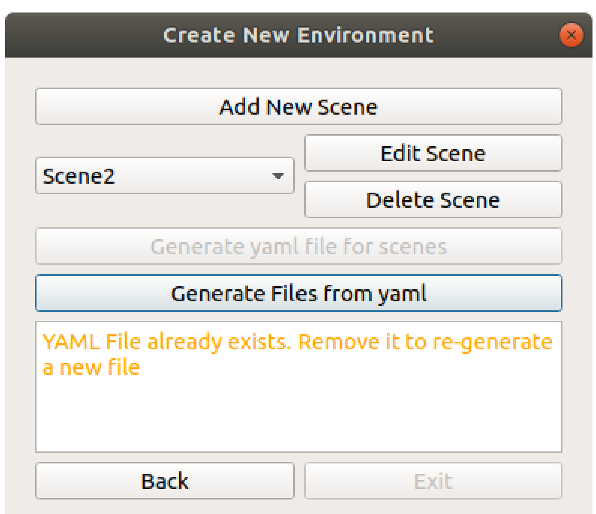
6.3.1 Generating yaml files



To generate this scene, click on the "Generate yaml for scene" button. An *environment.yaml* yaml file will be created in the src/scenes/<scene_name> folder


```yaml
robot:
  name: ur5
  brand: universal_robot
  filepath: ""
  base_link: base_link
  ee_link: ee_link
  links:
    - base_link
    - shoulder_link
    - upper_arm_link
    - forearm_link
    - wrist_1_link
    - wrist_2_link
    - wrist_3_link
    - ee_link
    - base
    - tool0
end_effector:
  name: robotiq_85
  brand: robotiq_85
  filepath: /home/rosi5/ros2_ws/src/assets/end_effectors/robotiq_85/robotiq_85_description
  base_link: gripper_base_link
  robot_link: ee_link
  ee_type: finger
  attributes:
    fingers: 2
  links:
    - gripper_base_link
    - gripper_finger1_knuckle_link
    - gripper_finger2_knuckle_link
    - gripper_finger1_finger_link
    - gripper_finger2_finger_link
    - gripper_finger1_inner_knuckle_link
    - gripper_finger2_inner_knuckle_link
    - gripper_finger1_finger_tip_link
    - gripper_finger2_finger_tip_link
```

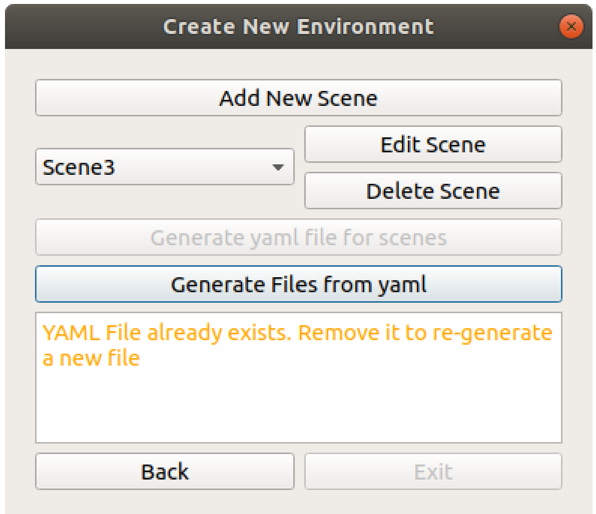
This file is an easy to read textual representation of the scene. This file is important and required to properly generate the workcell. This is also important for reloading a scene into the GUI for editing. It is possible to do these changes via the YAML file directly rather than the GUI. **However, do note that errors may arise if you do not follow the proper YAML format.**



Once succesful, the Generate Files from yaml will be available

6.3.2 Other Files (General)

Click the Generate Files from yaml button to generate the actual required components to build the simulation



The following components will be generated through this button:

Environment Object Packages

The environment objects that were created in the GUI will now have its URDF Xacros and folders generated in the `assets/environment_objects/<object_name>_description` folder. This folder will be referenced in the environment urdf

environment.urdf.xacro

This file is located in the scenes/<scene_name>/urdf/ folder. This is the main urdf file that combines all the different elements of the workcell and will be the main file that launch files will reference to when launching files.

arm_hand.srdf.xacro

This file is located in the `scenes/<scene_name>/urdf/` folder. This is the srdf file that combines the robot arm and end effector and contains information such as ignoring link collisions , and will be referenced in launch files.

6.3.3 Other Files (ROS1)

<will be edited later>

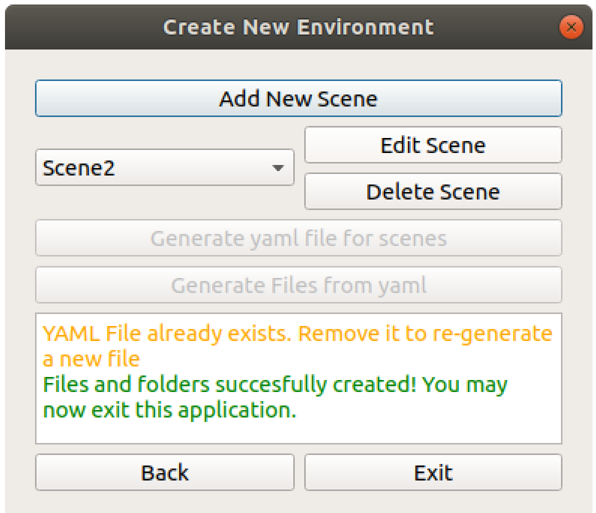
1. The launch file for simulation
2. The move_group.launch file
3. The planning_context.launch

6.3.4 Other Files (ROS2)

demo.launch.py

This file is located in the scenes/<scene_name>/launch/ folder. This file serves as a demo launch file that launches an example RVIZ simulation of the workspace. Note that this launch file does not incorporate Moveit2 Components. To find out how to do so, check out grasp_execution_demo

Once done, you will see the following screen. You can then exit the GUI



Next Steps: *Run workcell demo*

6.4 Run workcell demo

To run the demo simulation for the scene you just generated, do the following:

6.4.1 ROS1

```
$ source /opt/ros/melodic/setup.bash

$ catkin build

$ source devel/setup.bash

$ roslaunch <scene_name> demo.launch
```

6.4.2 ROS2

In a new terminal, navigate to your workcell_ws

```
$ source /opt/ros/foxy/setup.bash

$ colcon build

$ source install/setup.bash

$ ros2 launch <scene_name> demo.launch.py
```

Rviz will then be launched and you should see your scene displayed. To integrate this scene with Moveit2, check out grasp_execution_demo

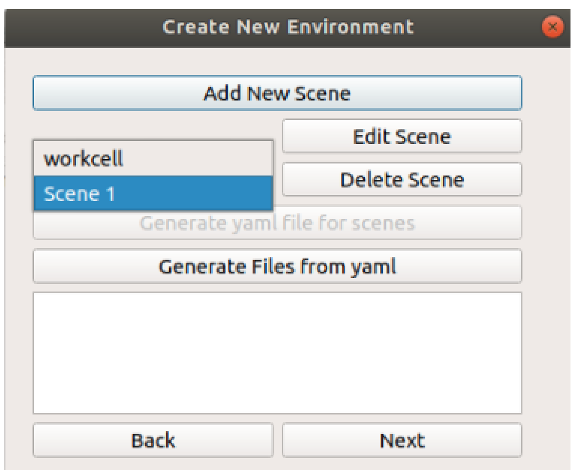
6.5 Editing Existing Scene

If you want to edit the existing scene, there are multiple ways you can do so. The first way is to directly edit the `environment.yaml` file for minor changes. **Do note that you need to follow the YAML format for editing files, if not errors will be thrown.**

6.5.1 Loading YAML file into GUI

To load a yaml file into the GUI, you need to ensure that the **scene folder** is located in the workcell_ws/src/scenes/<scene_name>/ folder. You also need to make sure you have the **environment.yaml** file in the workcell_ws/src/scenes/<scene_name>/ folder as well, if not the GUi will not be able to load the files to edit.

If the scene can be found, it will be available to be clicked to be edited.



Ensure that after you edit the scene, follow the rest of the steps in ref:*Generate Files and Folders* to re-generate the required files.

6.6 Conventions

6.6.1 Naming Conventions

Following a standardized naming convention is highly recommended to avoid any issues with generating the workspace.

Description folders

Any folder that provides a visual representation of each object in scene should be named <name> _description

```
The current exception to this rule is the description folder for universal robots, which␣
↪is currently stored as a folder named  ur_description that encapsulates all the␣
↪current robot models
```

URDF folders

If the folder contains URDF files for description, it should be in a **xacro** format stored in the `urdf` folder, and named:

Robot <robot_model>.urdf.xacro

End effector <end_effector_model> _gripper.urdf.xacro

Environment objects <object_name>.urdf.xacro

moveit_config folder

All end effectors and robots should come with a moveit_config folder named <name> _moveit_config and should be located in the same directory as your robot/end effector description folders.

This folder should be generated using the Moveit Setup Wizard. However, the package generated is currently in ROS1, hence you must make sure that the package is converted into a ROS2 package if your workcell is run in ROS2

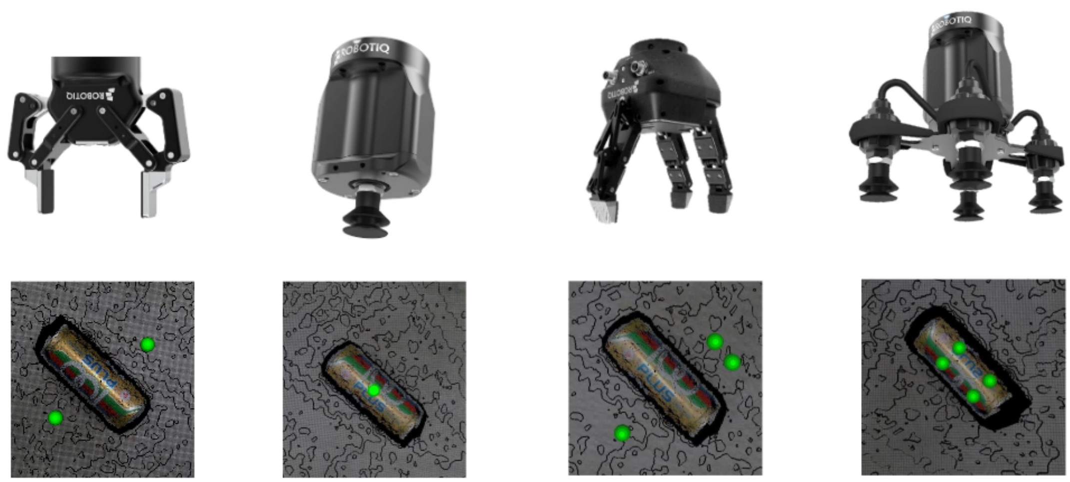
CHAPTER

SEVEN

GRASP PLANNER

7.1 Overview

The Easy Manipulation Deployment Grasp Planner is an Algorithmic Based Point Cloud Grasp Planner that provides a 4 DOF Grasp Pose for both Multifinger End Effectors and Suction Array End Effectors.



7.1.1 Benefits of EMD Grasp Planner

The Grasp Planner aims to **eliminate** the following issues that users would face when deploying Machine Learning based Grasp Planners:

1. Long training times and tedious Dataset acquisition and labelling

Current datasets available such as the Cornell Grasping Dataset and Jacquard Grasping Dataset generally account for two finger grippers and is training on general objects. For custom use cases, datasets need to be generated and hand labelled which requires huge amount of time and manual labour. Semantic description of multifinger grippers and suction arrays may be hard to determine as well.

The Grasp Planner presented in this ROS2 package requires zero datasets and training, and supports multifingered parallel grippers as well as suction cup arrays.

2. Lack of On-The-Fly End Effector Switching

In high mix, low volume pick-and-place scenarios, different end effectors may be needed for different types of objects. Changing of end effectors would then mean that the user would need to collect a whole new dataset, relabel and retrain the dataset and models before use.

The Grasp Planner presented in this ROS2 package allows for on-the-fly end effector switching through a simple configuration file that is highly customizable and understandable.

7.2 Before running the Grasp Planner

Recommended information to read before running the Grasp Planner

7.2.1 Grasp Planning Methodology

Grasp planning methodologies

The current grasp planner supports two main types of end effectors:

Multifinger Linear End Effectors

Defined by parallel actuation of fingers towards the center of the end effectors

Grasp Planner Methodology (Finger)

WIP, Come back soon!

Suction Cup Array End Effectors

Defined by suction cups arranged in a 2D plane

Grasp Planner Methodology (Suction)

WIP, Come back soon!

7.2.2 Grasp Planner Configuration File

The grasp planner aims to be highly customizable, and this customization is done using the configuration file, that is typically stored in the config folder of your package. The YAML format is used for this file for better understanding and readability.

Due to the huge number of parameters that can be tweaked, the parameters can be divided into the following subgroups, where explanation of each parameter will be provided

Grasp Planner General Parameters

The parameters described here are the general configuration components for both finger and suction end effectors. Most of the parameters here are used for either point cloud processing or ROS2 component definitions.

```
grasp_planning_node:
  ros__parameters:
    grasp_output_service: "grasp_requests"
    easy_perception_deployment:
      epd_enabled: false
      tracking_enabled: false
      epd_topic: "/processor/epd_localize_output"
    camera_parameters:
      point_cloud_topic: "/camera/pointcloud"
      camera_frame: "camera_color_optical_frame"
    point_cloud_params:
      passthrough_filter_limits_x: [-0.50, 0.50]
      passthrough_filter_limits_y: [-0.15, 0.40]
      passthrough_filter_limits_z: [0.01, 0.70]
      segmentation_max_iterations: 50
      segmentation_distance_threshold: 0.01
      cluster_tolerance: 0.01
      min_cluster_size: 750
      cloud_normal_radius: 0.03
      fcl_voxel_size: 0.02
    end_effectors:
      end_effector_names: [finger_gripper_1, suction_gripper_1]
      finger_gripper_1:
        ....
      suction_gripper_1:
        .....
    visualization_params:
      point_cloud_visualization: true
```

Parameter Descriptions

grasp_output_service

Description	ROS2 service name for the grasp execution component
Type	string

Details of the GraspRequest Service can be found here: *Grasp Planner Output Message Types*

It is recommended to use the EMD Grasp Execution component. If you do so, keep set `grasp_output_service` as `"grasp_requests"`

easy_perception_deployment

Grasp Planner General Parameters (EPD)

The parameters described here are the configuration components related to the easy_perception_deployment ROS2 Package

```
grasp_planning_node:
  ros__parameters:
    .....
    easy_perception_deployment:
      epd_enabled: false
      tracking_enabled: false
      epd_topic: "/processor/epd_localize_output"
```

easy_perception_deployment.epd_enabled

Description	Enables the use of the EPD workflow
Type	bool

if `true`, EPD Workflow is triggered

if `false`, Direct Camera Workflow is triggered

Details of the different workflows can be found here: *Grasp Planner Input Message Types*

easy_perception_deployment.tracking_enabled

Note: This parameter will only be used if `epd_enabled` is set to `true`

Description	Enables the use of EPD Precision Level 3 Object Tracking
Type	bool

if `true`, EPD Precision Level 3, Object Tracking will be taken as input.

if `false`, EPD Precision Level 2, Object Localization will be taken as input.

To understand more about the different precision levels, visit the easy_perception_deployment documentation

To find out more about the Precision Level ROS2 message differences: *Grasp Planner Input Message Types*

easy_perception_deployment.epd_topic

Note: This parameter will only be used if `epd_enabled` is set to `true`

Description	Topic name of output from the easy_perception_deployment package
Type	string

If your `tracking_enabled` was set to `true` , The default value of `epd_topic` should be `"/processor/epd_tracking_output"`

If your `tracking_enabled` was set to `false` , The default value of `epd_topic` should be `"/processor/epd_localize_output"`

camera_parameters

Grasp Planner General Parameters (Camera)

Parameters that define the camera parameters. May vary depending on the type of camera used.

```
grasp_planning_node:
  ros__parameters:
    .....
    camera_parameters:
      point_cloud_topic: "/camera/pointcloud"
      camera_frame: "camera_color_optical_frame"
```

camera_parameters.point_cloud_topic

```
point_cloud_topic: "/camera/pointcloud"
```

Description	Topic published by the camera using the PointCloud2 message type
Type	string

camera_parameters.camera_frame

```
camera_frame: "camera_color_optical_frame"
```

Description	Tf reference frame which the point cloud is referenced from.
Type	string

Note: The `camera_frame` value may be different depending on the definition of the URDF. In order to determine what the frame is:

1. Run the ROS2 package for your camera

2. Use `ros2 topic echo` command to look at the message published by the camera. Typically if the message type has a sub-message of Header type, refer to the `frame_id` portion.

point_cloud_params

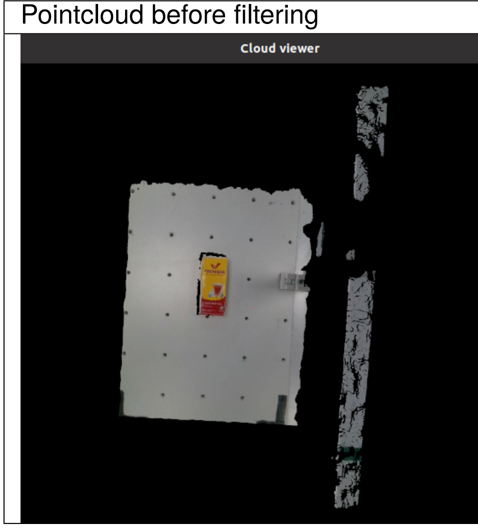
Grasp Planner General Parameters (Point Cloud)

Parameters that define the Point Cloud parameters used for point cloud processing

```
grasp_planning_node:
  ros__parameters:
    .....
    point_cloud_params:
      passthrough_filter_limits_x: [-0.50, 0.50]
      passthrough_filter_limits_y: [-0.15, 0.40]
      passthrough_filter_limits_z: [0.01, 0.70]
      segmentation_max_iterations: 50
      segmentation_distance_threshold: 0.01
      cluster_tolerance: 0.01
      min_cluster_size: 750
      cloud_normal_radius: 0.03
      fcl_voxel_size: 0.02
      octomap_resolution: 0.01
```

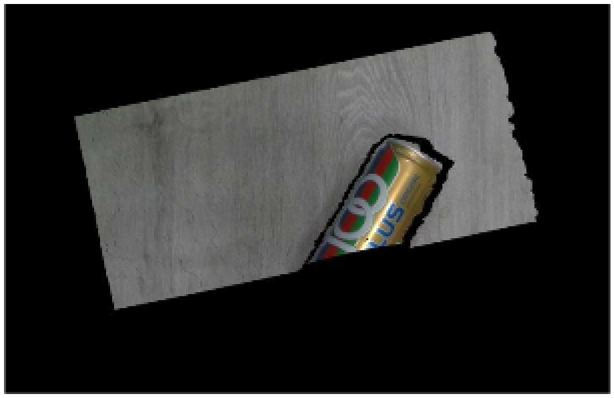
Passthrough Filtering Parameters

In order to **reduce point cloud processing and iteration times**, the Point Cloud Library's implementation of the passthrough filter function is used to crop out useful parts of the pointcloud in question. This is done through setting filter limits in the X, Y and Z axes which act as the range within which points in the point cloud will be kept.

Pointcloud before filtering	Pointcloud after filtering
	

As shown above, unnecessary information on the right side of the pointcloud was removed. Limits can be set even tighter in order to crop out more of the work surface.

Warning: Ensure that an appropriate passthrough filter limit is chosen for the X, Y and Z axes. You might end up cropping out useful information if the limits are too tight.



More information can be found in the Passthrough Filter Tutorials

point_cloud_params.passthrough_filter_limits_x

```
passthrough_filter_limits_x: [-0.50, 0.50]
```

Description	Lower and Upper Passthrough Filter limits in the X-Axis
Type	double array

Warning: The tighter the limits, the faster the point cloud will be processed, but you run the risk of croppping out important information

The wider the limits, the slower the point cloud, but you will not crop out important information.

point_cloud_params.passthrough_filter_limits_y

```
passthrough_filter_limits_y: [-0.15, 0.40]
```

Description	Lower and Upper Passthrough Filter limits in the Y-Axis
Type	double array

Warning: The tighter the limits, the faster the point cloud will be processed, but you run the risk of croppping out important information

The wider the limits, the slower the point cloud, but you will not crop out important information.

point_cloud_params.passthrough_filter_limits_z

```
passthrough_filter_limits_z: [0.01, 0.70]
```

Description	Lower and Upper Passthrough Filter limits in the Z-Axis
Type	double array

> **Warning:** The tighter the limits, the faster the point cloud will be processed, but you run the risk of croppping out important information.
>
> The wider the limits, the slower the point cloud, but you will not crop out important information.

Plane Segmentation Parameters

For Object extraction, the Grasp Planner will first remove the set of points in the point cloud representing the surface on which the objects are placed. This is done using the Point Cloud Library's Plane Segementation functions, specifically SAmple Consensus (SAC) methods

point_cloud_params.segmentation_max_iterations

```
segmentation_max_iterations: 50
```

Description	Set the Maximum number of iterations for SAC methods
Type	Int

> **Warning:** The higher the number, the better the result, but longer times will be taken The lower the number, the less accurate the result, but shorter time will be taken

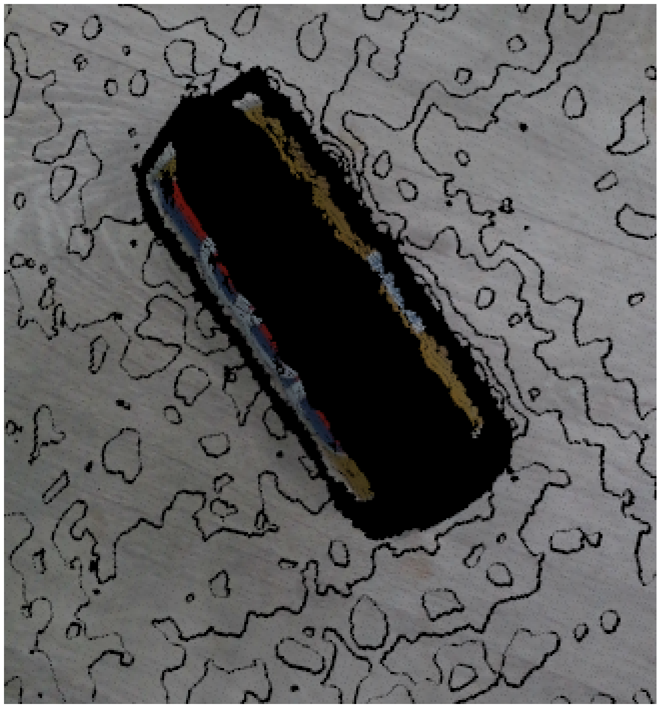
point_cloud_params.segmentation_distance_threshold

```
segmentation_distance_threshold: 0.01
```

Description	Determines how close a point must be to the object in order to be considered an inlier
Type	Double

> **Warning:** A higher number ensures that more points will be clustered with the object cluster, but you run the risk of including points on the table as part of the object

Warning: A lower number ensures that only objects really close to the cluster is included inside, but you may run the risk of missing out some points on the surface



Object Segmentation Parameters

After the plane has been removed from the point cloud input, the assumption is made that the rest of the pointcloud represents the pick objects (Unless using the EPD workflow, more on that here: *Grasp Planner Input Message Types*). The remaining Point Cloud will then be split into clusters using Euclidean Cluster Extraction provided by the Point Cloud Library, each cluster representing a grasp object

point_cloud_params.cluster_tolerance

```
cluster_tolerance: 0.01
```

Description	Get the spatial cluster tolerance as a measure in the L2 Euclidean space.
Type	Double

point_cloud_params.min_cluster_size

```
min_cluster_size: 750
```

Description	For Euclidean cluster extraction. Determine the minimum number of points to be considered a cluster.
Type	Int

> **Warning:** If it is set too high, small objects may be clustered together to satisfy the minimum cluster size.
>
> If set too small,certain objects might be split into multiple clusters if the point cloud is not dense enough

Normals Estimation Parameters

For both finger and suction gripper grasp planning, we take into account the surface of the object as well, which involves understanding the curvature of the surfaces, which requires estmation of point normals for each point on the surface of the object, which we will use the Point Cloud Library to do so.

More information on the normal estimation for PCL can be found here

point_cloud_params.cloud_normal_radius

```
cloud_normal_radius: 0.03
```

Description	The radius of points around each point to determine the normal estimation.
Type	Double

> **Warning:** If the radius value is too big, you run the risk of including adjacent surfaces around that point, which distorts the estimated point features

FCL Collision Object Parameters

In order to account for collision between end effector and the grasp area/grasp object, we use the Flexible Collision Library (FCL) as a method to generate collision objects for both grasp samples and grasp area.

This is done by first converting the point cloud to an octomap, then to an FCL collision object. In order to speed up conversion time, we first downsample the pointcloud before conversion

point_cloud_params.fcl_voxel_size

```
fcl_voxel_size: 0.02
```

Description	Size of resulting voxels after downsampling of pointclouds.
Type	Double

> **Warning:** If it is set too large, collision object conversion will be faster, but certain features may be lost during downsampling.
>
> If set too small, collision object conversion will be longer, but the collision object shapes will more accurately represent the original point cloud

point_cloud_params.octomap_resolution

```
octomap_resolution: 0.01
```

Description	Resolution of octomap (Used for conversion to collision object)
Type	Double

end_effectors

Grasp Planner General Parameters (End Effectors)

Parameters that define the end effectors involved in grasp planning. Multiple end effectors can be declared in one config file

```
grasp_planning_node:
  ros__parameters:
    ....
    end_effectors:
      end_effector_names: [finger_gripper_1, suction_gripper_1]
      finger_gripper_1:
        ....
      suction_gripper_1:
        .....
```

end_effectors.end_effector_names

```
end_effector_names: [finger_gripper_1, suction_gripper_1]
```

Description	The array of names for end effectors used for grasp planning
Type	String array

Note: Make sure the names here matches the name of the end effector in corresponding end effector parameters.

visualization_params

Grasp Planner General Parameters (Visualization)

The parameters described here are the configuration components related to grasp visualization

```
grasp_planning_node:
  ros__parameters:
    .....
    visualization_params:
      point_cloud_visualization: true
```

visualization_params.point_cloud_visualization

```
point_cloud_visualization: true
```

Description	Provides 3D visualization of the grasp samples using PCL Viusalizer
Type	Bool

Warning: If you set this parameter to `true`, the PCL Visualizer will be spun, and your grasp plans will be displayed. this is a blocking process, so your grasp plans will not be published until you exit the Visualizer. **Thus it is recommended to leave this as `false`**

To move to the next grasp sample, press `q` on your keyboard with the Visualizer window selected to move to the next grasp sample.

Grasp Planner Finger Parameters

```
grasp_planning_node:
  ros__parameters:
    ...
    end_effectors:
      end_effector_names: [finger_gripper_1, suction_gripper_1]
      finger_gripper_1:
        type: finger
        num_fingers_side_1: 1
        num_fingers_side_2: 1
        distance_between_fingers_1: 0.0
        distance_between_fingers_2: 0.0
        finger_thickness: 0.02
        gripper_stroke: 0.105
        gripper_coordinate_system:
          grasp_stroke_direction: "x"
          grasp_stroke_normal_direction: "y"
          grasp_approach_direction: "z"
        grasp_planning_params:
          grasp_plane_dist_limit: 0.007
          voxel_size: 0.01
          grasp_rank_weight_1: 1.5
          grasp_rank_weight_2: 1.0
          world_x_angle_threshold: 0.5
          world_y_angle_threshold: 0.5
          world_z_angle_threshold: 0.25
```

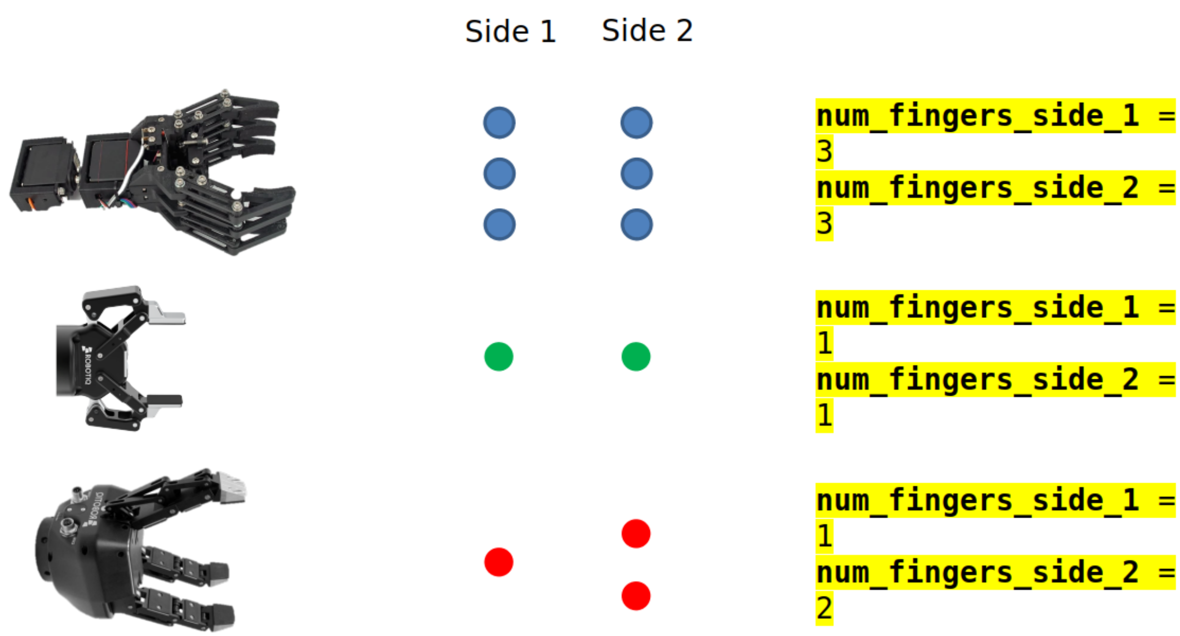
Physical Attributes

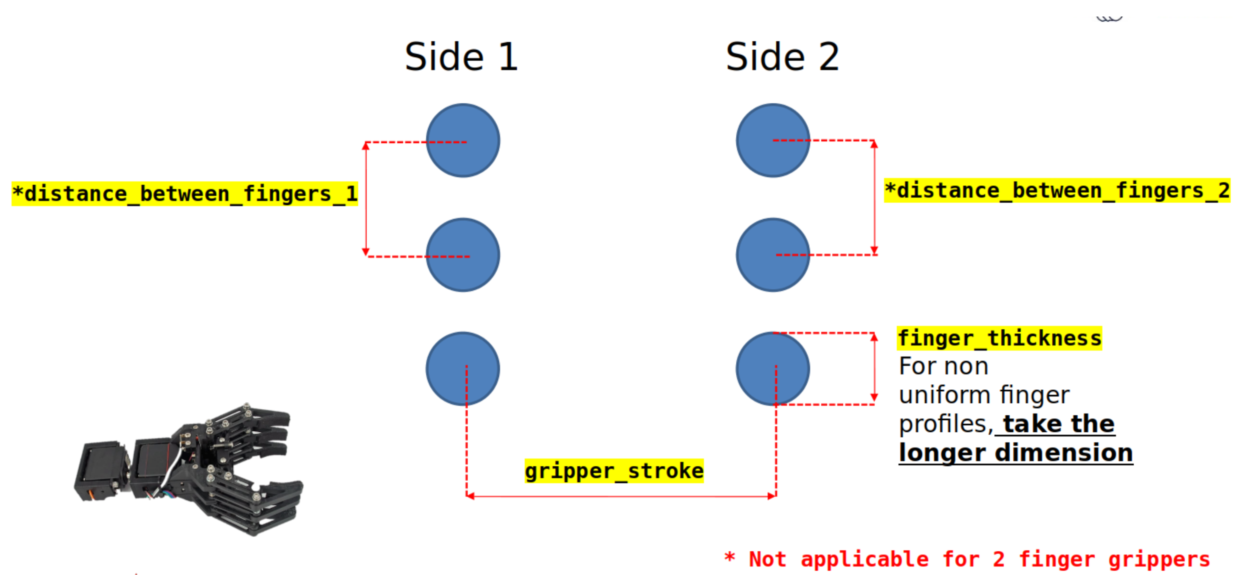
Grasp Planner Finger Parameters (Physical Attributes)

The Parameters in this Section specifically defines the physical attributes of the finger gripper. These parameters provide additional flexibility for grasp planner to support a myriad of finger grippers.

The current grasp planner supports **linear finger grippers**, Where there are two main sides containing the fingers, and grasping motions are paralell to the direction of distribution of fingers

```
finger_gripper_1:
  type: finger
  num_fingers_side_1: 1
  num_fingers_side_2: 1
  distance_between_fingers_1: 0.0
  distance_between_fingers_2: 0.0
  finger_thickness: 0.02
  gripper_stroke: 0.105
```





<finger_gripper_name>.type

```
type: finger
```

Description	Describes gripper type
Type	String

Warning: Do not change this parameter, leave it as `finger`

<finger_gripper_name>.num_fingers_side_1

```
num_cups_length: 1
```

Description	Number of fingers on side 1
Type	Int

Warning: Should be at least 1

<finger_gripper_name>.num_fingers_side_2

```
num_cups_breadth: 1
```

Description	Number of fingers on side 2
Type	Int

Warning: Should be at least 1

<finger_gripper_name>.distance_between_fingers_1

```
distance_between_fingers_1: 0.0
```

Description	Center-to-center finger distance between fingers in side 1
Type	Double

Warning: If `num_fingers_side_1` is 1, set `distance_between_fingers_1` as `0.0`

<finger_gripper_name>.distance_between_fingers_2

```
distance_between_fingers_2: 0.0
```

Description	Center-to-center finger distance between fingers in side 2
Type	Double

Warning: If `num_fingers_side_2` is 1, set `distance_between_fingers_2` as `0.0`

<finger_gripper_name>.finger_thickness

```
finger_thickness: 0.02
```

Description	Maximum dimension of the finger (dimensions along the axis perpedicular to the approach direction)
Type	Double

Note: We represent each finger as a sphere, which only requires one dimension, hence the largest dimension of the finger should be provided

<finger_gripper_name>.gripper_stroke

```
gripper_stroke: 0.105
```

Description	Distance between both sides of the finger gripper
Type	Double

Coordinate System Attributes

Grasp Planner Finger Parameters (Coordinate Systems)

The parameters in this section provides user the flexibility to define the coordinate system definition for their gripper.

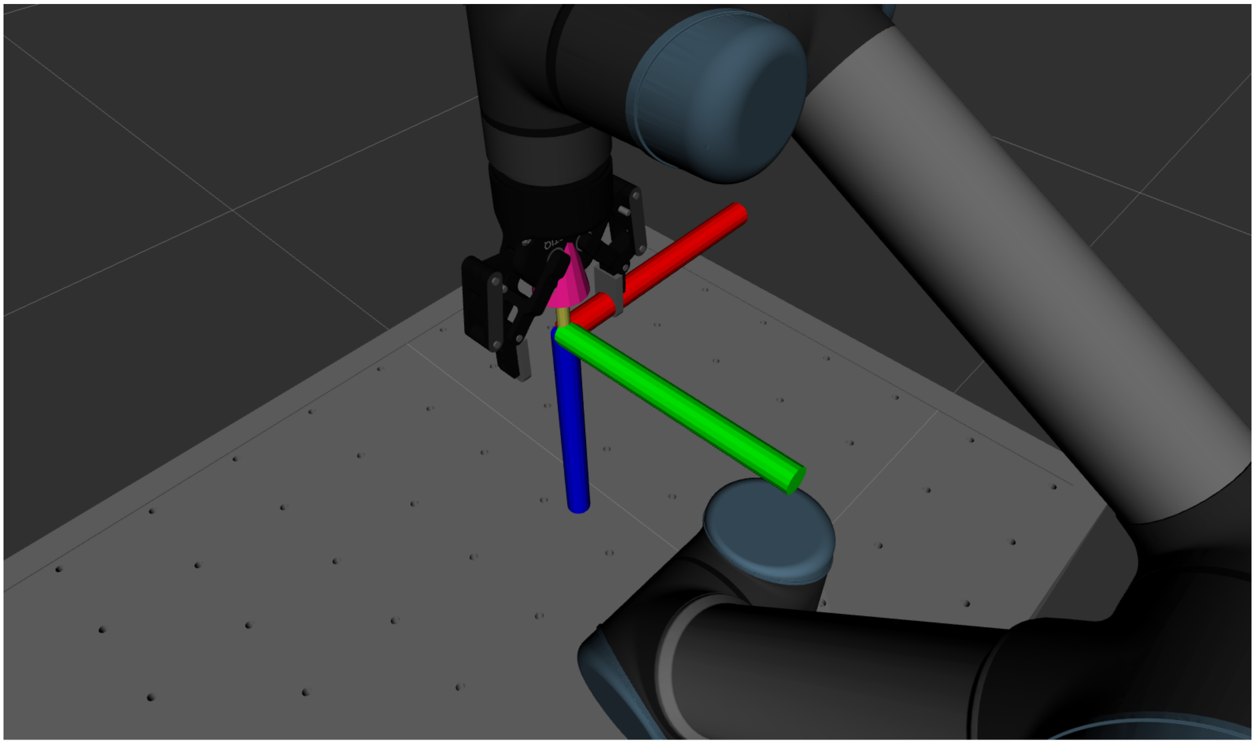
For `grasp_approach_direction` axis, it is defined as the direction along which the end effector will travel to approach the object to grasp it.

For the `grasp_stroke_direction` axis, it is defined as the direction from one side of the finger gripper to another, i.e the direction of movement when the finger gripper closes

For the `grasp_stroke_normal_direction` axis, is defined as the direction perpendicular to both `grasp_stroke_direction` and `grasp_approach_direction`.

For this particular configuration below, assuming the RGB-XYZ convention, the coordinate system is defined as the following:

```
gripper_coordinate_system:
  grasp_stroke_direction: "x"
  grasp_stroke_normal_direction: "y"
  grasp_approach_direction: "z"
```



<finger_gripper_name>.gripper_coordinate_system.grasp_stroke_direction

```
length_direction: "x"
```

Description	Axes defining the grasp stroke direction
Type	String

> **Warning:** Restricted to "x" , "y" or "z"

<finger_gripper_name>.gripper_coordinate_system.grasp_stroke_normal_direction

```
breadth_direction: "y"
```

Description	Axes defining the grasp stroke normal direction
Type	String

> **Warning:** Restricted to "x" , "y" or "z"

<finger_gripper_name>.gripper_coordinate_system.grasp_approach_direction

```
grasp_approach_direction: "z"
```

Description	Axes defining the grasp approach direction
Type	String

Warning: Restricted to "x" , "y" or "z"

Grasp Planning Attributes

Grasp Planner Finger Parameters (Planning)

These parameters directly affect the grasp planning aspects of the finger gripper.

To find out more about how the grasp is being ranked, go to *Grasp Planner Methodology (Finger)*

```
grasp_planning_params:
  grasp_plane_dist_limit: 0.007
  voxel_size: 0.01
  grasp_rank_weight_1: 1.5
  grasp_rank_weight_2: 1.0
  world_x_angle_threshold: 0.5
  world_y_angle_threshold: 0.5
  world_z_angle_threshold: 0.25
```

<finger_gripper_name>.grasp_planning_params.grasp_plane_dist_limit

```
num_sample_along_axis: 3
```

Description	Determine the distance from the grasp plane which to determine the grasp area
Type	Int

Note: The greater the number, the more points included in the grasp area, which increases accuracy, but also increases grasp planning times.

<finger_gripper_name>.grasp_planning_params.voxel_size

```
search_resolution: 0.01
```

Description	Determines the voxel size for downsampling of grasp clusters.
Type	Double

This parameter determines how much downsampling is done after grasp clusters are determined.

Note: The smaller the voxel size, the less downsampling is done, which means more grasp samples can be generated, but it means that grasp planning times will increase

<finger_gripper_name>.grasp_planning_params.grasp_rank_weight_1

```
grasp_rank_weight_1: 1.5
```

Description	Weight for first ranking portion of finger gripper
Type	Double

<finger_gripper_name>.grasp_planning_params.grasp_rank_weight_2

```
grasp_rank_weight_2: 1.0
```

Description	Weight for second ranking portion of finger gripper
Type	Double

<finger_gripper_name>.grasp_planning_params.world_x_angle_threshold

Currently not used

<finger_gripper_name>.grasp_planning_params.world_y_angle_threshold

Currently not used

<finger_gripper_name>.grasp_planning_params.world_z_angle_threshold

Currently not used

Grasp Planner Suction Parameters

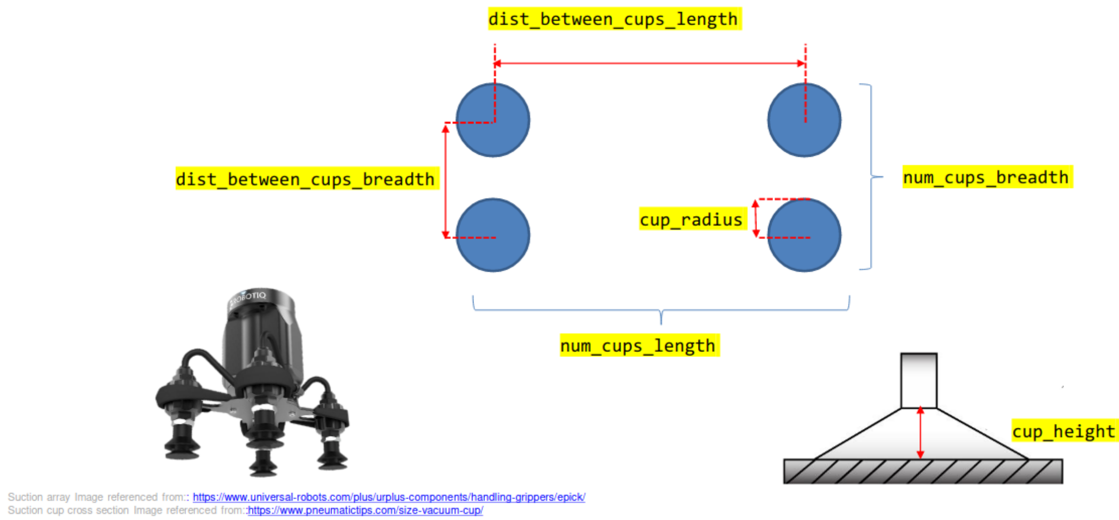
```
grasp_planning_node:
  ros__parameters:
    ...
    end_effectors:
      end_effector_names: [finger_gripper_1, suction_gripper_1]
      suction_gripper_1:
        type: suction
        num_cups_length: 1
        num_cups_breadth: 1
        dist_between_cups_length: 0.06
        dist_between_cups_breadth: 0.06
        cup_radius: 0.005
        cup_height: 0.01
        gripper_coordinate_system:
          length_direction: "x"
          breadth_direction: "y"
          grasp_approach_direction: "z"
        grasp_planning_params:
          num_sample_along_axis: 3
          search_resolution: 0.01
          search_angle_resolution: 4
          weights:
            curvature: 1.0
            grasp_distance_to_center: 1.0
            number_contact_points: 1.0
```

Physical Attributes

Grasp Planner Suction Parameters (Physical Attributes)

The Parameters in this Section specifically defines the physical attributes of the suction gripper. These parameters provide additional flexibility for grasp planner to support a myriad of suction grippers.

```
suction_gripper_1:
  type: suction
  num_cups_length: 1
  num_cups_breadth: 1
  dist_between_cups_length: 0.06
  dist_between_cups_breadth: 0.06
  cup_radius: 0.005
  cup_height: 0.01
```



Note: "Length" and "Breadth" elements can be determined by the user, as long as it remains consistent for the whole file

<suction_gripper_name>.type

```
type: suction
```

Description	Describes gripper type
Type	String

Warning: Do not change this parameter, leave it as `suction`

<suction_gripper_name>.num_cups_length

```
num_cups_length: 1
```

Description	Number of cups in the length direction
Type	Int

Warning: Should be at least 1

<suction_gripper_name>.num_cups_breadth

```
num_cups_breadth: 1
```

Description	Number of cups in the breadth direction
Type	Int

Warning: Should be at least 1

<suction_gripper_name>.dist_between_cups_length

```
dist_between_cups_length
```

Description	Center-to-center distance between in the length direction (m)
Type	Double

<suction_gripper_name>.dist_between_cups_breadth

```
dist_between_cups_breadth
```

Description	Center-to-center distance between in the breadth direction (m)
Type	Double

<suction_gripper_name>.cup_radius

```
cup_radius: 0.005
```

Description	Radius of each suction cup
Type	Double

<suction_gripper_name>.cup_height

```
cup_height: 0.01
```

Description	Height of each suction cup
Type	Double

Coordinate System Attributes

Grasp Planner Suction Parameters (Coordinate Systems)

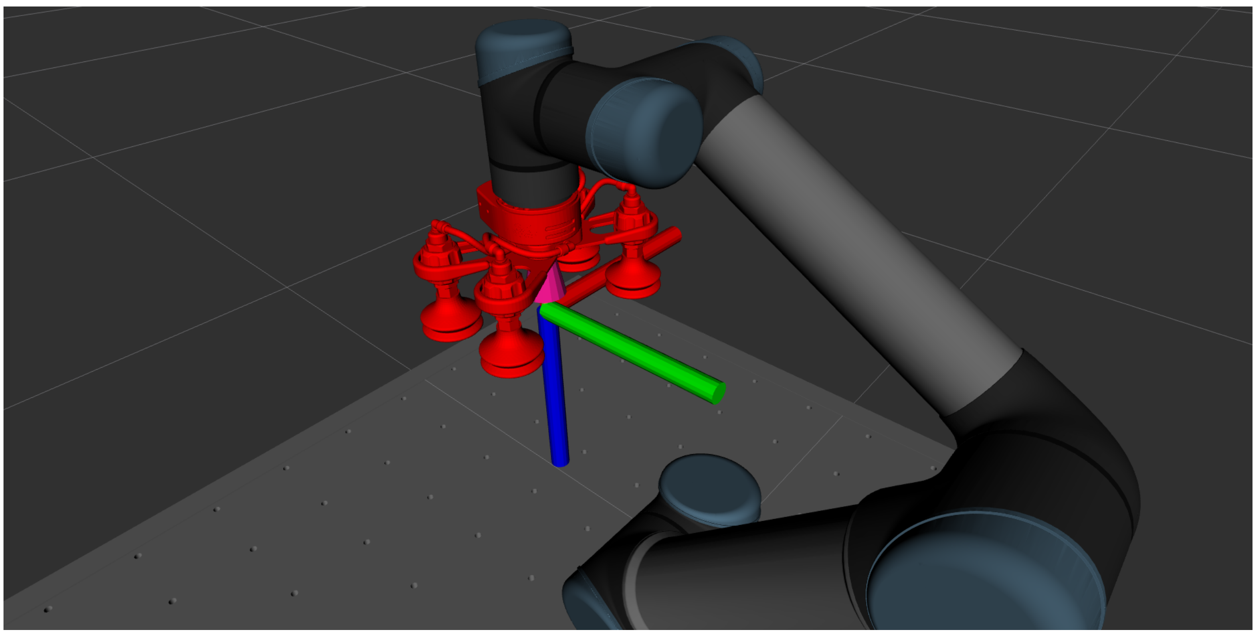
The parameters in this section provides user the flexibility to define the coordinate system definition for their gripper.

For `grasp_approach_direction` axis, it is defined as the direction along which the end effector will travel to approach the object to grasp it.

For the `length_direction` and `breadth_direction`, these are the perpendicular axes along which the suction cups are arranged. Which direction is length and breadth can be arbitrarily defined, but should be consistent with the definition for the parameters in *Grasp Planner Suction Parameters (Physical Attributes)*

For this particular configuration below, assuming the RGB-XYZ convention, the coordinate system is defined as the following:

```
gripper_coordinate_system:
  length_direction: "x"
  breadth_direction: "y"
  grasp_approach_direction: "z"
```



<suction_gripper_name>.gripper_coordinate_system.length_direction

```
length_direction: "x"
```

Description	Axes defining the length direction
Type	String

Warning: Restricted to "x" , "y" or "z"

<suction_gripper_name>.gripper_coordinate_system.breadth_direction

```
breadth_direction: "y"
```

Description	Axes defining the breadth direction
Type	String

Warning: Restricted to "x" , "y" or "z"

<suction_gripper_name>.gripper_coordinate_system.grasp_approach_direction

```
grasp_approach_direction: "z"
```

Description	Axes defining the grasp approach direction
Type	String

Warning: Restricted to "x" , "y" or "z"

Grasp Planning Attributes

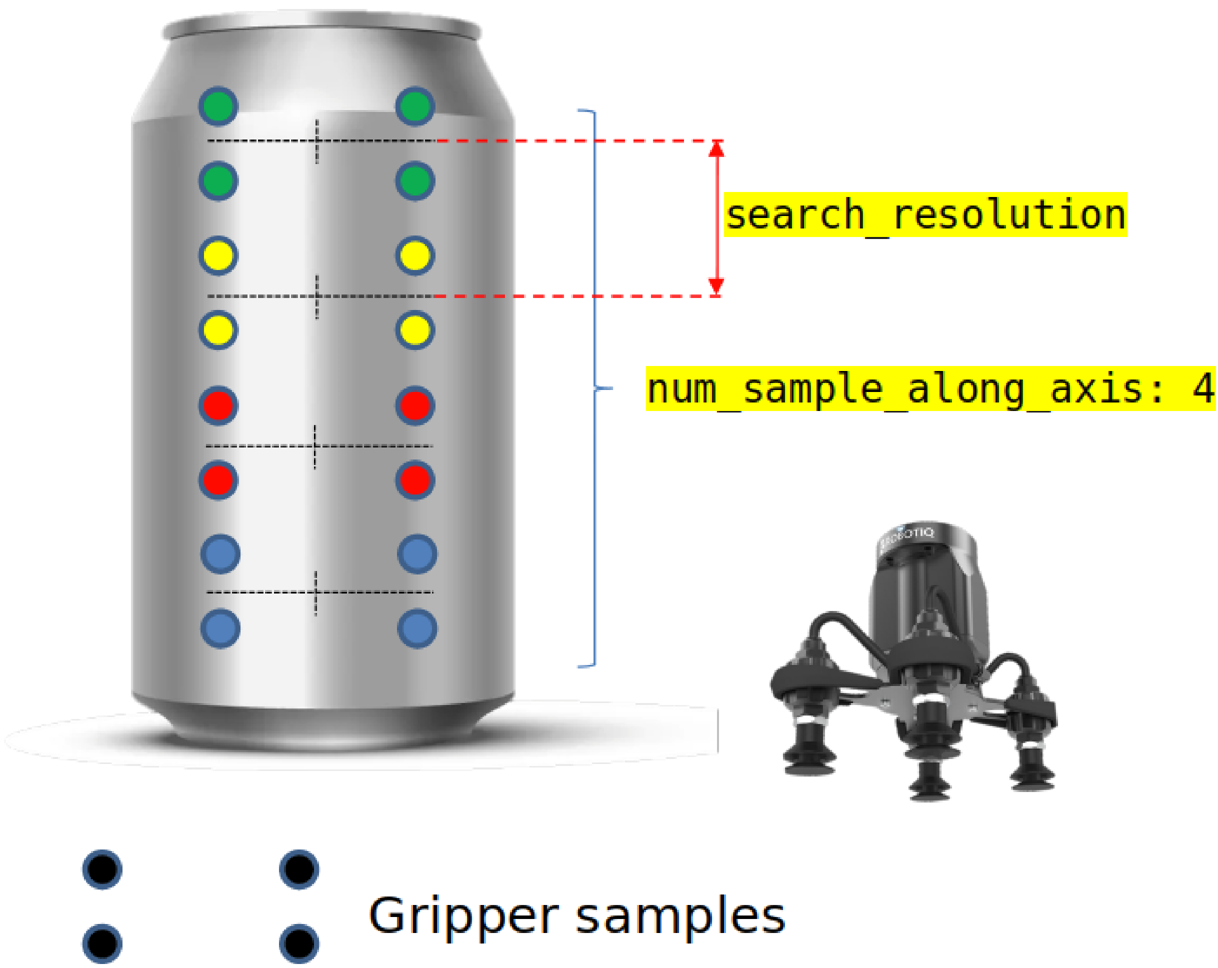
Grasp Planner Suction Parameters (Planning)

These parameters directly affect the grasp planning aspects of the suction gripper

```
grasp_planning_params:
  num_sample_along_axis: 3
  search_resolution: 0.01
  search_angle_resolution: 4
  weights:
    curvature: 1.0
    grasp_distance_to_center: 1.0
    number_contact_points: 1.0
```

Grasp sample Generation

These parameter affects the amount of grasp samples generated for each instance of grasp planning



<suction_gripper_name>.grasp_planning_params.num_sample_along_axis

```
num_sample_along_axis: 3
```

Description	Total number of samples generated along the axis of the object
Type	Int

Note: The greater the number, the more samples along the axis will be generated and tested, but the grasp planning times will increase

<suction_gripper_name>.grasp_planning_params.search_resolution

```
search_resolution: 0.01
```

Description	Provides the distance between each generated sample along the axis of the object
Type	Double

<suction_gripper_name>.grasp_planning_params.search_angle_resolution


```
search_angle_resolution: 4
```

Description	Provides the number of rotated grasp samples within an entire rotation about a particular grasp sample
Type	Int

Note: The greater the number, the more rotated samples generated, but the grasp planning times will increase.

Grasp Planning Weights

Parameters here directly contribute to the ranking of each suction grasp sample. Configure each of the weight based on the user's particular use case and which attribute is more valued.

Note: Ensure that each weight is a positive value less than or equal to 1.

For default values, users can leave all weights at 1.0

To find out more about how the grasp is being ranked, go to *Grasp Planner Methodology (Suction)*

<suction_gripper_name>.grasp_planning_params.weights.curvature

```
curvature: 1.0
```

Description	Weights for the curvature component of the grasp ranking
Type	Double

<suction_gripper_name>.grasp_planning_params.weights.grasp_distance_to_center

```
grasp_distance_to_center: 1.0
```

Description	Weights for the distance to object center component of the grasp ranking
Type	Double

<suction_gripper_name>.grasp_planning_params.weights.number_contact_points

```
number_contact_points: 1.0
```

Description	Weights for the number of contact point component of the grasp ranking
Type	Double

Sample configuration yaml file

If you are unsure of how to begin writing a yaml file, the following is an example of the full configuration of the yaml file.

```yaml
grasp_planning_node:
  ros__parameters:
    grasp_output_topic: "/grasp_tasks"
    easy_perception_deployment:
      epd_enabled: false
      tracking_enabled: false
      epd_topic: "/processor/epd_localize_output"
    camera_parameters:
      point_cloud_topic: "/camera/pointcloud"
      camera_frame: "camera_color_optical_frame"
    point_cloud_params:
      passthrough_filter_limits_x: [-0.50, 0.50]
      passthrough_filter_limits_y: [-0.15, 0.40]
      passthrough_filter_limits_z: [0.01, 0.70]
      segmentation_max_iterations: 50
      segmentation_distance_threshold: 0.01
      cluster_tolerance: 0.01
      min_cluster_size: 750
      cloud_normal_radius: 0.03
      fcl_voxel_size: 0.02
    end_effectors:
      end_effector_names: [finger_gripper_1, suction_gripper_1]
      finger_gripper_1:
        type: finger
        num_fingers_side_1: 1
        num_fingers_side_2: 1
        distance_between_fingers_1: 0.0
        distance_between_fingers_2: 0.0
        finger_thickness: 0.02
        gripper_stroke: 0.105
        gripper_coordinate_system:
          grasp_stroke_direction: "x"
          grasp_stroke_normal_direction: "y"
          grasp_approach_direction: "z"
        grasp_planning_params:
          grasp_plane_dist_limit: 0.007
          voxel_size: 0.01
          grasp_rank_weight_1: 1.5
          grasp_rank_weight_2: 1.0
          world_x_angle_threshold: 0.5
          world_y_angle_threshold: 0.5
          world_z_angle_threshold: 0.25
      suction_gripper_1:
        type: suction
        num_cups_length: 1
        num_cups_breadth: 1
        dist_between_cups_length: 0.06
        dist_between_cups_breadth: 0.06
```

(continues on next page)

(continued from previous page)

```
        cup_radius: 0.005
        cup_height: 0.01
        gripper_coordinate_system:
          length_direction: "x"
          breadth_direction: "y"
          grasp_approach_direction: "z"
        grasp_planning_params:
          num_sample_along_axis: 3
          search_resolution: 0.01
          search_angle_resolution: 4
          weights:
            curvature: 1.0
            grasp_distance_to_center: 1.0
            number_contact_points: 1.0
    visualization_params:
      point_cloud_visualization: true
```

7.3 Running the Grasp Planner

7.3.1 Running the Grasp Planner

To properly run the Grasp Planner, there are 3 main components needed for running the grasp planner

Note: Ensure that for these three terminals, the ROS2 distributions are sourced, and the workspace is built and sourced as well

1. Perception source

   Perception Source can be provided via Direct Camera Input, or via the Easy Perception Deployment ROS2 package.

2. Package Publishing the TF of the camera frame

   The Grasp Planner uses a TF2 Message Filter that waits until the camera frame is published before triggering the planning itself, thus ensure that whatever package you are using is publishing the tf of the camera frame. You can configure the camera frame name directly in the grasp planner configuration file. More can be found here: *Grasp Planner General Parameters (Camera)*

   The EMD Grasp Execution Component provides such a tf publisher feature, thus you can use that as well, by running

```
ros2 launch grasp_execution grasp_execution.launch.py
```

3. EMD Grasp Planner

- Copy any of the files found in `grasp_planner/example/launch`, rename it to `grasp_planner_(end_effector)_launch.py` and replace the `params_(end_effector).yaml` within the launch file with the name of the .yaml file you have created. More can be found here: *Grasp Planner Configuration File*

To run the grasp planner, run the following command

```
ros2 launch grasp_planner grasp_planner_(end_effector)_launch.py
```

The package will then show the following when waiting for the perception topic

```
[pcl_test_node-1] waiting...
```

Note: A blank Cloud Viewer window will pop up, but will only be used if the `point_cloud_visualization` parameter in the config file is true.

Example Grasp Planning Output:

Finger gripper:

```
[demo_node-1] [INFO] [1622278539.845614356] [GraspScene]: Using Direct Camera Input....
[demo_node-1] [INFO] [1622278539.847663649] [GraspScene]: waiting....
[demo_node-1] [INFO] [1622278568.222839760] [GraspScene]: Camera Point Cloud Received!
[demo_node-1] [INFO] [1622278568.222866905] [GraspScene]: Processing Point Cloud...
[demo_node-1] [INFO] [1622278568.314758446] [GraspScene]: Applying Passthrough filters
[demo_node-1] [INFO] [1622278568.388353728] [GraspScene]: Removing Statistical Outlier
[demo_node-1] [INFO] [1622278569.016152941] [GraspScene]: Downsampling Point Cloud
[demo_node-1] [INFO] [1622278569.025191278] [GraspScene]: Segmenting plane
[demo_node-1] [INFO] [1622278569.052835633] [GraspScene]: Point cloud successfully␣
↪processed!
[demo_node-1] [INFO] [1622278574.641794991] [GraspScene]: Extracting Objects from point␣
↪cloud
[demo_node-1] [INFO] [1622278575.542850053] [GraspScene]: Extracted 1 from point cloud
[demo_node-1] [INFO] [1622278575.542983933] [GraspScene]: Loading finger gripper robotiq_
↪2f
[demo_node-1] [INFO] [1622278575.543278724] [GraspScene]: All End Effectors Loaded
[demo_node-1] [INFO] [1622278575.723888039] [GraspScene]: Grasp planning time for␣
↪robotiq_2f 10 [ms]
[demo_node-1] [INFO] [1622278575.723914241] [GraspScene]: 19 Grasp Samples have been␣
↪generated.
```

- 1. Proceed to click on the Cloud Viewer window and it will show the pointcloud and bounding box of the object (Use the mouse scroll to view the pointclouds better).
- 2. Press the `Q` key within the Cloud Viewer window to view the results of the grasp_samples
- 3. The terminal running grasp_planner_launch.py will show the ranks of all ranked grasps on the object and the total number of grasps that can be sampled for the object.
- 4. First grasp visualized on the viewer is the best grasp.
- 5. Pressing `Q` will show the rest of the consecutively ranked grasps.
- 6. Once all the grasps have been screened through, the grasp_planner will publish the /grasp_tasks topic.

Warning: If the pointclouds shown on Cloud Viewer is not satisfactory, adjust the passthrough_filter_limits parameters defined in *Grasp Planner Configuration File* to suit to your desired environment.

Suction gripper:

```
[demo_node-1] [INFO] [1622278648.176338963] [GraspScene]: Using Direct Camera Input....
[demo_node-1] [INFO] [1622278648.177783690] [GraspScene]: waiting....
[demo_node-1] [INFO] [1622278652.348536010] [GraspScene]: Camera Point Cloud Received!
[demo_node-1] [INFO] [1622278652.348569074] [GraspScene]: Processing Point Cloud...
[demo_node-1] [INFO] [1622278652.453359019] [GraspScene]: Applying Passthrough filters
[demo_node-1] [INFO] [1622278652.531642625] [GraspScene]: Removing Statistical Outlier
[demo_node-1] [INFO] [1622278653.242787447] [GraspScene]: Downsampling Point Cloud
[demo_node-1] [INFO] [1622278653.257366582] [GraspScene]: Segmenting plane
[demo_node-1] [INFO] [1622278653.289915130] [GraspScene]: Point cloud successfully␣
↪processed!
[demo_node-1] [INFO] [1622278721.836112062] [GraspScene]: Extracting Objects from point␣
↪cloud
[demo_node-1] [INFO] [1622278722.761648258] [GraspScene]: Extracted 1 from point cloud
[demo_node-1] [INFO] [1622278722.761778564] [GraspScene]: Loading suction gripper␣
↪suction_cup
[demo_node-1] [INFO] [1622278722.761936044] [GraspScene]: All End Effectors Loaded
[demo_node-1] [INFO] [1622278723.371410203] [GraspScene]: Grasp planning time for␣
↪suction_cup 490 [ms]
[demo_node-1] [INFO] [1622278723.371440840] [GraspScene]: 64 Grasp Samples have been␣
↪generated.
```

- 1. Proceed to click on the Cloud Viewer window and it will show the pointcloud and bounding box of the object (Use the mouse scroll to view the pointclouds better).
- 2. Press the `Q` key within the Cloud Viewer window to view the results of the grasp_samples
- 3. The terminal running grasp_planner_launch.py will show the ranks of all ranked grasps on the object and the total number of grasps that can be sampled for the object.
- 4. First grasp visualized on the viewer is the best grasp.
- 5. Pressing `Q` will show the rest of the ranked grasps consecutively.
- 6. Once all the grasps have been screened through, the grasp_planner will publish the /grasp_tasks topic.

> **Warning:** If the pointclouds shown on Cloud Viewer is not satisfactory, adjust the passthrough_filter_limits parameters defined in *Grasp Planner Configuration File* to suit to your desired environment.

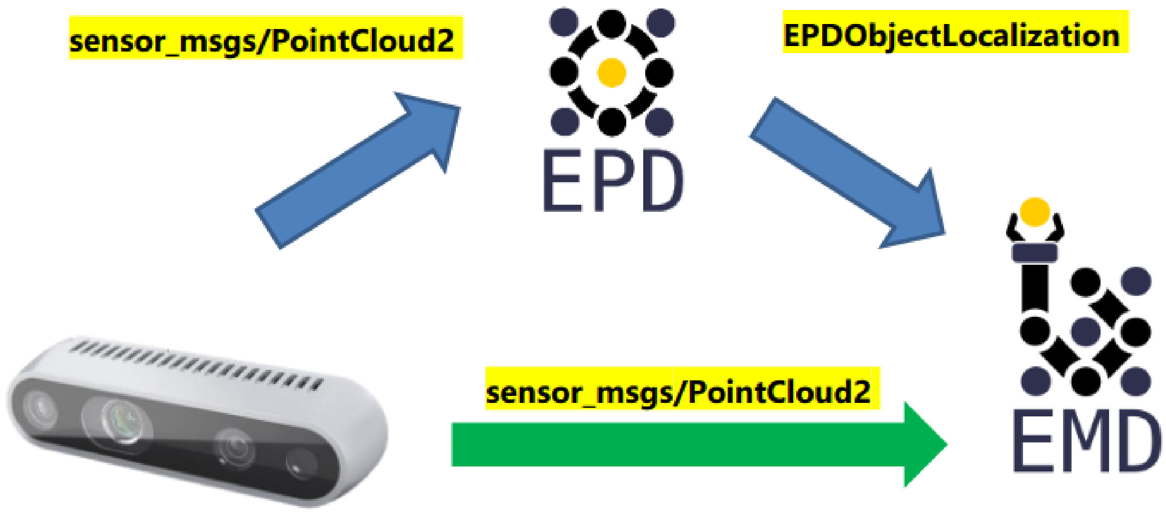
The pose and orientation of the top ranked grasp will then be published for *Grasp Execution Example*

7.4 Useful information

Information that you may need for further customization of the package.

7.4.1 Grasp Planner Input Message Types

For a grasp planner, a perception system will provide the necessary inputs required to plan grasps. Currently there are two main workflows for the grasp planner:



You can choose which workflow you want to use via the configuration file at *Grasp Planner General Parameters (EPD)*

Direct Camera Input

The EMD Grasp Planner can receive a PointCloud2 message type as input into the system. This is a commonly used message type for many cameras to send point clouds. The purpose of supporting this Input time is to provide a flexible option for users to choose the input type based on their needs and hardware specifications

EPD Input

The EMD Grasp Planner also supports output from the easy_perception_deployment

Currently the EMD Grasp Planner supports both **Object Localization** and **Object Tracking** outputs from EPD Precision level 3.

To understand more about the output messages from EPD, do visit the EPD documentation

Which EMD Workflow to choose?

1. Do you need to pick up all the objects in the area, or only specific ones/in a specific order?

All the objects in the area: Direct Camera Workflow

For the Direct Camera Workflow, as objects are determined through pointcloud processing rather than using deep learning methods of identification, even objects that may be typically hard for deep learning methods to detect can be detected through raw point clouds. As long as the camera can generate the point cloud, grasps can be planned for that object.

Specific Objects/ Specific Order: EPD Workflow

For the EPD workflow, the grasp area is first processed with a deep learning model, thus Object identity of each object as well as the location is known, thus users can determine which object should be picked, or which object should be picked first.

Note: Currently, EMD does not allow object prioritization as of now, but the EPD workflow is definitely the workflow required for this feature.

2. Preparation time before pipeline execution

Less time required: Direct Camera Workflow

For the Direct Camera Workflow, Zero training is needed, as no deep learning components are used. Thus no datasets of the objects is needed before running the whole pipeline. The only preprocessing step required is writing the configuration yaml file for the grasp planner: *Grasp Planner Configuration File*

More time required: EPD Workflow

As there is a deep learning component for the EPD workflow, labelled datasets for grasping objects is required to train the perception system, which may take some time to prepare especially if the user requires idenfication of larger number of objects.

3. Hardware requirements

Lower hardware requriements: Direct Camera Workflow

Without the need for Deep learning, the pick and place pipeline can generally run relatively well with CPU, and can be enhanced with GPU usage,

Greater hardware requriements: EPD Workflow

Due to the higher hardware limitations for deploying of deep learning models, hardware requirements will be higher for the EPD workflow. Check the easy_perception_deployment documentation for a more comprehensive hardware requirement specifications.

Which EMD Workflow to choose (TLDR)?

In summary, at a quick glance this would be how you could choose your workflow for your use case.

Requirements	EPD	Camera
Picking requirements:	Specific objects/order	All Objects/ No Order
Preparation time:	Slower	Faster
Hardware requirements:	Higher	Lower

7.4.2 Grasp Planner Output Message Types

This section provides an understanding of the output from the EMD Grasp Planner. The output from the planner is typically provided to the Grasp Execution Component of EMD, but you can also provide your own grasp execution solutions that takes in such messages.

The EMD Grasp Planner consists of a ROS2 client that submits a request of the list of objects to be picked and how to pick them.

Service Name : *grasp_requests*

GraspRequest.srv

- Service representing the entire pick and place operation. Contains a list of items (GraspTargets) to be grasped in the scene

Request name	Field Type	Explanation
grasp_targets	GraspTarget[]	Array of Grasp Targets (Refer below to GraspTarget message type)

Result name	Field Type	Explanation
success	bool	Indicates successful run of triggered service
message	string	Any other useful information from Grasp Execution

GraspTarget.msg

- Represents a single object to be picked. Contains a list of end effector grasp plans (GraspMethods)

Message name	Field Type	Explanation
target_type	string	Object ID. Object Names will be used if using EPD Workflow
target_pose	geometry_msgs/PoseStamped	Position and Orientation of target Object
target_shape	shape_msgs/SolidPrimitive	Shape of target object (Used to create collision objects for path planning)
grasp_methods	GraspMethod[]	Array of Grasp Targets (Refer below to GraspMethod message type)

GraspMethod.msg

- Represents a Single end effector option. Contains a list grasp poses for that gripper, sorted by ranks

Message name	Field Type	Explanation
ee_id	string	Name of End effector
grasp_poses	geometry_msgs/PoseStamped[]	Array of grasp poses
grasp_ranks	float32[]	Array of grasp ranks
grasp_markers	visualization_msgs/Marker[]	Array of markers representing grasp samples

7.5 Acknowledgements

Initial inspiration for grasp planning algorithim was provided by the following paper, and have been repurposed to support multiple fingers as well as suction cup arrays

Fast Geometry-based Computation of Grasping Points on Three-dimensional Point Clouds

CHAPTER

EIGHT

GRASP EXECUTION

8.1 Overview

The Easy Manipulation Deployment Grasp Execution package was developed to provide a robust path planning process to navigate robot to the taget location for grasping. The package serves as a grasp execution simulator using Moveit2 path planners and the results from the *Grasp Planner*.

Note: It is recommended that you use scene packages generated by the *Workcell Builder*. If you are using a robot, make sure you have the **moveit config folder** (check out *Workcell Initialization* for more information about the moveit_config folder)

8.1.1 Benefits of EMD Grasp Execution

1. Seamless intergration with EMD Grasp Planner

The EMD Grasp Execution package communicates with the Grasp Planner package through the subscription to a single ROS2 topic with the GraspTask.msg type. Details of the GraspTask Message can be found here: *Grasp Planner Output Message Types*

2. Dynamic Safety Capabilities

In real life use cases, collaborative robots often operate closely with human operators or reside in an ever-changing environment. There is thus a need for the robot to be equipped with the dynamic safety capability, to detect possible collision during its trajectory execution and avoid these occurring obstacles.

Grasp Execution provides users with a vision based dynamic collision avoidance capability using Octomaps. When a collision has been deemed to occur in the trajectory of the robot, the dynamic safety module will be triggered. This would either stop the robot to avoid collision, or call for the dynamic replanning of its trajectory given the new collision objects in the scene.

8.2 Package configuration

Before running the grasp execution package, make sure that you have a scene package in the `workcell/src/scenes/` folder.

8.2.1 Launch file

Open the `grasp_execution.launch.py` file in `/workcell/src/easy_manipulation_deployment/grasp_execution/example/launch/` and change the parameters accordingly.

scene_pkg

The name of your scene package

Example:

```
scene_pkg = 'ur5_2f_test'
```

base_link

The base link of the robot connected to the table surface

Example:

```
robot_base_link = 'base_link'
```

8.2.2 grasp_execution_node.cpp

Open the `demo_node.cpp` file in `/workcell/src/easy_manipulation_deployment/grasp_execution/example/src/` and change the parameters accordingly.

```
static const char PLANNING_GROUP[] = "manipulator";

static const char EE_LINK[] = "ee_palm";

static const float CLEARANCE = 0.1;

static const char GRASP_TASK_TOPIC[] = "grasp_tasks";
```

PLANNING_GROUP

Robot group state that is declared in the srdf. for example, in the file `ur5_moveit_config/config/ur5.srdf.xacro`:

```
<!--GROUPS: Representation of a set of joints and links. This can be useful for␣
↪specifying DOF to plan for, defining arms, end effectors, etc-->
<group name="manipulator">
    <chain base_link="base_link" tip_link="ee_link" />
</group>
```

the planning_group is then *manipulator*

EE_LINK

Tip link of end effector

CLEARANCE

Distance above the object that the end effector would plan to before moving down to pick the object up

GRASP_TASK_TOPIC

ROS2 topic from the Grasp Planning component that will output the GraspTask message

8.2.3 Addidional configurations

Other configurations that can be customized for grasp execution

Grasp Execution Configuration Files

Grasp Execution has many features and capabilities that can be turned on and off, and customized to the your liking. This is done in the configuration files, that are typically stored in the config folder of your package. The YAML format is used for this file for better understanding and readability.

Below lists the configuration files in the package that change different parameters in the grasp execution pipeline.

- Changing the start positions
- Changing the fake object published
- Changing the grasp execution parameters
- Changing the dynamic safety execution parameters
- Changing the workcell configurations

Changing the start positions

To change the home state, edit the values of each joint in the file `grasp_execution/example/config/start_positions.yaml`:

```
initial_positions:
  shoulder_pan_joint: 1.57
  shoulder_lift_joint: -2.35
  elbow_joint: 1.83
  wrist_1_joint: -1.03
  wrist_2_joint: -1.57
  wrist_3_joint: 0.0
```

Changing the fake object published

To change the location and dimensions of the fake object published, edit following parameters in the file `grasp_execution/example/config/fake_grasp_pose_publisher.yaml`:

```
fake_grasp_pose_publisher:
  ros__parameters:
    interface: service
    frame_id: camera_frame
    ee_id: robotiq_2f
    grasp_pose: [0.0126, 0.0322, 0.442,
                 0.0, 0.0, 0.9997, 0.0250]

    object_pose: [0.0128, 0.0330, 0.443,
                  0.0, 0.0, 0.9997, 0.0250]

    object_dimensions: [0.087, 0.167, 0.023]

    delay: 1.5
```

Name	Type	Description
frame_id	String	Base frame
grasp_pose	double array	Location that the robot will plan to
object_pose	double array	Location that the object will spawn at
object_dimensions	double array	Dimensions of the object (x,y,z)
delay	double	

Changing the grasp execution parameters

To change the configuration of the default grasp execution, edit following parameters in the file `grasp_execution/example/config/grasp_execution.yaml`:

```
grasp_execution_node:
  ros__parameters:
    planning_scene_monitor_options:
      name: "planning_scene_monitor"
      robot_description: "robot_description"
      joint_state_topic: "/joint_states"
      attached_collision_object_topic: "/moveit_cpp/planning_scene_monitor"
      publish_planning_scene_topic: "/moveit_cpp/publish_planning_scene"
      monitored_planning_scene_topic: "/moveit_cpp/monitored_planning_scene"
      wait_for_initial_state_timeout: 10.0

    planning_pipelines:
      #namespace: "moveit_cpp"  # optional, default is ~
      pipeline_names: ["ompl"]

    plan_request_params:
      planning_attempts: 1
      planning_time: 0.5
      planning_pipeline: ompl
      max_velocity_scaling_factor: 1.0
      max_acceleration_scaling_factor: 1.0
```

Table 1: planning_scene_monitor_options

Name	Type	Description
name	string	
robot_description	string	
joint_state_topic	string	
attached_collision_object_topic	string	
publish_planning_scene_topic	string	
monitored_planning_scene_topic	string	
wait_for_initial_state_timeout	double	

Table 2: planning_pipelines

Name	Type	Description
pipeline_names	string array	Planning pipelines to be used (as of now only ompl is supported)

Table 3: plan_request_params

Name	Type	Description
planning_attempts	int	Number of planning attempts
planning_pipeline	string	planning pipeline used
max_velocity_scaling_factor	double	Maximum velocity scale
max_acceleration_scaling_factor	double	Maximum acceleration scale

Changing the dynamic safety execution parameters

To change the configuration of the grasp execution with dynamic safety, edit following parameters in the file `grasp_execution/example/config/dynamic_safety_demo.yaml`:

```
dynamic_safety_demo_node:
  ros__parameters:
    planning_scene_monitor_options:
      name: "planning_scene_monitor"
      robot_description: "robot_description"
      joint_state_topic: "/joint_states"
      attached_collision_object_topic: "/moveit_cpp/planning_scene_monitor"
      publish_planning_scene_topic: "/moveit_cpp/publish_planning_scene"
      monitored_planning_scene_topic: "/moveit_cpp/monitored_planning_scene"
      wait_for_initial_state_timeout: 10.0

    planning_pipelines:
      #namespace: "moveit_cpp"  # optional, default is ~
      pipeline_names: ["ompl"]

    plan_request_params:
      planning_attempts: 1
      planning_time: 0.5
      planning_pipeline: ompl
      max_velocity_scaling_factor: 1.0
      max_acceleration_scaling_factor: 1.0

    # Load octomap
    load_octomap: true

    # Dynamic safety parameters
    rate: 20
    allow_replan: true
    visualize: true

    safety_zone:
      manual: true
      unit_type: second
      collision_checking_deadline: 0.05
      slow_down_time: 0.2
      replan_deadline: 1.2
      look_ahead_time: 1.65

    collision_checker:
```

(continues on next page)

(continued from previous page)

```
    distance: false
    continuous: false
    step: 0.1
    thread_count: 8
    realtime: false

  next_point_publisher:
    command_out_type: "trajectory_msgs/JointTrajectory"
    publish_joint_position: true
    publish_joint_velocity: false
    publish_joint_effort: false

  replanner:
    planner_name: ompl

  visualizer:
    publish_frequency: 10
    step: 0.1
    topic: "/dynamic_safety/displayed_state"
```

The first half of the parameters are the same as described in grasp_execution.yaml

Name	Type	Description
load_octomap	bool	Load the octomap

Name	Type	Description
rate	double	
allow_replan	bool	Replan if collision is detected. If set to false the robot will simply stop to avoid collision
visualize	bool	

Table 4: safety_zone

Name	Type	Description
manual	bool	
unit_type	string	
collision_checking_deadline	double	
slow_down_time	double	
replan_deadline	double	
look_ahead_time	double	

Table 5: collision_checker

Name	Type	Description
distance	bool	
continuous	bool	
step	double	
thread_count	int	
realtime	bool	

Table 6: next_point_publisher

Name	Type	Description
command_out_type	string	
publish_joint_position	bool	
publish_joint_velocity	bool	
publish_joint_effort	bool	

Table 7: replanner

Name	Type	Description
planner_name	string	

Table 8: visualizer

Name	Type	Description
publish_frequency	double	
step	double	
topic	string	

Changing the workcell configurations

To change the configuration of your workcell, edit following parameters in the file `grasp_execution/example/config/dynamic_safety_demo.yaml`:

```
workcell:
- group_name: manipulator
  executors:
    default:
      plugin: grasp_execution/DefaultExecutor
    ds_async:
      plugin: grasp_execution/DynamicSafetyAsyncExecutor
      controller: ur5_arm_controller
  end_effectors:
    robotiq_2f0:
      brand: robotiq_2f
      link: ee_palm
      clearance: 0.1
      driver:
        plugin: grasp_execution/DummyGripperDriver
        controller: ""
```

Name	Type	Description
group_name	string	
end_effectors.robotiq_2f0.link	string	Tip link of end effector
end_effectors.robotiq_2f0.clearance	double	Distance above the object that the end effector would plan to before moving down to pick the object up

8.3 Running grasp execution

```
ros2 launch grasp_execution grasp_execution.launch.py
```

You should then see RViz launch with the initial workcell scene, where the robot arm is at its home state.

Note: Ensure that on your terminal, the ROS2 distribution is sourced, Moveit2 is sourced, and the workspace is built and sourced as well.

8.3.1 Publishing a Fake Task

```
ros2 launch grasp_execution fake_task_publisher.launch.xml
```

This publishes an object for the robot to conduct a pick-and-place operation. The location and dimensions of the object can be configured in `grasp_execution/example/config/fake_grasp_pose_publisher.yaml` as described in the *configuration page*.

CHAPTER

NINE

STEP-BY-STEP TUTORIALS

This section presents a highly detailed end to end example of a pick and place manipulation pipeline using the easy_manipulation_deployment package. If you want general information about the packages, you can look at the other sections of the documentation. **It is recommended, if you follow this tutorial, to follow it throughout** to reduce any errors stemming from partial actions.

9.1 Workcell Builder Example

In this example we will be creating a simple robotic workcell using the UR5 and the Robotiq-2F gripper. The expected scene will be as shown. (Note that the robot is currently not in a home pose because the grasp execution node has not been initialized with this visualization) **In the package we include the UR and Robotiq description and moveit_config folders, but in this tutorial we will show you how to do it from scratch**

9.1.1 Before running the GUI

The following instructions provides an example of how you can incorporate new robots and end effector ROS1 packages into this package. There are currently a few robots and end-effectors included in the package, so if you do not need to add any more of these packages, skip forward to : *Starting the Workcell Builder*

Downloading Robot and End effector resources

Assuming that you have followed the *Download Instructions* and have successfully installed the workcell builder, remove all folders in the workcell_ws/src/assets/robots , workcell_ws/src/assets/end_effectors , and workcell_ws/src/assets/environment_objects folders.

Your resulting ROS2 workspace should look like this

```
|--workcell_ws
___|--src
______|-- ....other folders
______|--scenes
______|--assets
_________|--robots
_________|--end_effectors
_________|--environment_objects
```

Universal Robot

Next we will get the ur_description and ur5_moveit config folders from the ROS-Industrial Universal Robots repository . For this example, we can use *the kinetic-devel* branch

Clone the repository in the `assets/robots` folder. For this example we only require the `ur_description` and `ur5_moveit_config` folders, thus we remove the other folders for now.

Robotiq End Effector

Next we will get the robotiq_85_description and robotiq_85_moveit_config folders from the Robotiq gripper repository

Clone the repository in the `assets/robots` folder. For this example we only require the `robotiq_85_description` and `robotiq_85_moveit_config` folders, thus we remove the other folders for now.

Your workspace should look like this.

```
|--workcell_ws
___|--src
______|-- ....other folders
______|--scenes
______|--assets
_________|--robots
____________|--universal_robot
_______________|--ur5_moveit_config
_______________|--ur_description
_________|--end_effectors
____________|--robotiq_85_gripper
_______________|--robotiq_85_description
_______________|--robotiq_85_moveit_config
_________|--environment_objects
```

Edit CMakelists.txt and package.xml

As this example will be run on ROS2 Foxy, we will need to make some changes to the CMakelists and package.xml

Universal Robot

In the `/assets/robots/ur_description/CMakeLists.txt`, replace the contents with the following:

```
cmake_minimum_required(VERSION 3.10.2)
project(ur_description)
find_package(ament_cmake REQUIRED)

install(DIRECTORY meshes DESTINATION "share/${PROJECT_NAME}")
install(DIRECTORY urdf DESTINATION "share/${PROJECT_NAME}")
ament_package()
```

In the `/assets/robots/ur_description/package.xml`, replace the contents with the following:

```
<?xml version="1.0"?>
<package format="3">
  <name>ur_description</name>
  <version>1.2.7</version>
  <description>
      URDF description for Universal UR5/10 robot arms
  </description>

  <author>Wim Meeussen</author>
  <author>Kelsey Hawkins</author>
  <author>Mathias Ludtke</author>
```

(continues on next page)

(continued from previous page)

```
  <author>Felix Messmer</author>
  <maintainer email="g.a.vanderhoorn@tudelft.nl">G.A. vd. Hoorn</maintainer>
  <maintainer email="miguel.prada@tecnalia.com">Miguel Prada Sarasola</maintainer>
  <maintainer email="nhg@ipa.fhg.de">Nadia Hammoudeh Garcia</maintainer>

  <license>BSD</license>

  <url type="website">http://ros.org/wiki/ur_description</url>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <exec_depend>joint_state_publisher</exec_depend>
  <exec_depend>robot_state_publisher</exec_depend>
  <exec_depend>rviz</exec_depend>
  <exec_depend>urdf</exec_depend>
  <exec_depend>xacro</exec_depend>

  <export>
      <build_type>ament_cmake</build_type>
  </export>
</package>
```

In the /assets/robots/ur5_moveit_config/CMakeLists.txt, replace the contents with the following:

```
cmake_minimum_required(VERSION 3.10.2)
project(ur5_moveit_config)
find_package(ament_cmake REQUIRED)


install(DIRECTORY config DESTINATION "share/${PROJECT_NAME}")
install(DIRECTORY launch DESTINATION "share/${PROJECT_NAME}")
install(DIRECTORY tests DESTINATION "share/${PROJECT_NAME}")
ament_package()
```

In the /assets/robots/ur5_moveit_config/package.xml, replace the contents with the following:

```
<?xml version="1.0"?>
<package format="3">
  <name>ur5_moveit_config</name>
  <version>1.2.7</version>
  <description>
     An automatically generated package with all the configuration and launch files for␣
↪using the ur5 with the MoveIt Motion Planning Framework
  </description>
  <author>Felix Messmer</author>
  <maintainer email="g.a.vanderhoorn@tudelft.nl">G.A. vd. Hoorn</maintainer>
  <maintainer email="miguel.prada@tecnalia.com">Miguel Prada Sarasola</maintainer>
  <maintainer email="nhg@ipa.fhg.de">Nadia Hammoudeh Garcia</maintainer>

  <license>BSD</license>

  <url type="website">http://moveit.ros.org/</url>
```

(continues on next page)

(continued from previous page)

```
  <url type="bugtracker">https://github.com/ros-planning/moveit_setup_assistant/issues</
↪url>
  <url type="repository">https://github.com/ros-planning/moveit_setup_assistant</url>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <exec_depend>joint_state_publisher</exec_depend>
  <exec_depend>robot_state_publisher</exec_depend>
  <exec_depend>xacro</exec_depend>
  <depend>ur_description</depend>

  <export>
      <build_type>ament_cmake</build_type>
  </export>
</package>
```

Robotiq End Effector

In the /assets/end_effectors/robotiq_85_gripper/robotiq_85_description/CMakeLists.txt, replace the contents with the following:

```
cmake_minimum_required(VERSION 3.10.2)
project(robotiq_85_description)
find_package(ament_cmake REQUIRED)

install(DIRECTORY meshes DESTINATION "share/${PROJECT_NAME}")
install(DIRECTORY urdf DESTINATION "share/${PROJECT_NAME}")
ament_package()
```

In the /assets/end_effectors/robotiq_85_gripper/robotiq_85_description/package.xml, replace the contents with the following:

```
<?xml version="1.0"?>
<package format="3">
 <name>robotiq_85_description</name>
  <version>0.6.4</version>
  <description>Stanley Innovation Robotiq 85 Visual Models</description>
  <maintainer email="dev@stanleyinnovation.com">Patrick Hussey</maintainer>
  <author>Patrick Hussey</author>

  <license>BSD</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <exec_depend>joint_state_publisher</exec_depend>
  <exec_depend>robot_state_publisher</exec_depend>

  <export>
      <build_type>ament_cmake</build_type>
  </export>
</package>
```

In the /assets/end_effectors/robotiq_85_gripper/robotiq_85_moveit_config/CMakeLists.txt, replace the contents with the following:

```
cmake_minimum_required(VERSION 3.10.2)
project(robotiq_85_moveit_config)
find_package(ament_cmake REQUIRED)


install(DIRECTORY config DESTINATION "share/${PROJECT_NAME}")
install(DIRECTORY launch DESTINATION "share/${PROJECT_NAME}")
ament_package()
```

In the /assets/end_effectors/robotiq_85_moveit_config/package.xml, replace the contents with the following:

```
<package>
  <name>robotiq_85_moveit_config</name>
  <version>0.2.0</version>
  <description>
     An automatically generated package with all the configuration and launch files for using the robotiq_85_gripper with the MoveIt Motion Planning Framework
  </description>
  <author email="assistant@moveit.ros.org">MoveIt Setup Assistant</author>
  <maintainer email="assistant@moveit.ros.org">MoveIt Setup Assistant</maintainer>

  <license>BSD</license>

  <url type="website">http://moveit.ros.org/</url>
  <url type="bugtracker">https://github.com/ros-planning/moveit_setup_assistant/issues</url>
  <url type="repository">https://github.com/ros-planning/moveit_setup_assistant</url>

<buildtool_depend>ament_cmake</buildtool_depend>

  <exec_depend>joint_state_publisher</exec_depend>
  <exec_depend>robot_state_publisher</exec_depend>
  <exec_depend>xacro</exec_depend>
  <build_depend>robotiq_85_description</build_depend>
  <exec_depend>robotiq_85_description</exec_depend>

  <export>
      <build_type>ament_cmake</build_type>
  </export>
</package>
```

Xacro-ize the SRDFs

As this workcell builder aims to create links between the manipulator and end effector, the semantic descriptions need to be accessible as macros.

In the /assets/end_effectors/robotiq_85_gripper/robotiq_85_moveit_config/config folder, make a copy of robotiq_85_gripper.srdf and rename it robotiq_85_gripper.srdf.xacro . in this file, add the xacro tags <xacro:macro name="robotiq_85"> and :code:` </xacro:macro>` to the start and end of the file, as well as adding the XML NameSpace <robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="robotiq_85_gripper">

Your robotiq_85_gripper.srdf.xacro file should be as shown

```
<?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 defined
-->
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="robotiq_85_gripper">
<xacro:macro name="robotiq_85_gripper">
...
...
...

    <disable_collisions link1="gripper_root_link" link2="robotiq_coupler_link" reason="Adjacent" />
</xacro:macro>
</robot>
```

In the /assets/end_effectors/robotiq_85_gripper/ur5_moveit_config/config folder, make a copy of ur5.srdf and rename it ur5.srdf.xacro . in this file, add the xacro tags <xacro:macro name="ur5"> and :code:` </xacro:macro>` to the start and end of the file, as well as adding the XML NameSpace <robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="ur5">

Your ur5.srdf.xacro file should be as shown

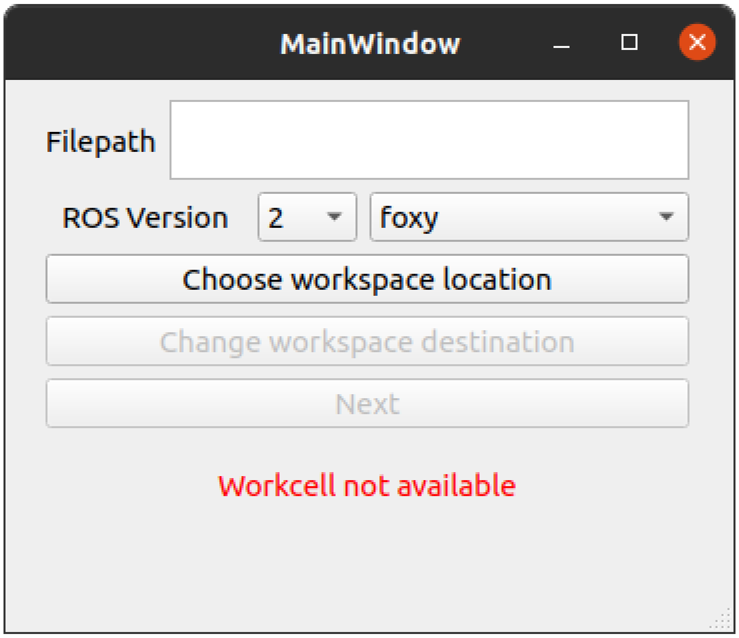
```
<?xml version="1.0" ?>
<!--This does not replace URDF, and is not an extension of URDF.
    This is a format for representing semantic information about the robot structure.
    A URDF file must exist for this robot as well, where the joints and the links that are referenced are defined
-->
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="ur5">
<xacro:macro name="ur5">
...
...
...

    <disable_collisions link1="wrist_2_link" link2="wrist_3_link" reason="Adjacent" />
</xacro:macro>
</robot>
```

Next step: *Starting the Workcell Builder*

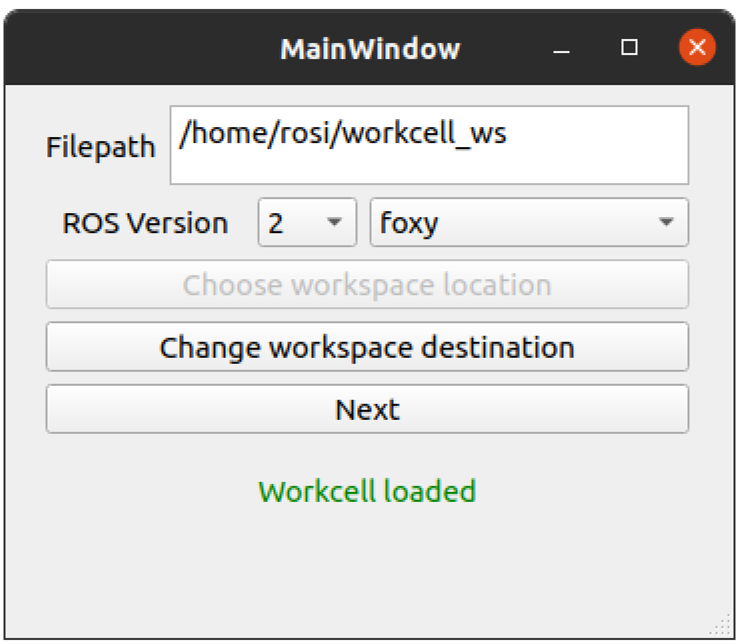
9.1.2 Starting the Workcell Builder

In the working directory /workcell_ws/ (Important!), run the workcell builder

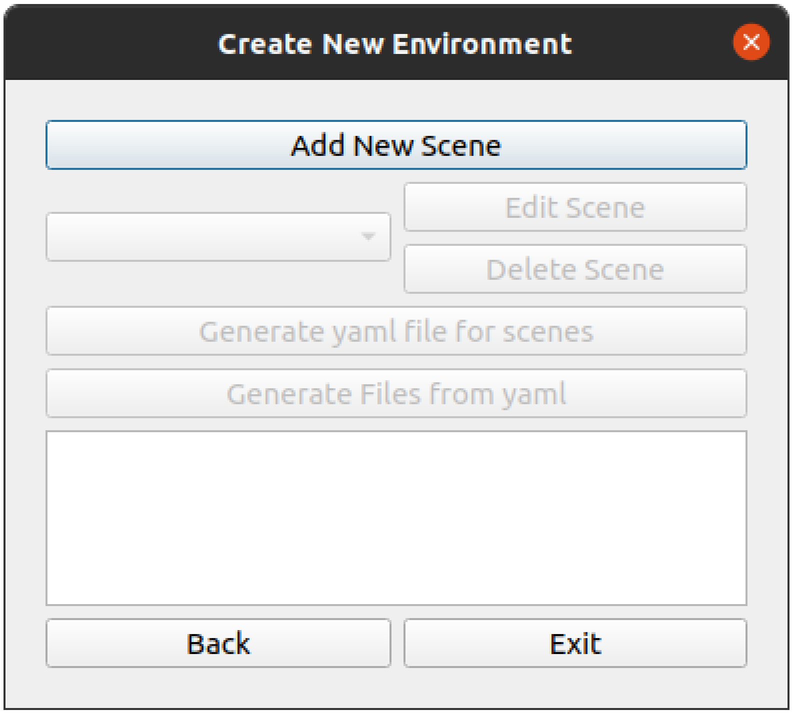
```
workcell_builder
```

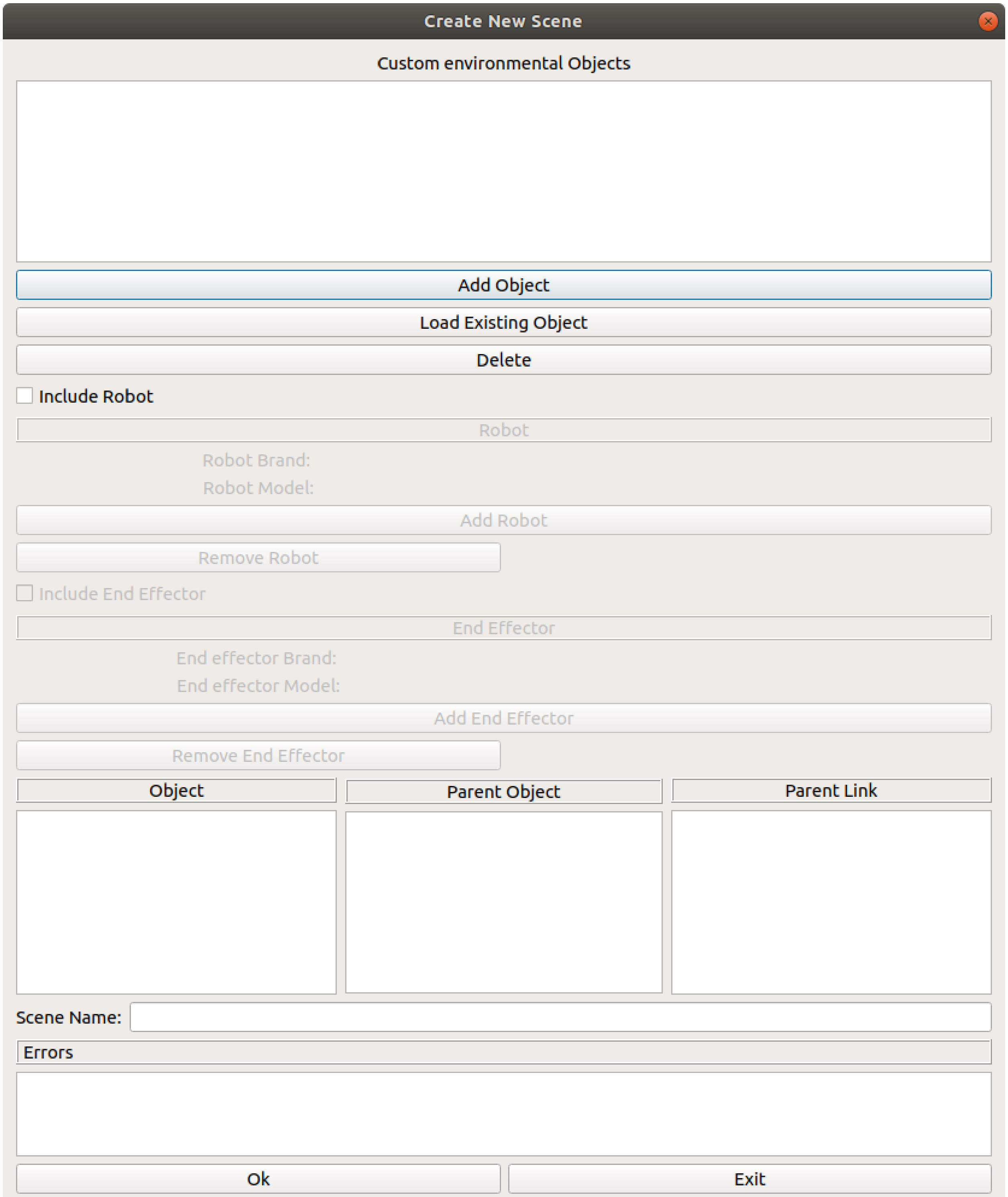


Set the ROS version and Distro to ROS2 Foxy. Next, click `Choose workspace location` and select the filepath to `workcell_ws`. You should see a green confirmation message that the workcell is loaded.



Click `Next`. you will then see the scene selection screen. Click on `New Scene`.


Create New Environment
Add New Scene
Edit Scene
Delete Scene
Generate yaml file for scenes
Generate Files from yaml
Back
Exit



Next step: *Adding A Manipulator*

9.1.3 Adding A Manipulator

Before you can add an end effector you need to first connect a robot to the world. Check the `Include Robot` and click `Add Robot`

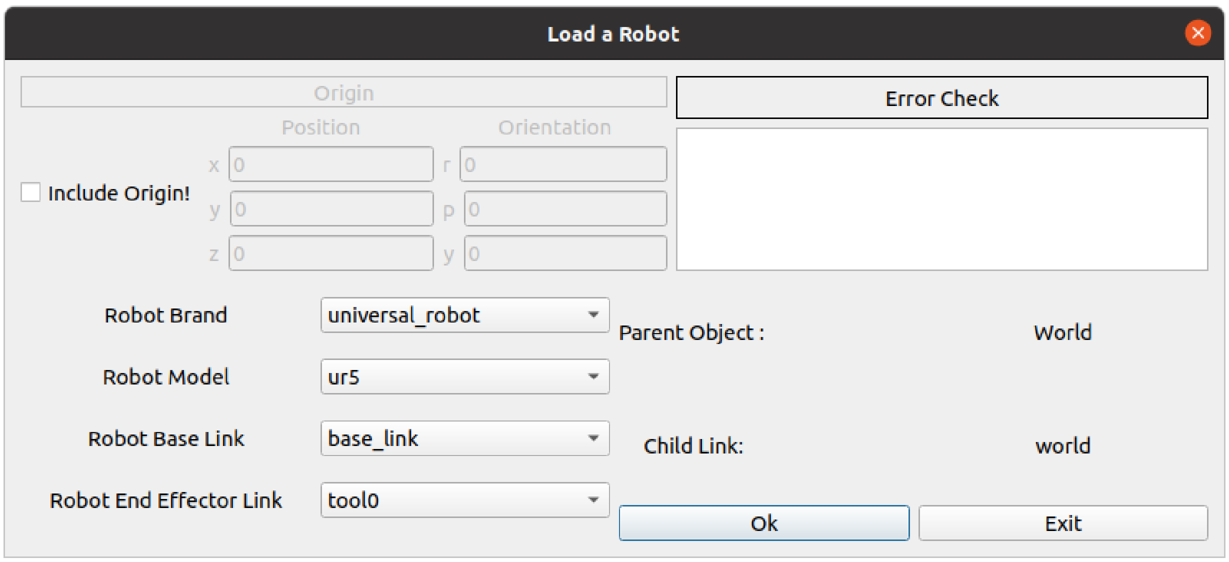
In the `Robot Brand` field , choose `Universal Robot`

In the `Robot Model` field, choose `ur5`

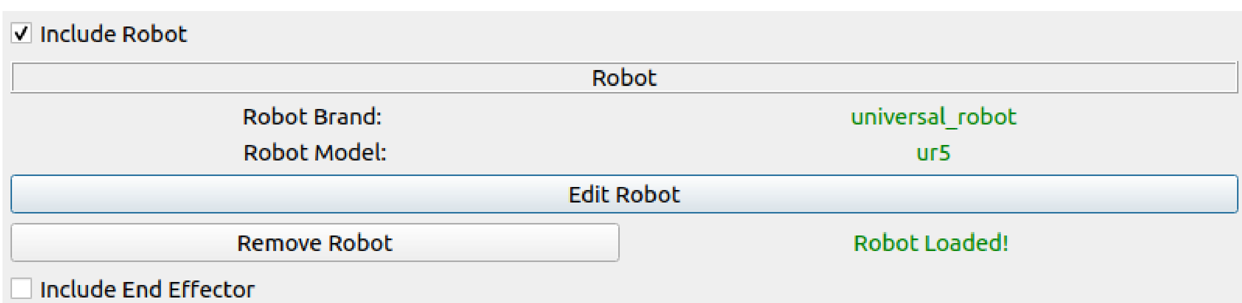
In the `Robot Base Link` field, choose `base_link`

In the `Robot End Effector Link` field, choose `ee_link`

Since we want the robot to be at the origin point of the world, we can leave the origin field unchecked. Your final window should look like this:

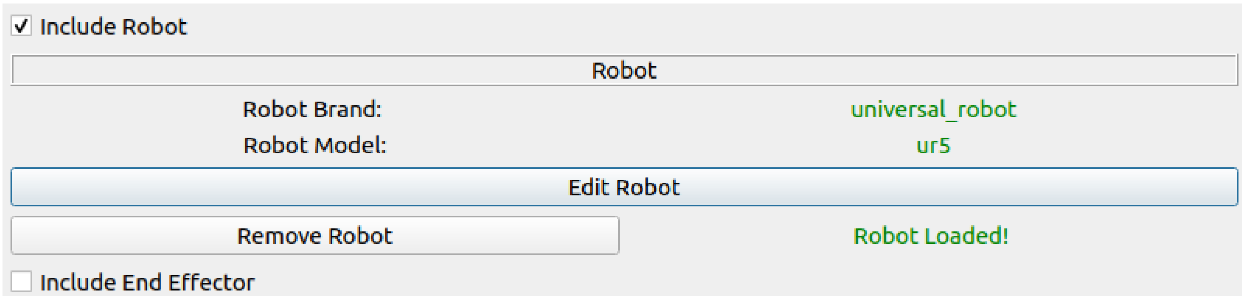


After clicking `Ok`, you returning to the scene creation window, there should be a confirmation that the robot is loaded and the option to include an end effector



Next step: *Adding an end effector*

9.1.4 Adding an end effector



In the scene creation window, check the `Include End Effector` and click `Add End Effector`

In the `End Effector Brand` field , choose `robotiq_85_gripper`

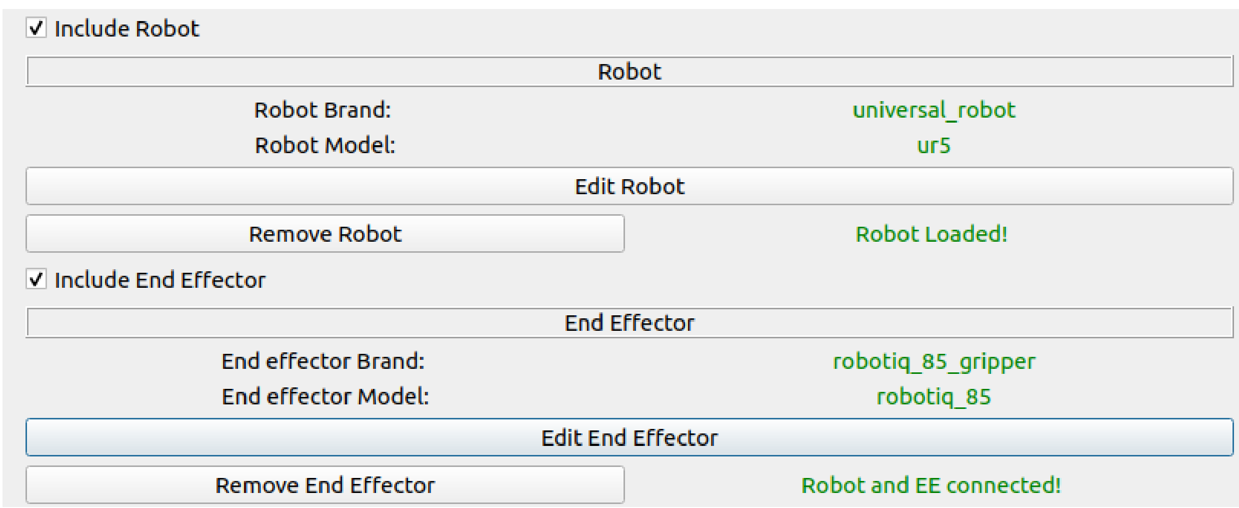
In the `End Effector Model` field, choose `robotiq_85`

In the `End Effector Link` field, choose `gripper_base_link`

In the `End Effector Type` field, choose `finger`

In the `Fingers` field, choose `2`

Since we want the end effector to be right at the point of the robot's end effector link, we can leave the origin field unchecked. If you want a certain offset for your end effector with respect to the robot, you can add an origin component with your required values. Your final window should look like this:



click `Ok` , and the scene creation window will show an end effector loaded confirmation as well.

Next step: *Adding an Object*

9.1.5 Adding an Object

This portion of the tutorial is to provide a guide on how to add a new object using the STL files. if you are planning on just using the existing table package created for you, skip to: *Loading an Object*

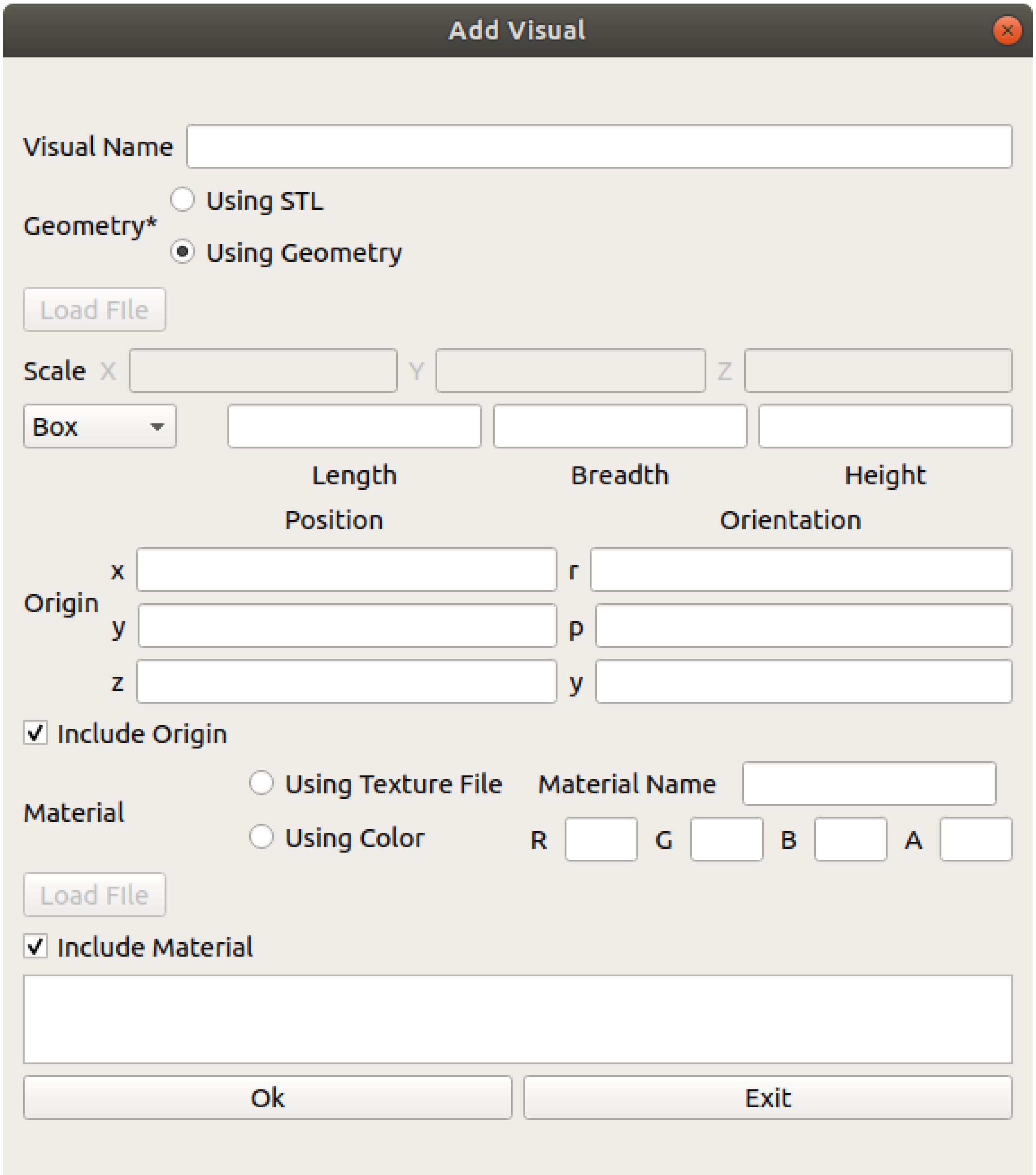
Next, we will create an Environment Object, the table. Click `Add Object`

In the `Object Name`, give your object a name, i.e *table*.

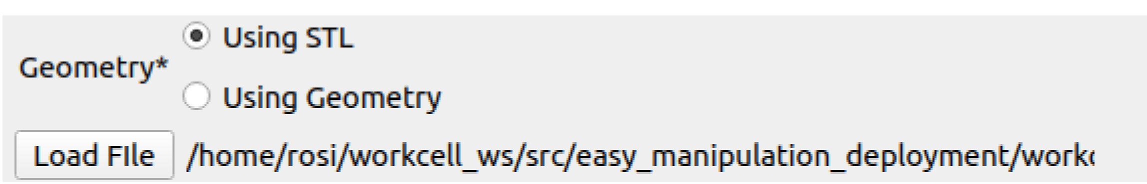
Click on `Add New Link`.

Create visual component of link

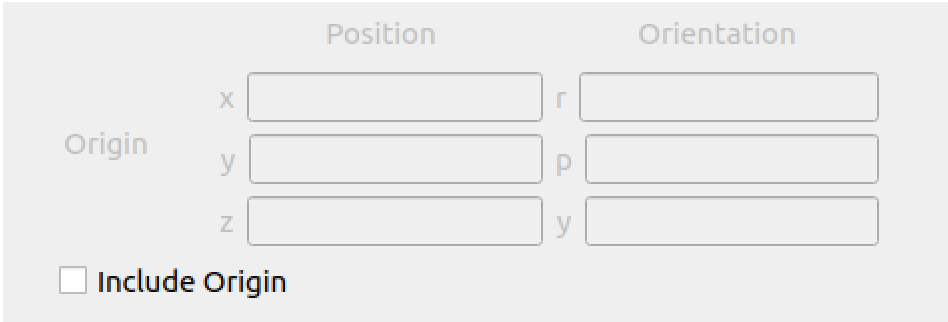
Give your link a name, e.g "table". In order to visualize the object, you need to have a link with a visual component. Check `Enable Viusal` and click `Create Visual`



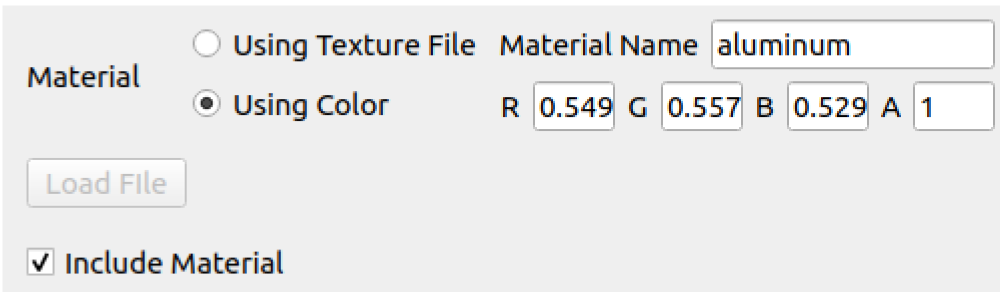
For this example, we will use an stl file. Select `Using STL` and click `Load File`. Select the location of your stl file. For this example, the `table.stl` file will be located at `/workcell_ws/src/easy_manipulation_deployment/workcell_builder/examples/resources/`. The stl file is currently too big, so we shall resize it by a factor of 0.001 on all axes (X, Y, Z)



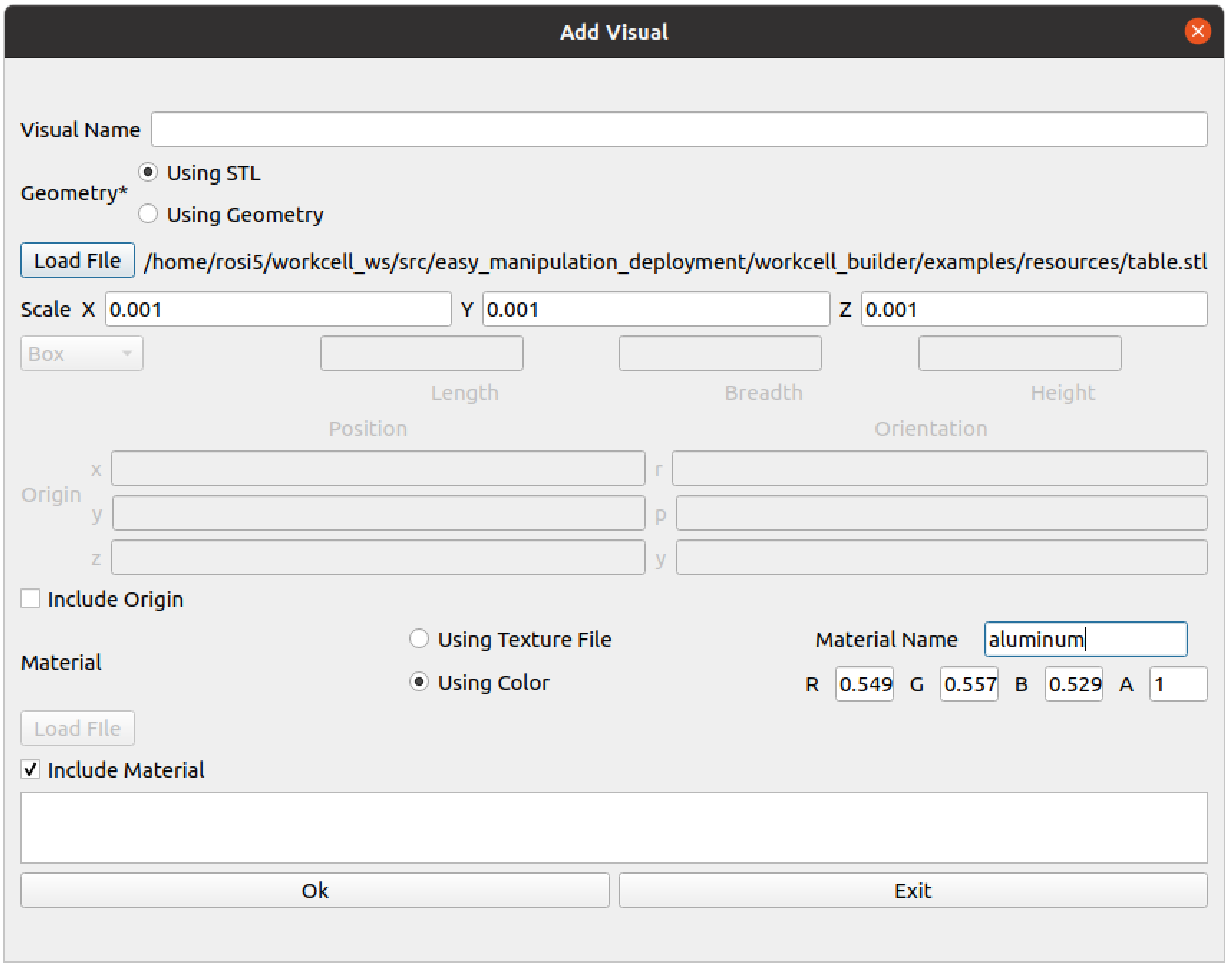
The table in this example will also be at the origin of the workcell world, so we will leave the origin unchecked.



The workcell table we have is slightly grey. Uncheck the `Include Material` , toggle to `Using color` and enter the following numbers into the RGBA fields. Name the material "aluminum".



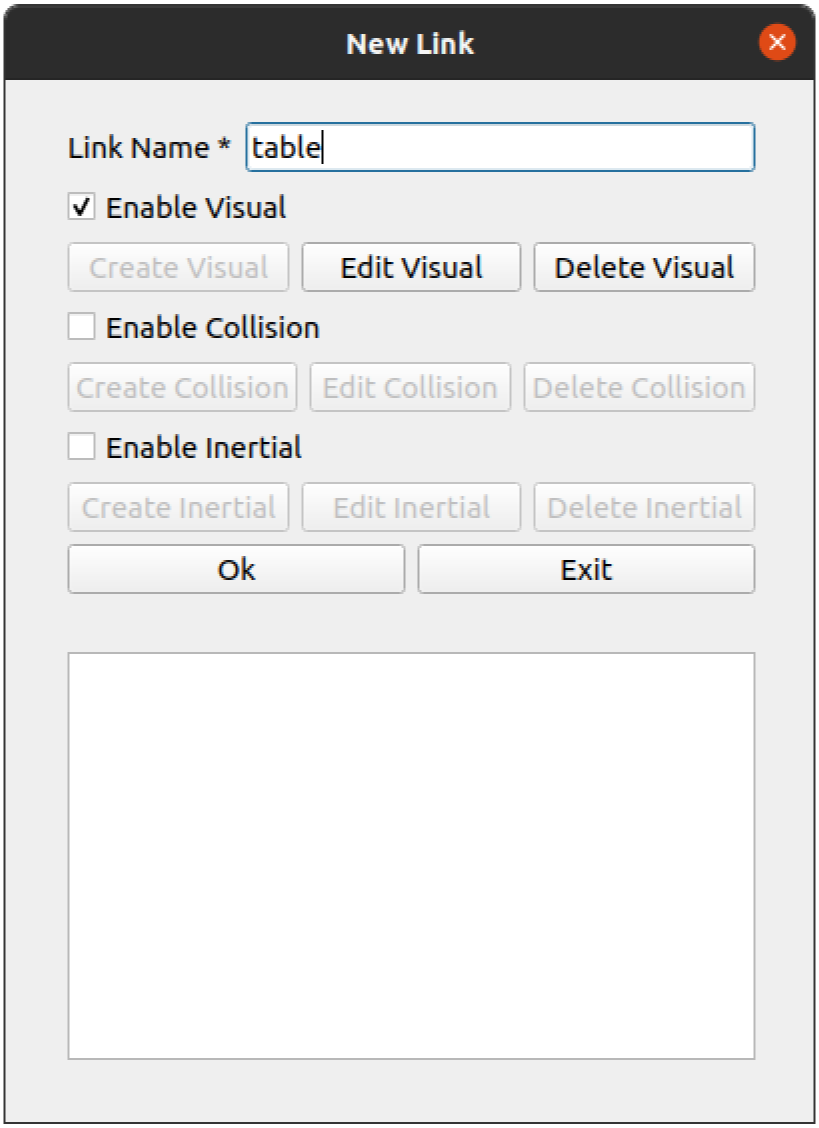
Your final visual_link window should look like this:



Click ok.

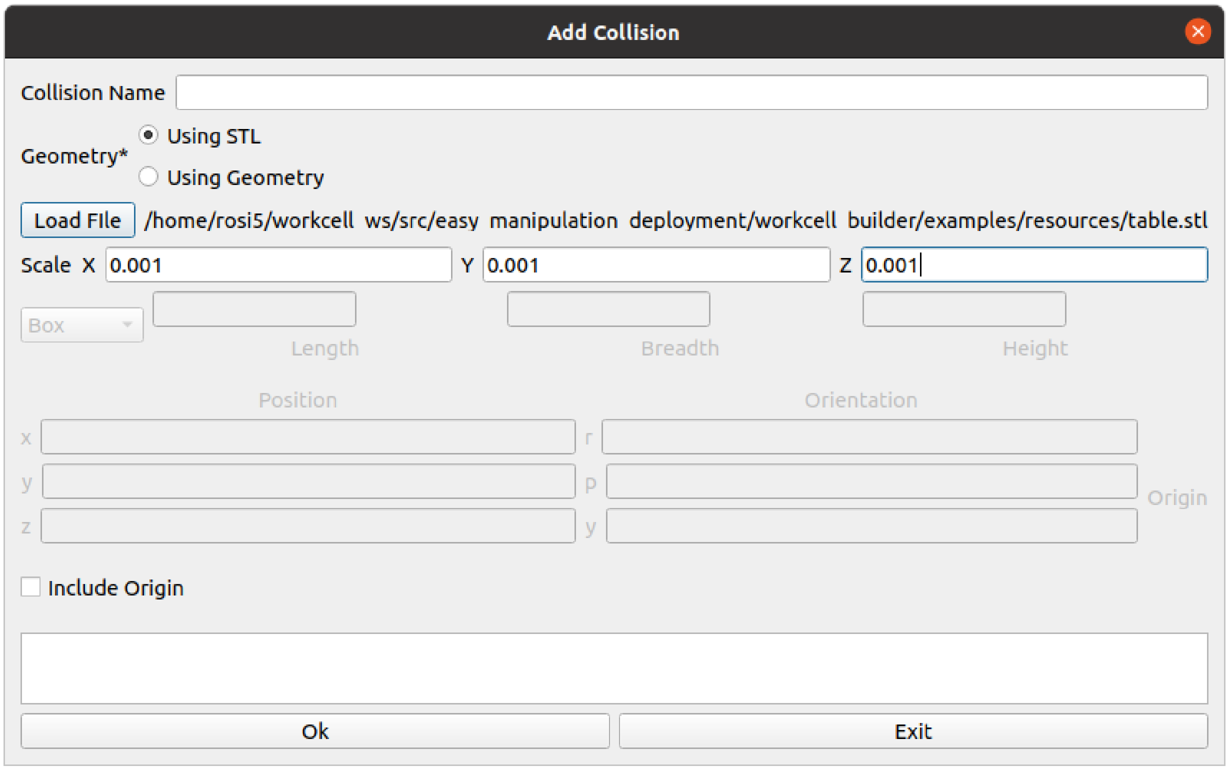
Create Collision component of link

Next, if you want this table object to be accounted for as a collision object, you need to add a collision component. Check `Enable Collision` and click `Create Collision`

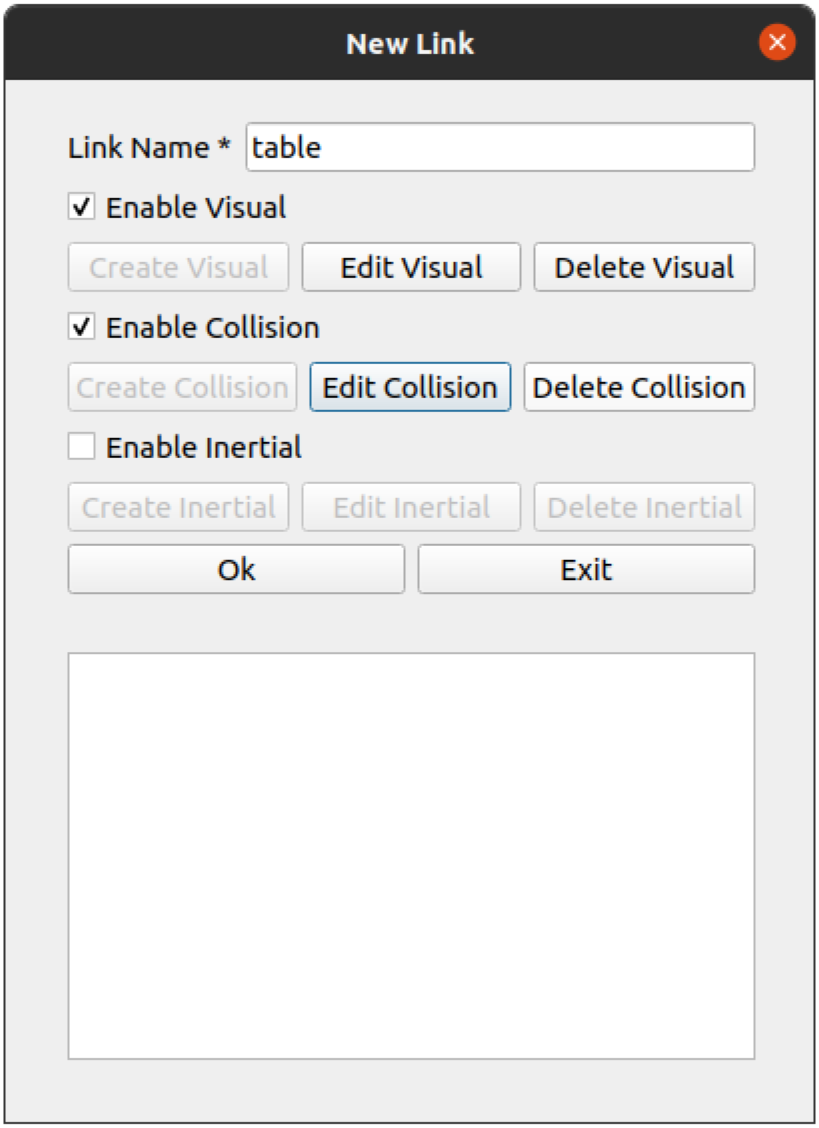


The steps to filling up this window is identical to adding visual component, where you add in the geometry. Similar to the visual component, you want your collision component to be in the shape of the table as well, hence we use the same stl as before.

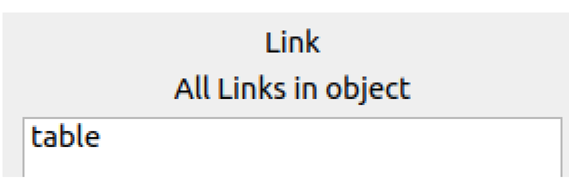
Your final collision_link window should look like this:



For this example, we do not require an inertial component so we will skip that option. Your final new link window should look like the following



Click Ok. Your link should now be displayed in the link window.



As this is a relatively simple environment object, there is only one link needed, and hence no internal joints need to be declared.

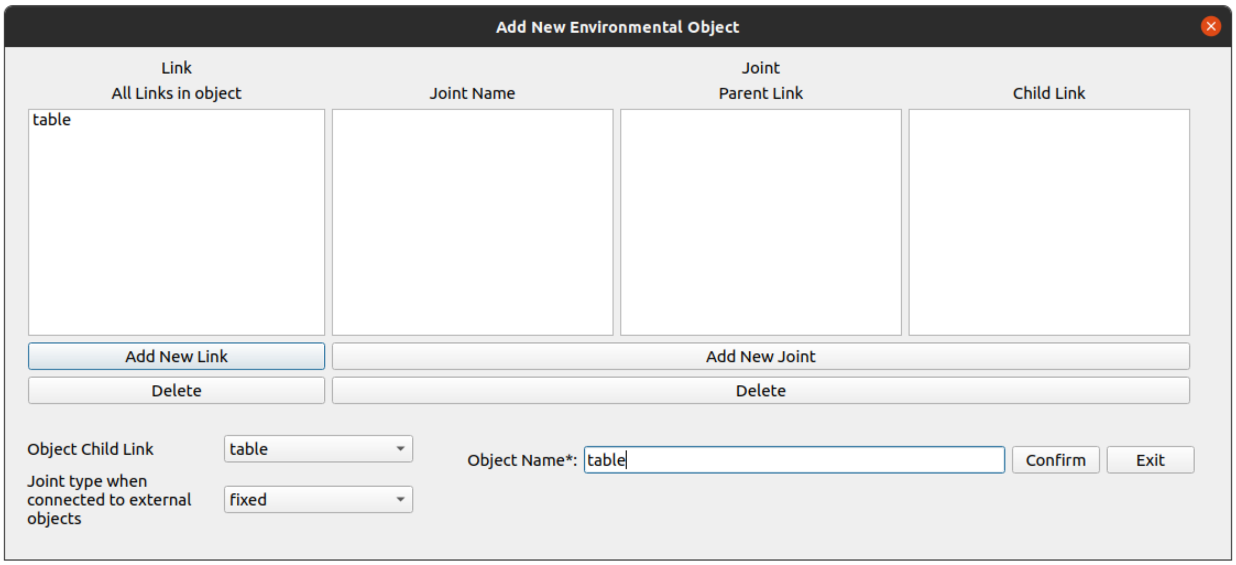
Set external joint attributes

In a scene, all objects need to be connected by at least one link on the object (as the child link) to another link (parent link) via an external joint. The only exception is `world`, which is an independant link that does not need to be connected to any object. In other words, it is the first parent link of the entire scene tree.

For our table, we will first choose the `child_link` that we want for the external joint, as well as the external joint type. For now, since the table only has one link, we select the link `table`. Set the external joint type to be `fixed` as well.

Make sure to name your object as well. We can simply name it *table*

Your final `Add New Environmental Object` window should look like this:



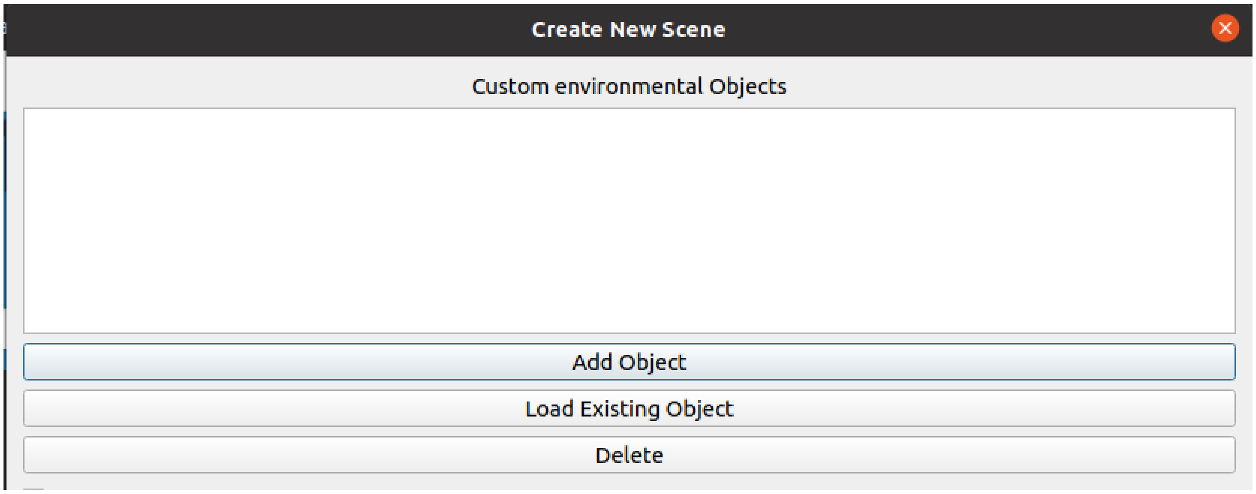
Click `confirm`

Next step: *Adding External joints for Objects*

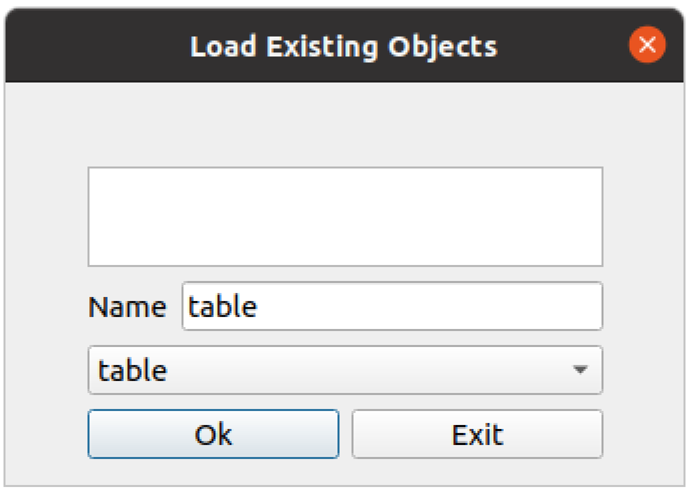
9.1.6 Loading an Object

This portion of the tutorial is to provide a guide on how to load the existing table object package that was previously generated by the workcell builder. If you have already created the table object in the same workcell builder session, skip this page and move on to: *Adding External joints for Objects*

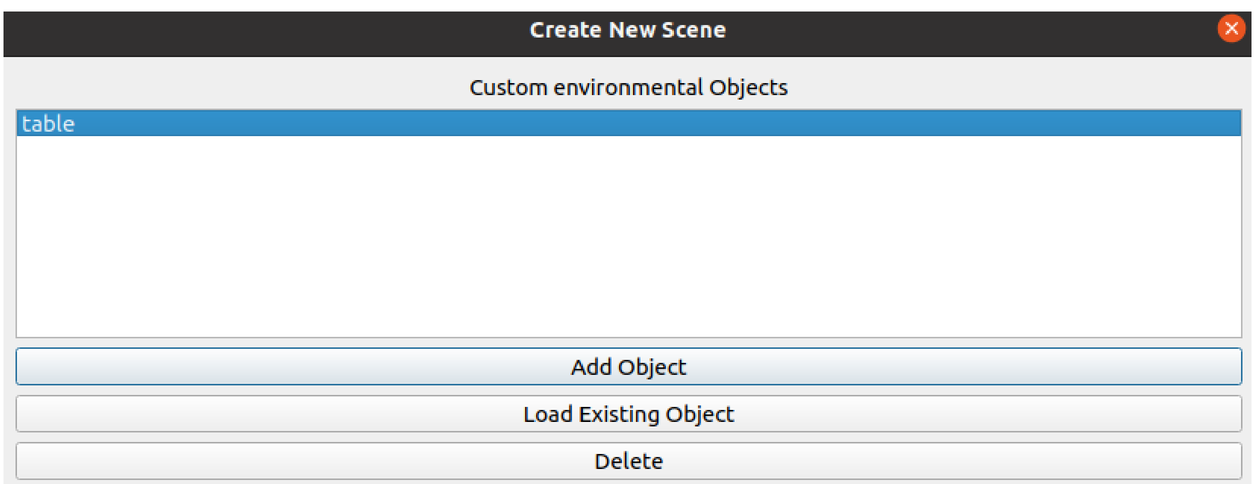
In the main scene window, click on the load object button



Select the table object package and leave the name as "table". Click the OK button.



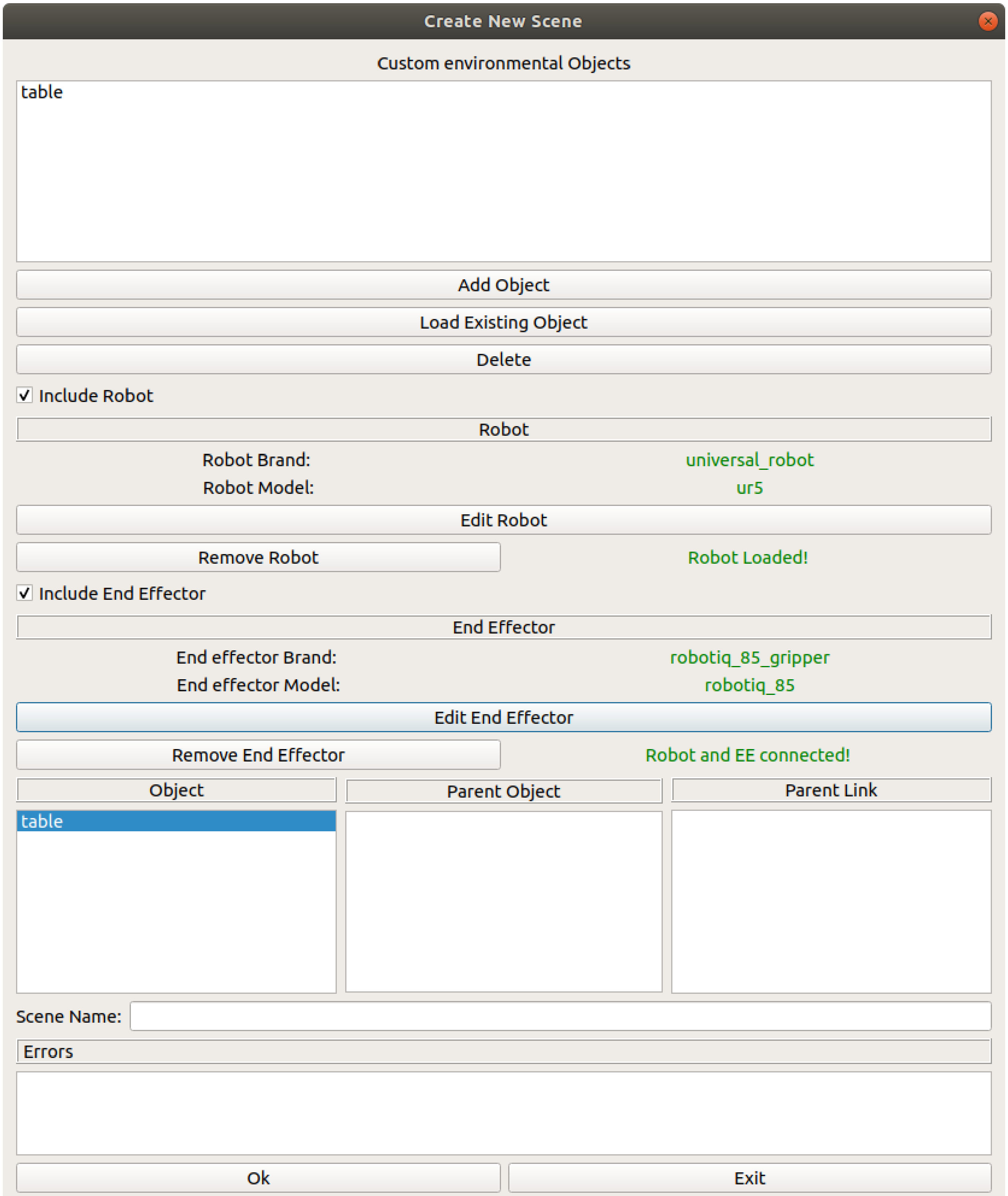
If the object is successfully loaded, your "Environment objects" field should be displayed as shown.



Next step: *Adding External joints for Objects*

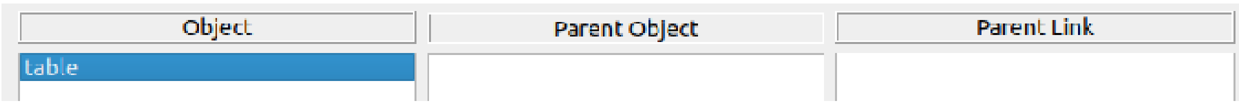
9.1.7 Adding External joints for Objects

Linking table to the external world

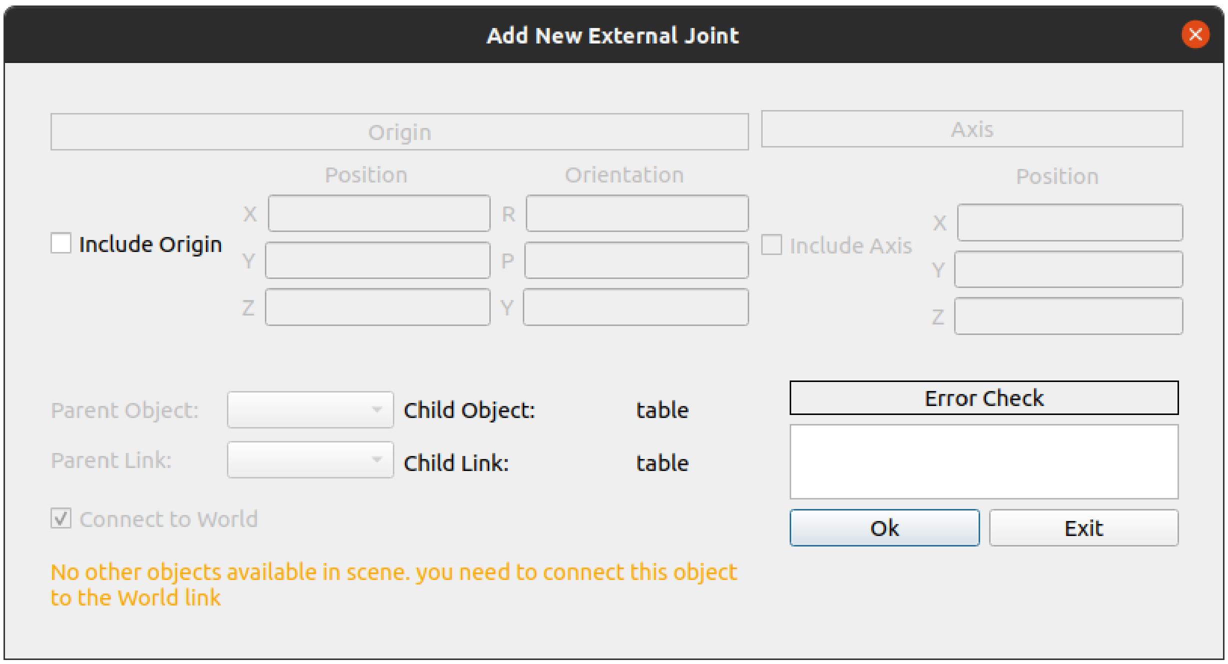


Awesome! You now have all the objects you require for the scene: A table, a manipulator, and an end effector. While the end effector and manipulator are automatically attached to the world (The manipulator is connected to `world` link and the end effector is connected to the manipulator's `ee_link` link), the table is currently not connected (As shown in the `parent link` and `child link` columns being empty)

To connect the table, we will create an external joint. Double click the `table` entry under the `Object` column (**Not** under the `custom objects` !)



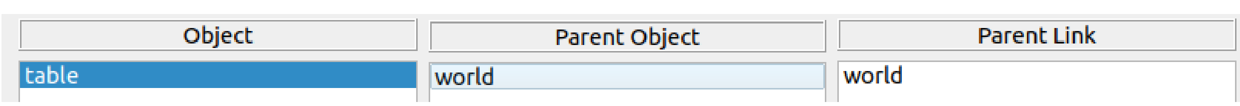
You should then see the `add new external joint` window pop up as shown below



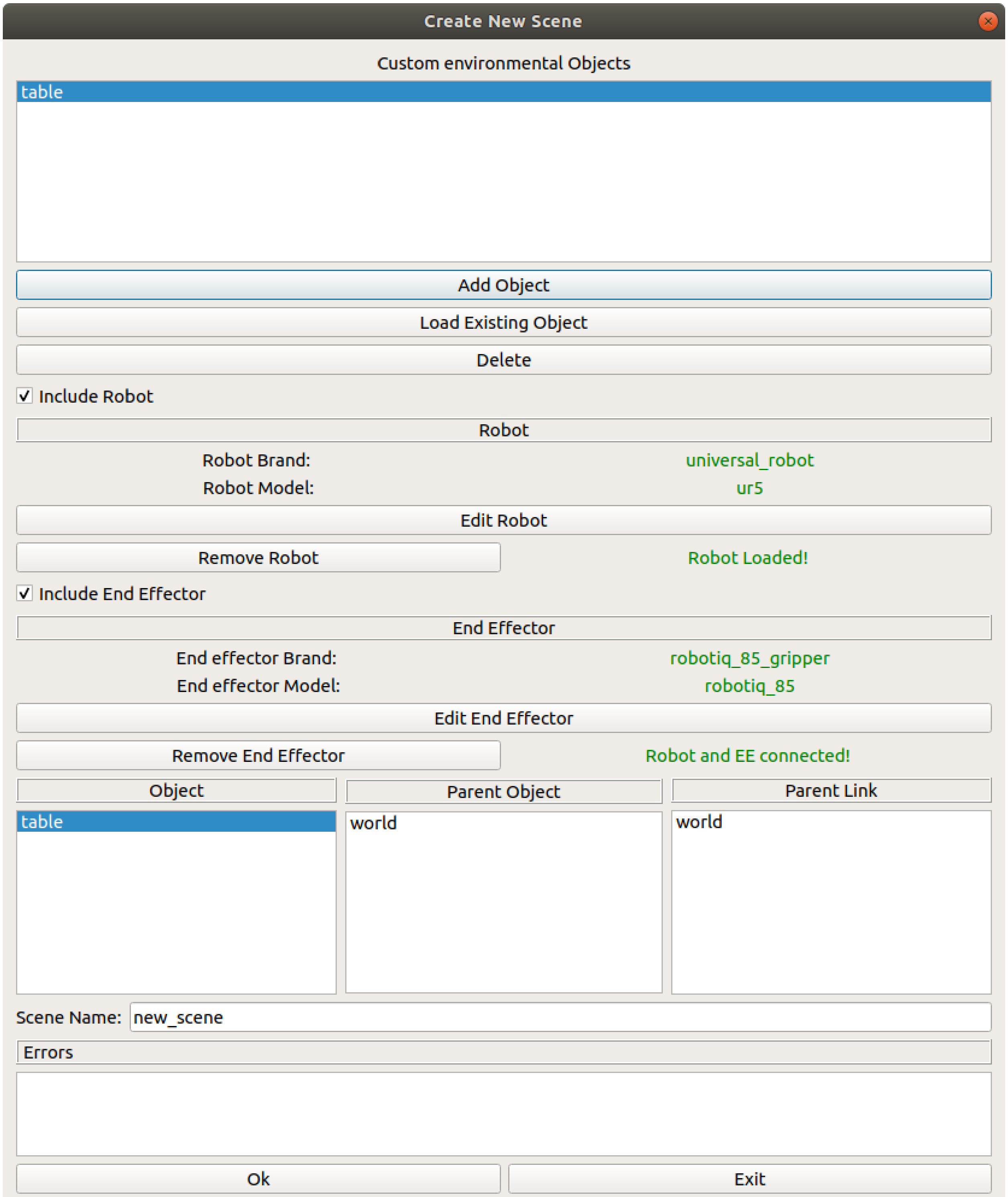
For now, environmental objects can only be attached to other environmental objects. Since the table is the only environmental objects in the scene, the only link you can connect to is `world`. If you have added more environment objects in scene, they will be displayed here.

For the table, since we want the table to be also connected to the origin of the world, where the base of the manipulator is located. Hence, we can just leave the origin checkbox unchecked (which defaults to xyz(0,0,0) and rpy(0,0,0)). You will then also see that the child link and child object are displayed there as well, based on the object name and the external joint child link you selected during object creation.

click `Ok`, and in the `Create New Scene` Window, you should now see the `Parent Object` and `Parent Link` Columns being filled with *world* . Your table is now successfully connected to the scene!



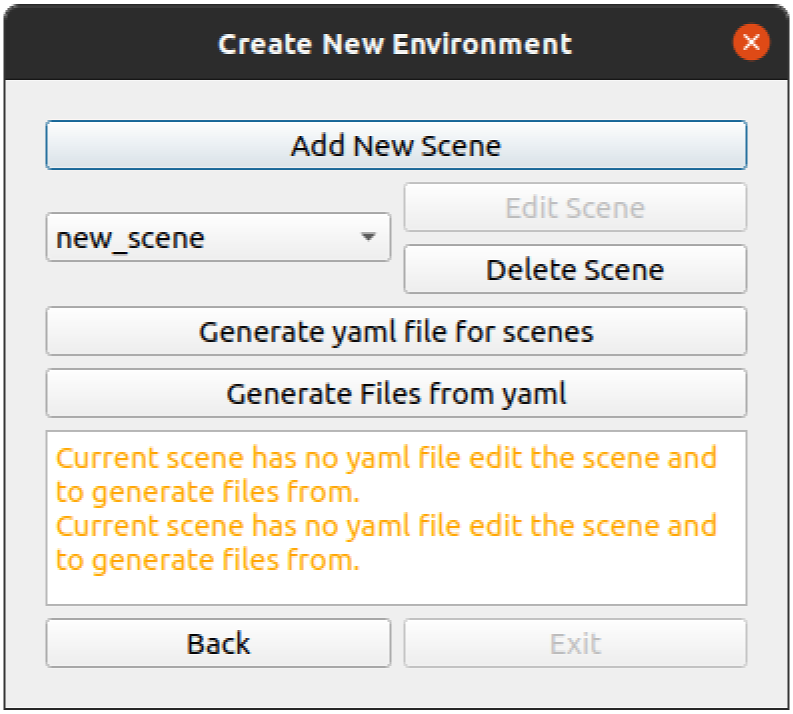
You are now officially done with creating your scene. Make sure to name your scene and then click `OK`.



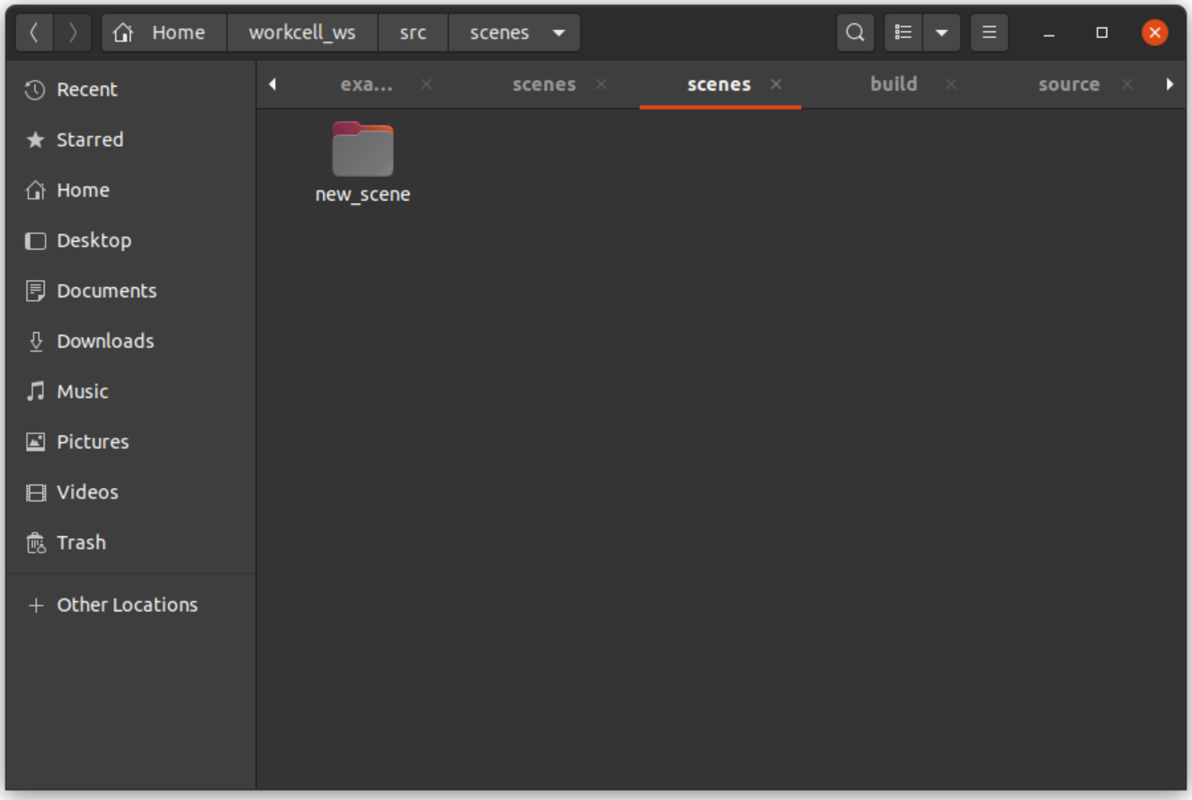
Next step: *Generating files and folders*

9.1.8 Generating files and folders

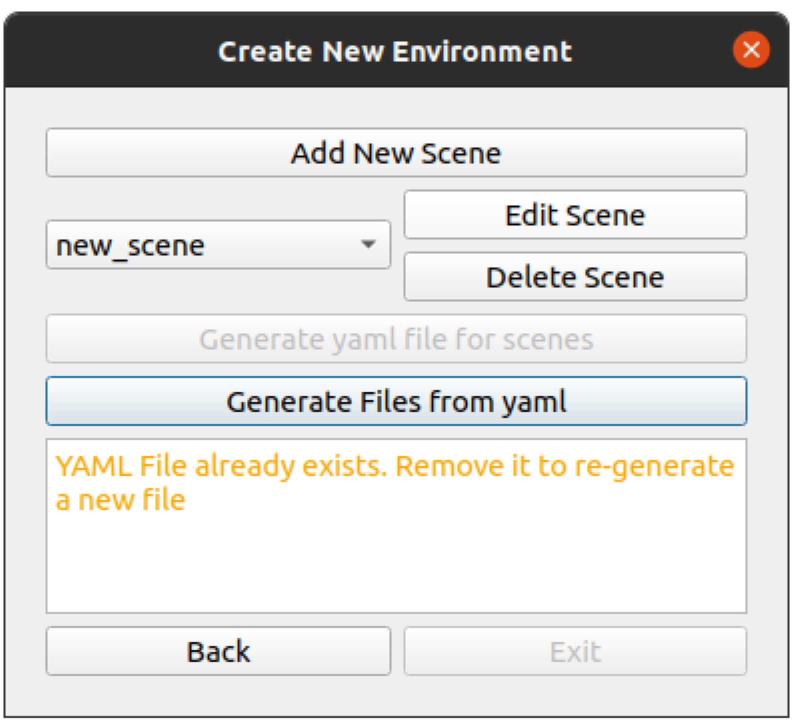
After creating the scene, you should see the name of your scene in the dropdown menu.



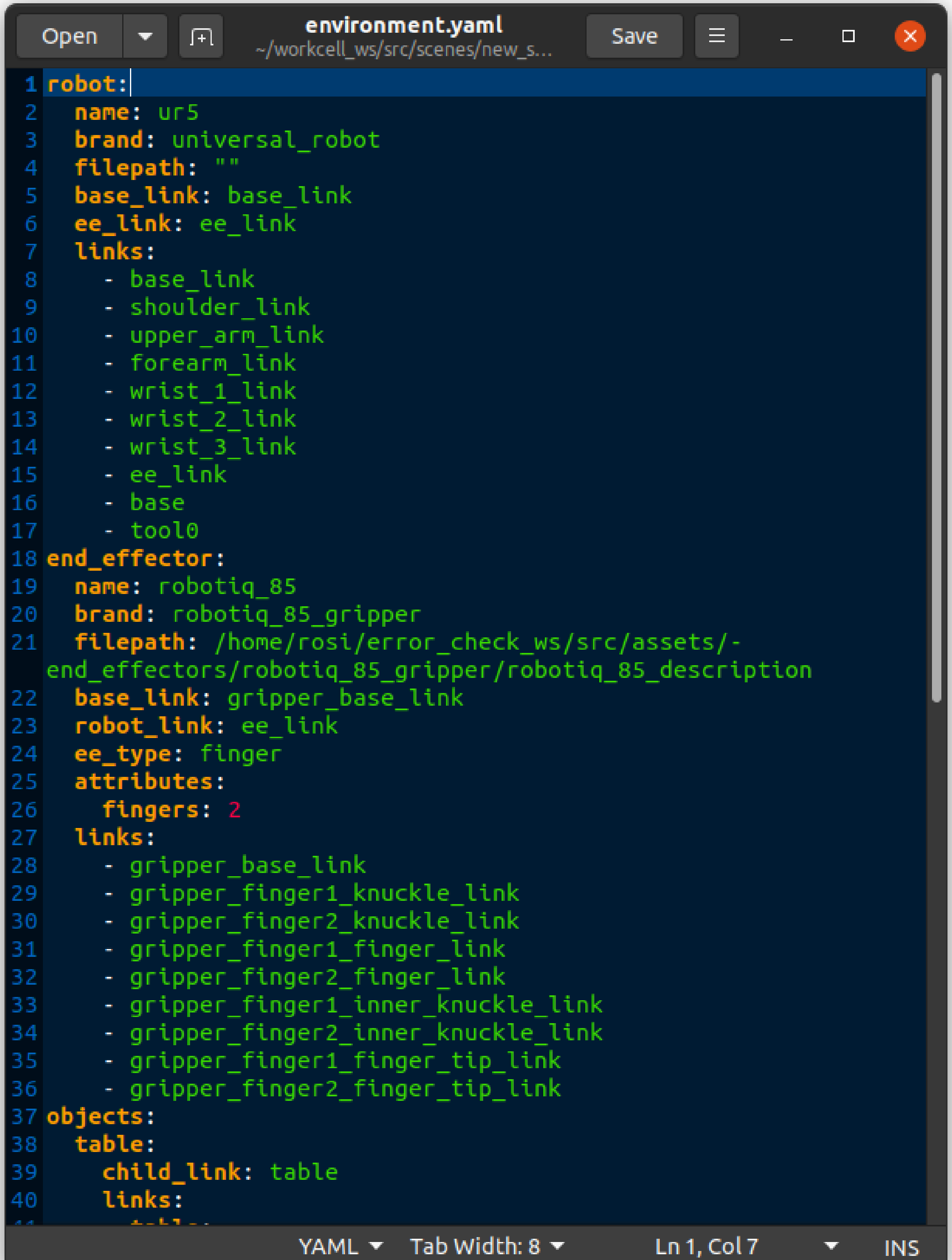
At this point, if you check your directory at `/workcell_ws/src/scenes/` , you should see your `new_scene` package being generated



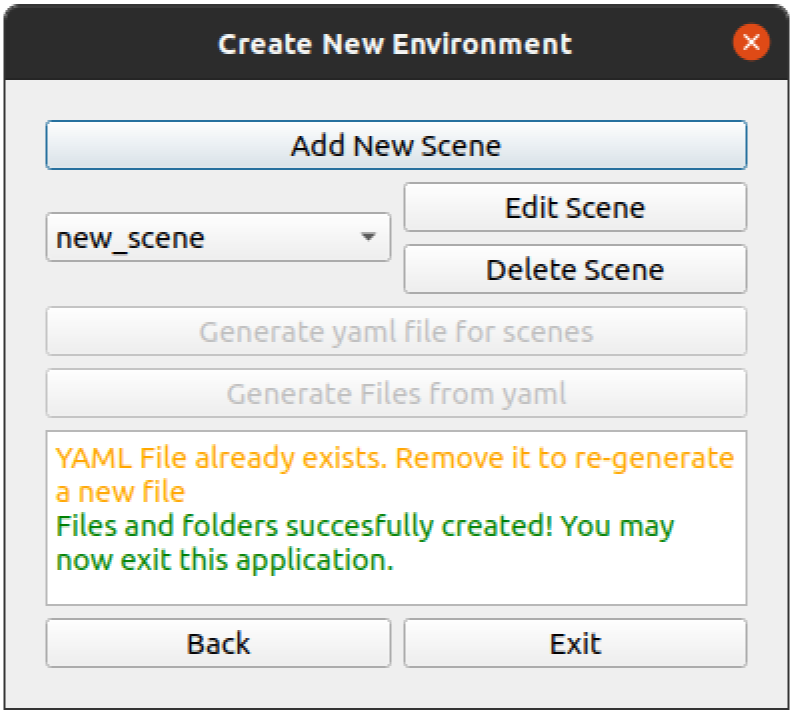
However, while the `CMakelists.txt` and `package.xml` files are generated, the rest of the other files are not. We need to first click on the `Generate Yaml file from scenes`. If successful, you should see the following message



To see what was the yaml that was generated, open the file at `/workcell_ws/src/scenes/new_scene/environment.yaml`



Next, we will need to generate the rest of the relevant files and folders for the scenes. Click on `Generate files from yaml` In the window. If successful, you will be prompted to exit the gui.



Click `exit`

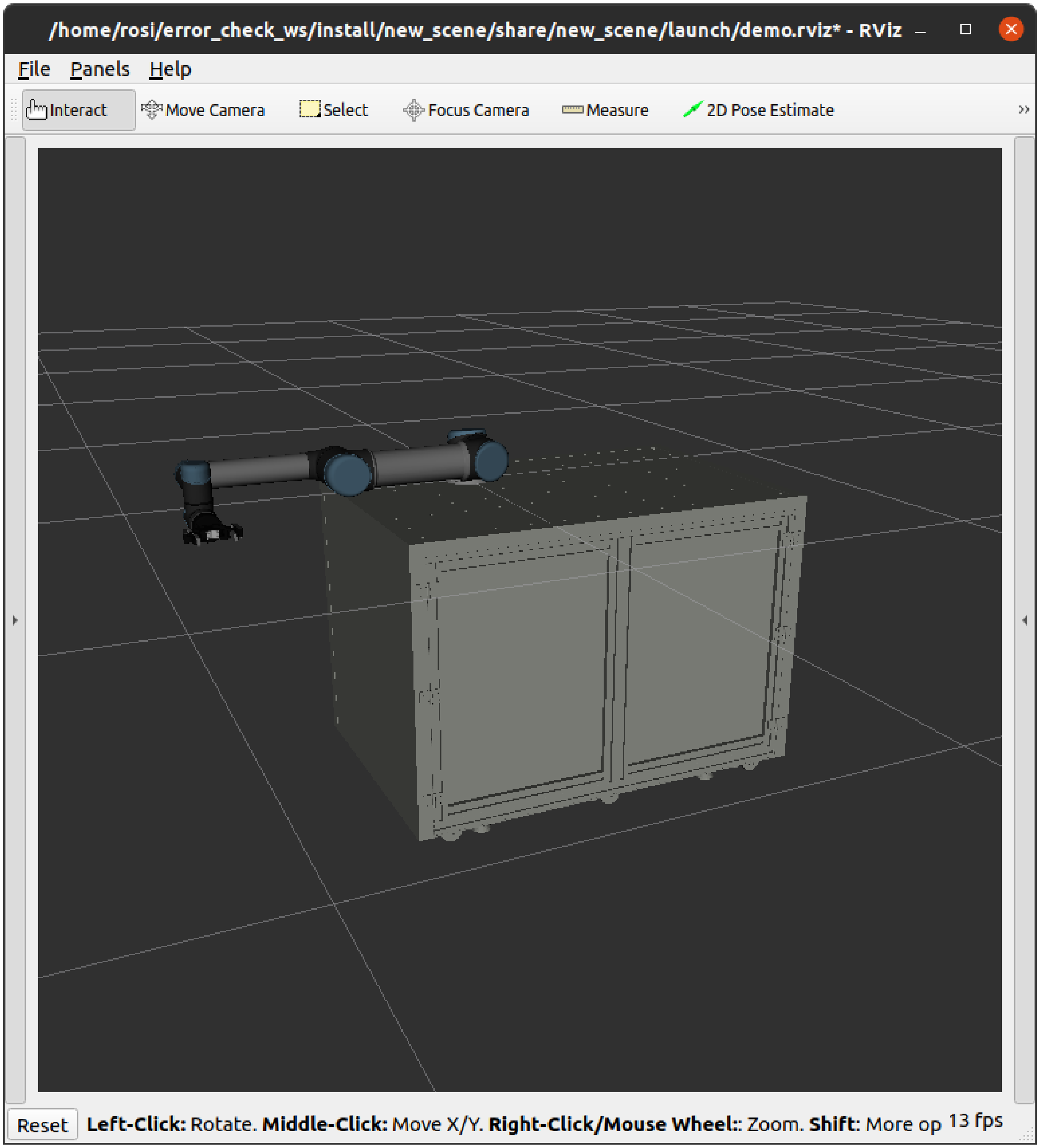
Next step: *Check if scene is properly generated*

9.1.9 Check if scene is properly generated

Next, to check if the scene is properly created, we will try running the package. In `/workcell_ws/` ,

```
source /opt/ros/foxy/setup.bash

cd ~/workcell_ws/src

colcon build

source install/setup.bash

ros2 launch new_scene demo.launch
```

RViz should be launched, and you should see your workspace in the simulation.



Now we have a simulated set up of the scene. However, typically a manipulation system would need some form of perception system, which will be addresed next: Next step: *Adding a camera to the scene*

9.1.10 Adding a camera to the scene

For manipulation systems with cameras, you would need to have a representation of the camera in the scene. The current workcell builder version **does not support** camera addition via the gui, but in this tutorial we will teach you how to add a camera to the scene.

For this example, we will adding the **Intel Realsense D415** depth camera in the scene.

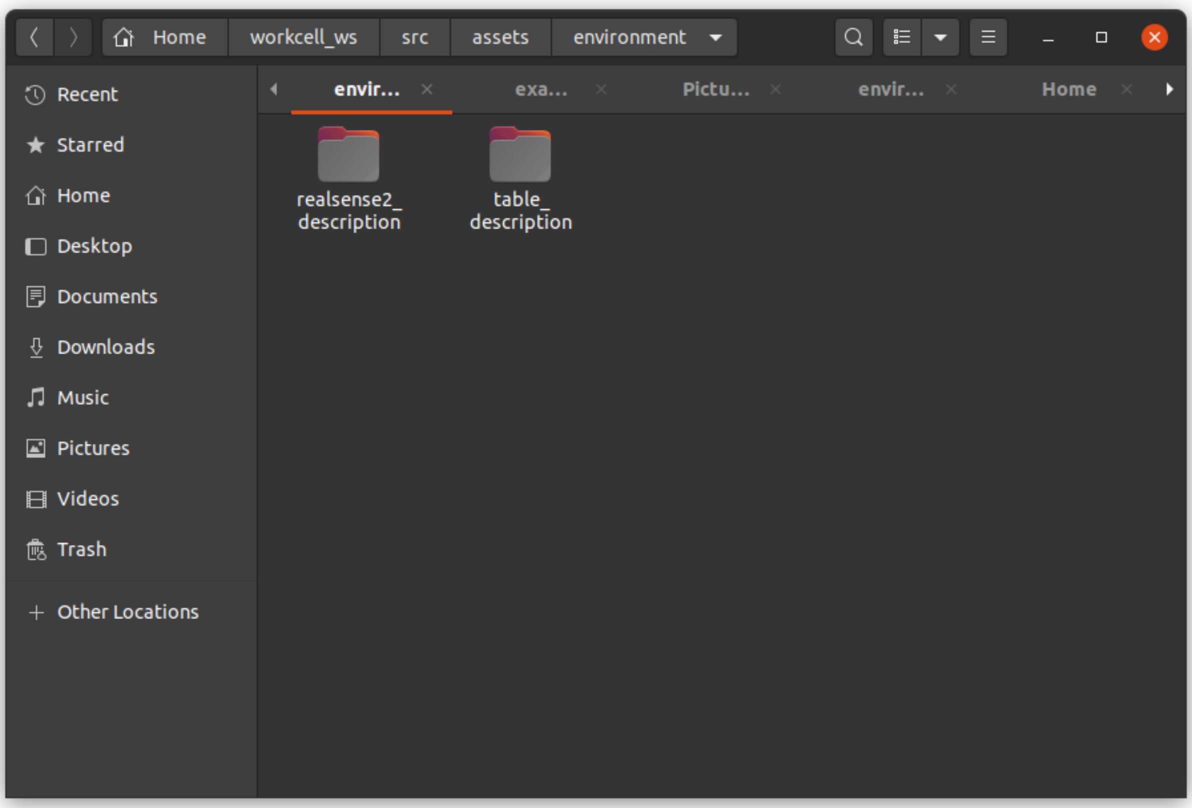
Downloading Camera Description Folder

Note that the default easy_manipulation_deployment package includes the intel realsense camera in the assets/environment/realsense2_description directory, but this portion provides a step by step guide on replicating it for other cameras. If you are planning to use what was provided, skip to the next step, "Add the camera to the scene"

In the directory `/workcell_ws/src/assets/environment/` , download the realsense repository

```
git clone https://github.com/IntelRealSense/realsense-ros.git -b foxy
```

Only keep the `realsense2_description` folder. Your `/workcell_ws/src/assets/environment/` folder should be as shown:



Next, build your package again to make sure that the realsense package builds correctly.

```
source /opt/ros/foxy/setup.bash

cd ~/workcell_ws
```

(continues on next page)

(continued from previous page)

```
colcon build

source install/setup.bash
```

Add the camera to the scene

Now we shall add the camera to the scene we created previously, `new_scene` .

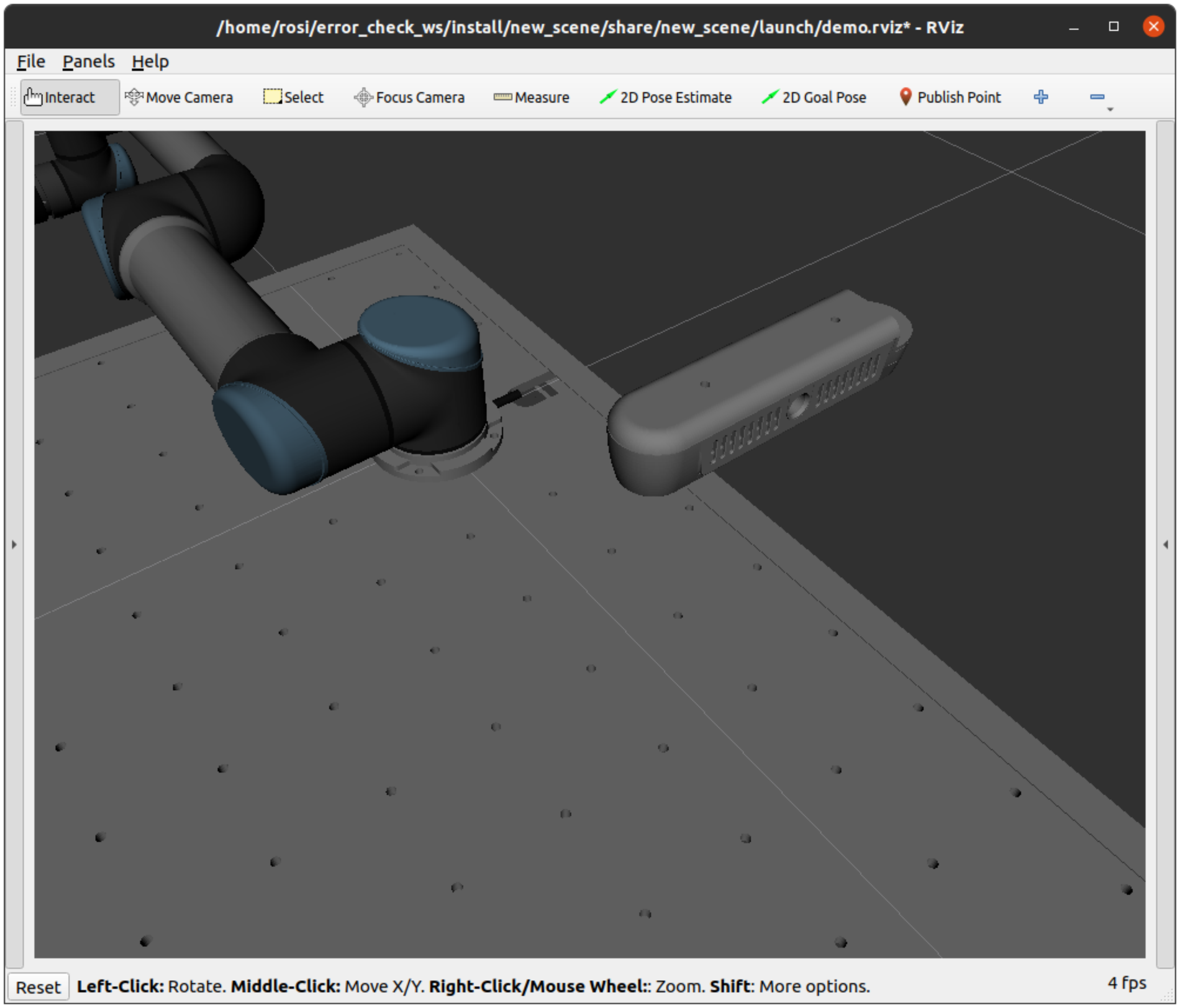
Open up the urdf file in `/workcell_ws/src/scenes/new_scene/urdf/scene.urdf.xacro` and add the following lines **before** the </robot> tag:

```
<xacro:include filename="$(find realsense2_description)/urdf/_d415.urdf.xacro"/>
   <xacro:arg name="use_nominal_extrinsics" default="true" />
   <xacro:sensor_d415 parent="table_" use_nominal_extrinsics="$(arg use_nominal_
↪extrinsics)">
    <origin xyz="-0.58 0.225 0.65" rpy="3.14159 1.57079506 0"/>
</xacro:sensor_d415>
```

Now, rebuild the package and launch the demo visualization

```
source /opt/ros/foxy/setup.bash

cd ~/workcell_ws/src

colcon build

source install/setup.bash

ros2 launch new_scene demo.launch.py
```

you should see the camera in scene.



Checking camera frame reference

First we need to check which link is the child link when connecting the camera to the xacro. This can be found in `/workcell_ws/src/assets/environment/realsense2_description/urdf/_d415.urdf.xacro`

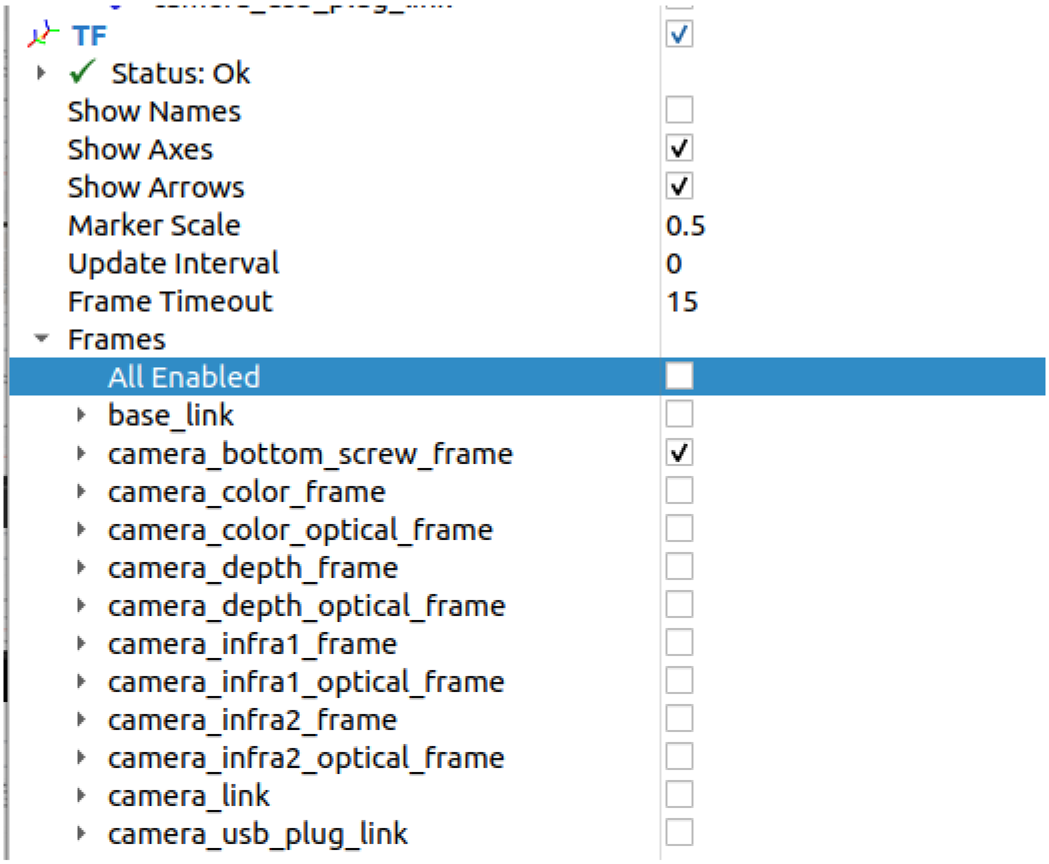
```
    <!-- camera body, with origin at bottom screw mount -->
    <joint name="${name}_joint" type="fixed">
      <xacro:insert_block name="origin" />
      <parent link="${parent}"/>
      <child link="${name}_bottom_screw_frame" />
    </joint>
```

From the URDF we can see that the link that is connected to the external scene is `${name}_bottom_screw_frame`.

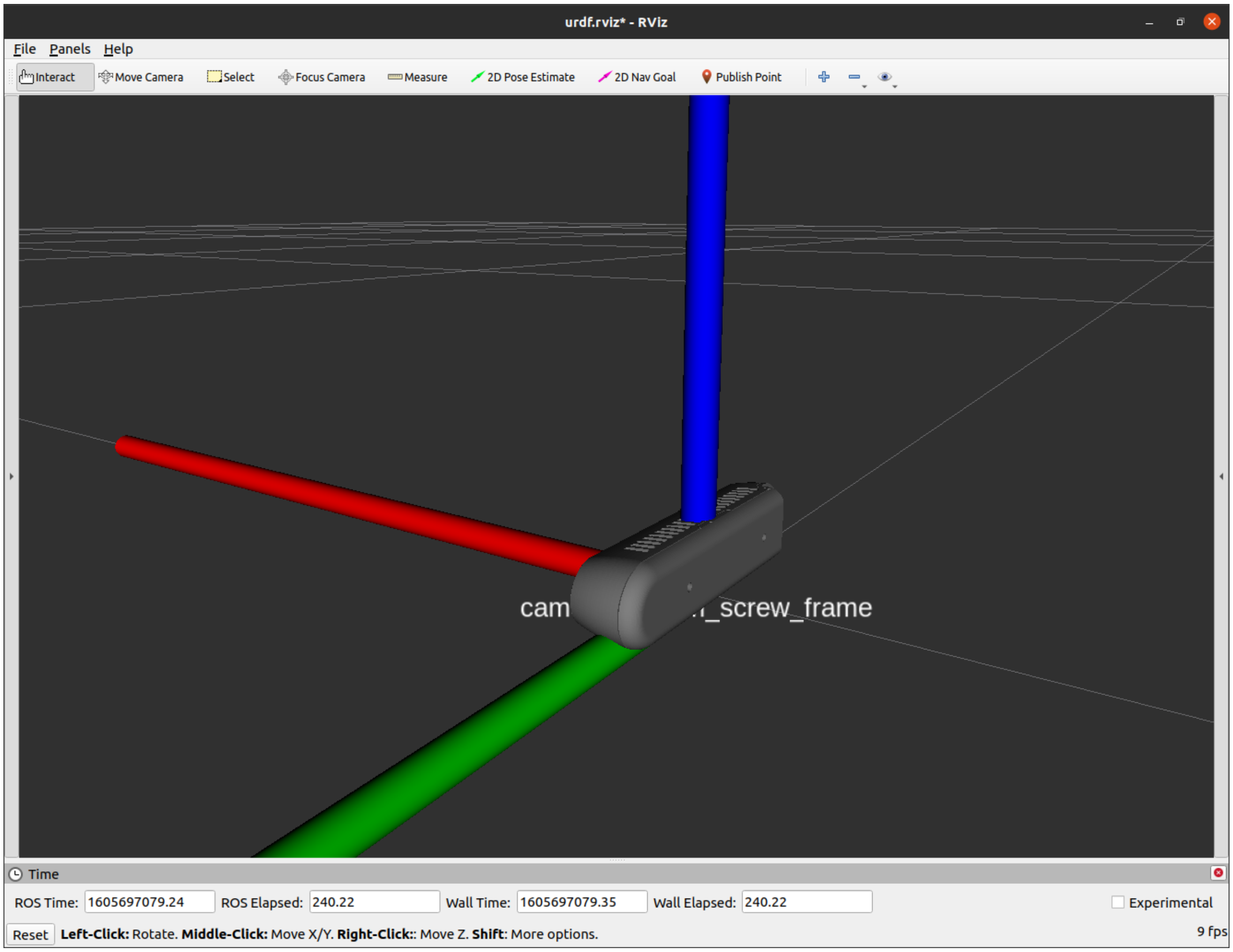
Next, We will launch RViz to check the orientation of this link.

```
ros2 launch realsense2_description view_model.launch.py model:=test_d415_camera.urdf.
↪xacro
```

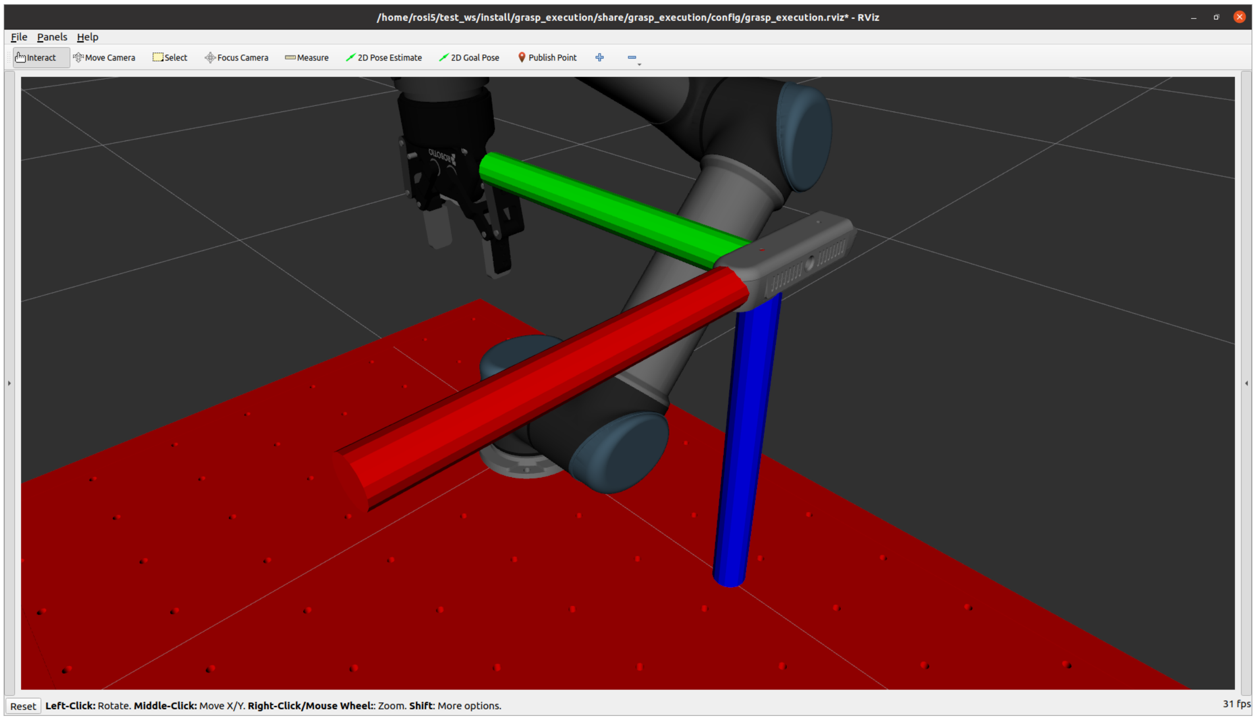
On the RViz GUI left panel, in order to see the frame, make sure to only check that link, and also increase the Marker Scale to about 0.5.



For some cameras, the link representing the model may not be in the same orientation as the actual camera frame the perception system references . This can be shown in RViz,



This is how we are currently referencing the camera in the scene. However, based off the perception system we are using (easy_perception_system), the actual camera frame is supposed to be as shown below.



To do so, we need to add a link in this orientation in the URDF. In the file `/workcell_ws/src/scenes/new_scene/urdf/scene.urdf.xacro` add the following lines under the declaration of the camera object:

```
<link name="camera_frame" />
<joint name="d415_to_camera" type="fixed">
    <parent link="camera_link"/>
    <child link="camera_frame"/>
    <origin xyz="0 0 0" rpy="1.57079506 0 1.57079506"/>
</joint>
```

This adds a new frame `camera_frame` that will be the frame in which the object is detected, and the frame that will be transformed to the world frame during the grasp execution phase of the pipeline.

Now that we have the main scene set up, we can move on to the grasp planner: *grasp_planner_example*

9.2 Grasp Execution Example

Currently the grasp execution portion of the package is under heavy development, but for now there is a basic example to execute the grasp plan.

In the example we will use the perception ROS2 bag as the input.

9.2.1 Grasp execution configuration

Grasp execution launch file

First we need to configure the grasp execution launch file. In `/workcell_ws/src/easy_manipulation_deployment/grasp_execution/example/launch/grasp_execution.launch.py`, make sure you define the correct scene package for the grasp execution, and the correct base link of the robot.

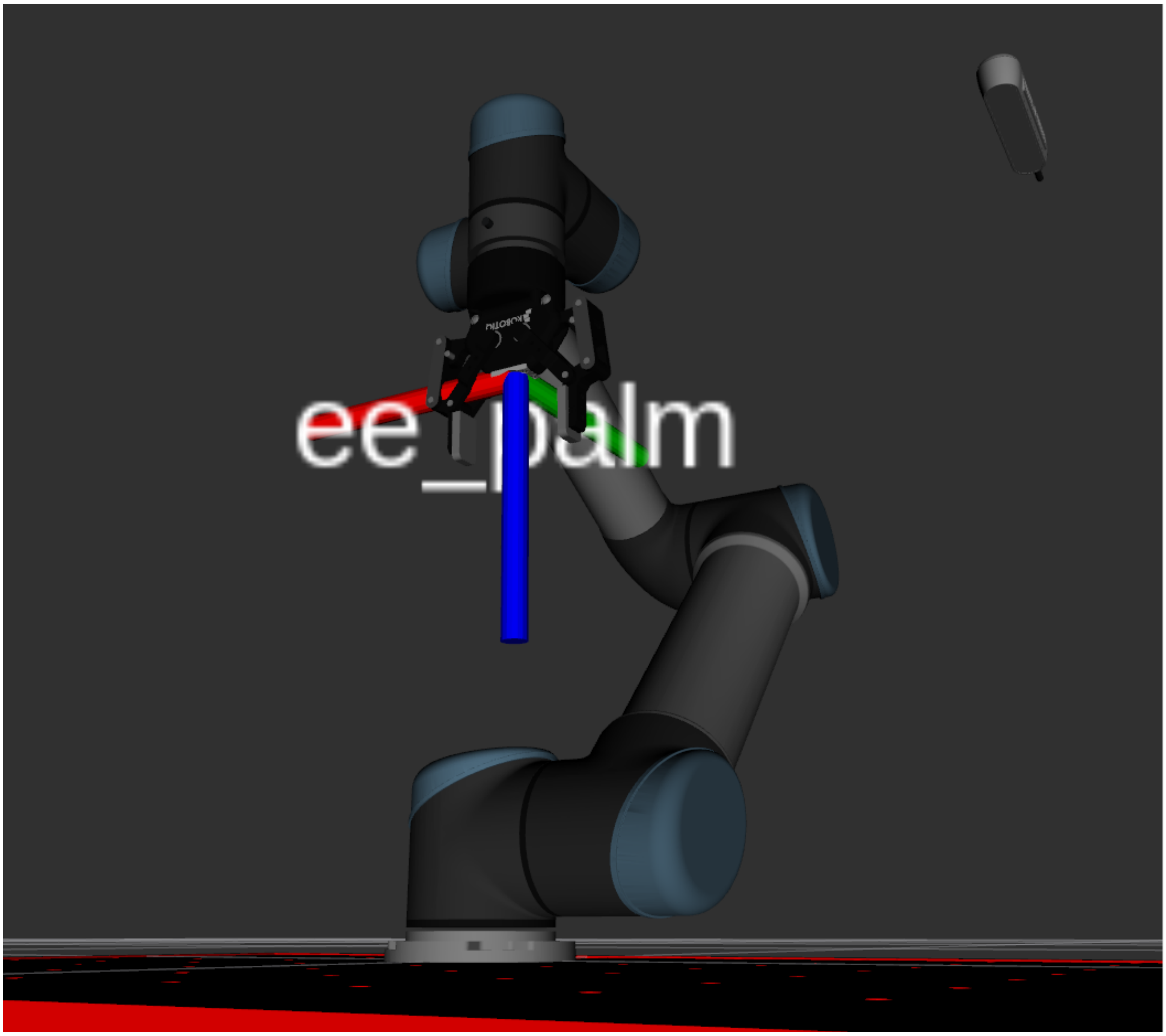
```
scene_pkg = 'new_scene'
robot_base_link = 'base_link'
```

Scene URDF File

In `/workcell_ws/src/scenes/new_scene/urdf/scene.urdf.xacro`, add the following lines *before* the `robot` tag:

```
<link name="ee_palm" />
<joint name="base_to_palm" type="fixed">
   <parent link="tool0"/>
   <child link="ee_palm"/>
   <origin xyz="0 0 0.09" rpy="0 0 0"/>
</joint>
```

This link, `ee_palm` represents the point of contact with respect to the grasp object.



Grasp execution node file

In /workcell_ws/src/easy_manipulation_deployment/grasp_execution/example/config/workcell_context.yaml you can edit the end_effectors.link parameter to reflect the link of the end effector that will represent the point of contact with respect to the grasp object. In this case, it will be ee_palm.

The link group name for the manipulator can also be defined by the group_name parameter. In this case, it will be manipulator.

9.2.2 Running full pipeline

After making the changes, remember to build the workspace. In `/workcell`,

```
colcon build
source install/setup.bash
```

The next part requires three different terminals.

Terminal 1: Grasp Execution

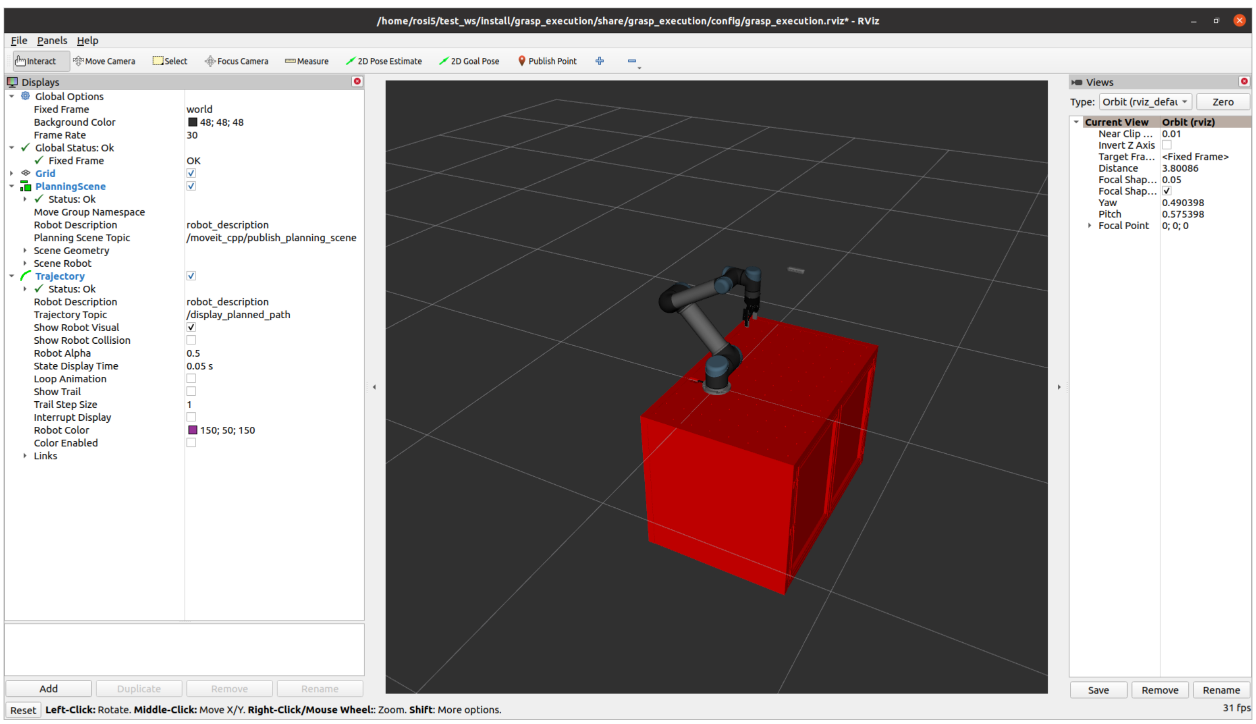
This terminal runs the manipulation workspace simulation. First, source all relevant repositories. In `/workcell`,

```
source /opt/ros/foxy/setup.bash
source ~/moveit2_ws/install/setup.bash
source install/setup.bash
```

Next, launch the grasp execution component.

```
ros2 launch grasp_execution grasp_execution_launch.py
```

You should then see rviz launch and the scene.



Terminal 2: Grasp Planner

This terminal runs the grasp_planner. First, source all relevant repositories. In /workcell,

```
source /opt/ros/foxy/setup.bash
source ~/moveit2_ws/install/setup.bash
source install/setup.bash
```

Next, launch the grasp planner.

```
ros2 run grasp_planning grasp_planning_node
```

You should then see the following

```
[easy_manipulation_deployment][Grasp Planner] Waiting for topic....
```

Terminal 3: Perception example rosbag

This terminal runs the perception example. First, source all relevant repositories. In /workcell,

```
source /opt/ros/foxy/setup.bash
source ~/moveit2_ws/install/setup.bash
source install/setup.bash
```

Next, run the rosbag

```
ros2 bag play src/easy_manipulation_deployment/grasp_planner/rosbag/perception_example/
↪rosbag/rosbag2_2020_09_25-15_54_55_0.db3
```

You should then see the following

```
[INFO] [1605754174.300681975] [rosbag2_storage]: Opened database 'src/easy_manipulation_
↪deployment/grasp_planner/rosbag/perception_example/rosbag/rosbag2_2020_09_25-15_54_55_
↪0.db3' for READ_ONLY.
```

Ideally, if all components run in sequence, you should then see the manipulator simulation move in Rviz. The object will be picked up and placed at a drop off location before going back to the home position.

9.3 Grasp Planner Example

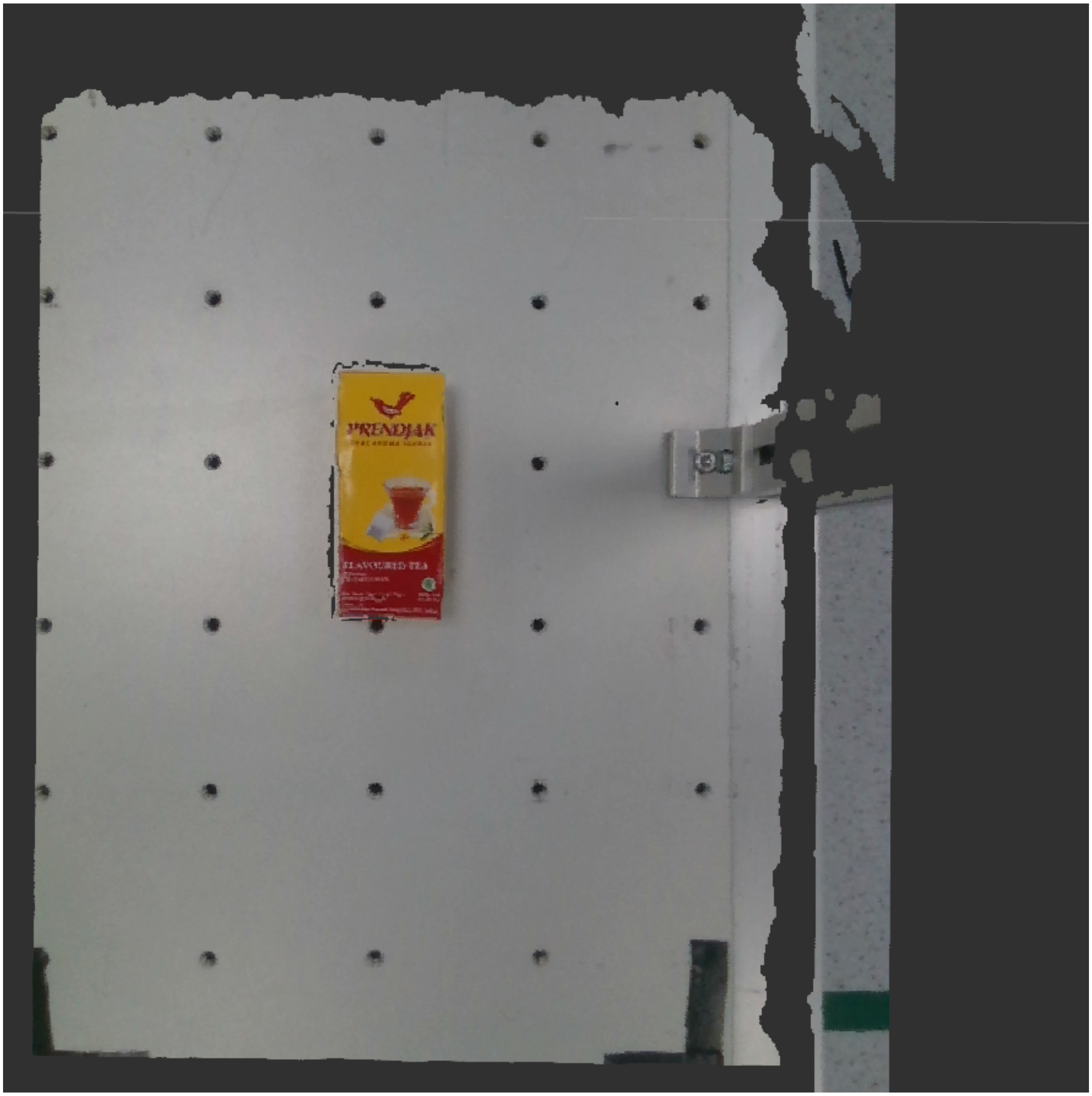
In this part of the tutorial we will reference the scene we have generated in :ref:'workcell_builder_example'

Currently, perception data inputs for the Grasp Planner only works with:

1. Pointcloud2
2. Easy Perception Deployment (EPD)

If you currently do not have a working perception system, you can still test out the package using either the epd_rosbag or pointcloud_rosbag located in *PATH TO ROSBAG folder TO BE WRITTEN*

The rosbags are using the stream of a simple tea box as shown below.



Note: Prority is given to Easy Perception Deployment topic if both Pointcloud and Easy Perception Deployment are running simultaneously.

9.3.1 Set up end effector parameters

The current version of Grasp Planner is able support end-effectors for both multiple suction arrays and multiple fingered grippers.

For the example, we will utilize the **2-Finger gripper** in line with the end-Effector used for the scene in *Workcell Builder Example*

The configuration files need to be set according to the type of End-effector that is being used. In the configuration file found in `/grasp_planner/example/config/params_2f.yaml`, the contents of the `.yaml` file should be as followed:

2-Finger gripper

```yaml
grasp_planning_node:
  ros__parameters:
    perception_topic: "/camera/pointcloud"
    camera_frame: "camera_color_optical_frame"
    point_cloud_params:
      passthrough_filter_limits_x: [-0.50, 0.50]
      passthrough_filter_limits_y: [-0.15, 0.40]
      passthrough_filter_limits_z: [0.01, 0.70]
      segmentation_max_iterations: 50
      segmentation_distance_threshold: 0.01
      cluster_tolerance: 0.01
      min_cluster_size: 750
      cloud_normal_radius: 0.03
    end_effectors:
      end_effector_names: [robotiq_2f]
      robotiq_2f:
          type: finger
          num_fingers_side_1: 1
          num_fingers_side_2: 1
          distance_between_fingers_1: 0.0
          distance_between_fingers_2: 0.0
          finger_thickness: 0.02
          gripper_stroke: 0.085
          grasp_planning_params:
            grasp_plane_dist_limit: 0.007
            voxel_size: 0.01
            grasp_rank_weight_1: 1.5
            grasp_rank_weight_2: 1.0
            world_x_angle_threshold: 0.5
            world_y_angle_threshold: 0.5
            world_z_angle_threshold: 0.25
```

Tip: For more indepth information on how to configure the `.yaml` file for your own end-effector. Head on over to *Grasp Planner Configuration File*

9.3.2 Running the Grasp Planner

This part of the example requires 2 terminals. We will be running the epd_rosbag for this example.

In terminal 1: (Grasp Planner Terminal)

```
source /opt/ros/foxy/setup.bash

source PATH_TO_MOVEIT_WS/install/setup.bash

cd PATH_TO_EMD_WS/

colcon build

source install/setup.bash

ros2 launch grasp_planner grasp_planner_launch.py
```

The package will then show the following when waiting for the perception topic

```
[pcl_test_node-1] waiting...
```

- A blank Cloud Viewer window will pop up

Proceed to run the perception topic

Note: Take note that Grasp Execution should be launched first as the Grasp Planner requires the frame `camera_color_optical_frame` to be present. If not the following will be shown on Terminal 1:

```
[pcl_test_node-1] [INFO] [1617252094.561454528] [pcl_node]: Message Filter dropping␣
↪message: frame 'camera_color_optical_frame' at time 0.000 for reason 'Unknown'
```

In terminal 2: (Rosbag/Perception stream Terminal)

Note: This step uses the epd_rosbag as an example, you can provide your own stream of pointcloud/EPD data or use the camera_rosbag(uses /pointcloud topic), found in the rosbag folder as well.

Tip: More information on acceptable message types can be found in *Grasp Planner Output Message Types*

epd_rosbag

```
source /opt/ros/foxy/setup.bash

cd ~/workcell_ws/

source install/setup.bash
```

(continues on next page)

(continued from previous page)

```
cd PATH_TO_CAMERA/EPD_ROSBAG

ros2 bag play epd_rosbag.db3
```

Once successfully launched, the output should be as shown below on Terminal 2.

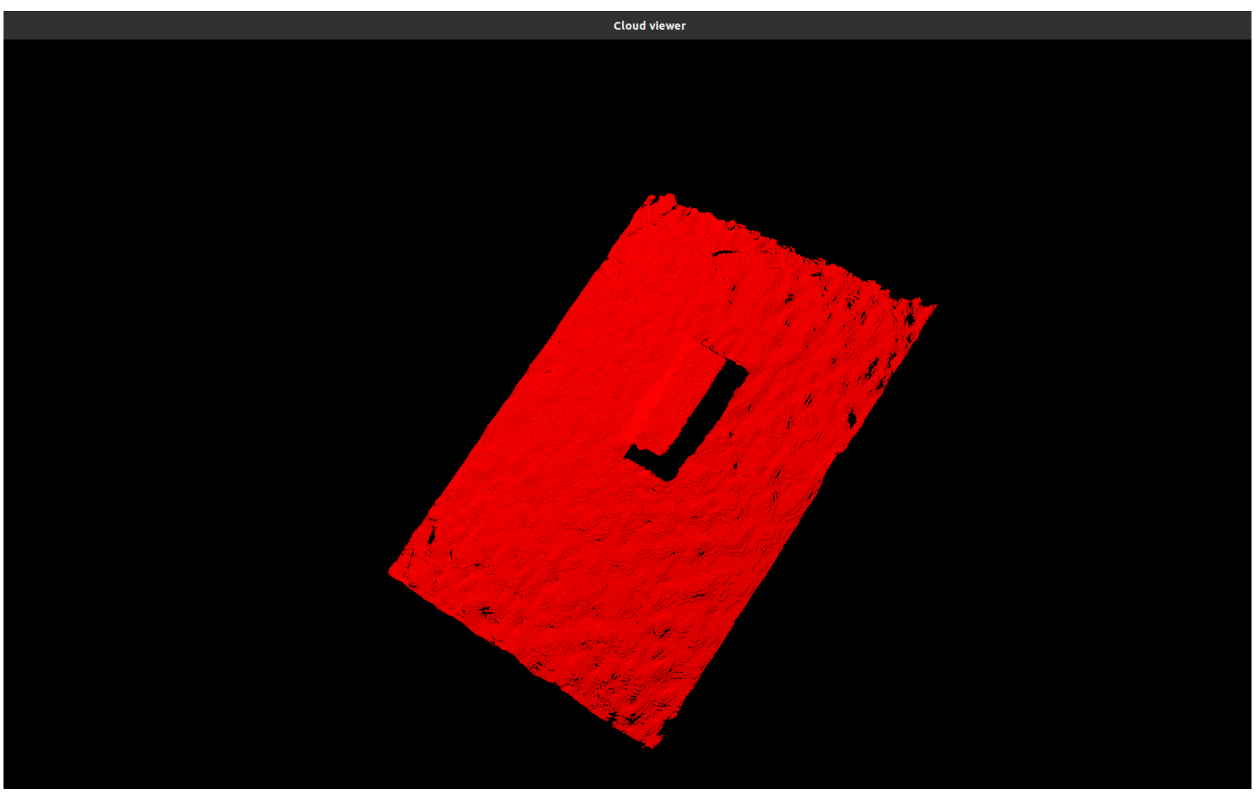
```
[INFO] [1617251978.247342106] [rosbag2_storage]: Opened database 'epd_rosbag.db3' for READ_ONLY.
```

9.3.3 Viewing grasping results on Cloud viewer

- 1. Proceed to click on the Cloud Viewer window and it will show the pointcloud and bounding box of the object (Use the mouse scroll to view the pointclouds better).
- 2. Press the `Q` key within the Cloud Viewer window to view the results of the grasp_samples
- 3. The terminal running grasp_planner_launch.py will show the ranks of all ranked grasps on the object and the total number of grasps that can be sampled for the object.
- 4. First grasp visualized on the viewer is the best grasp.
- 5. Pressing `Q` will show the rest of the consecutively ranked grasps.
- 6. Once all the grasps have been screened through, the grasp_planner will publish the /grasp_tasks topic.

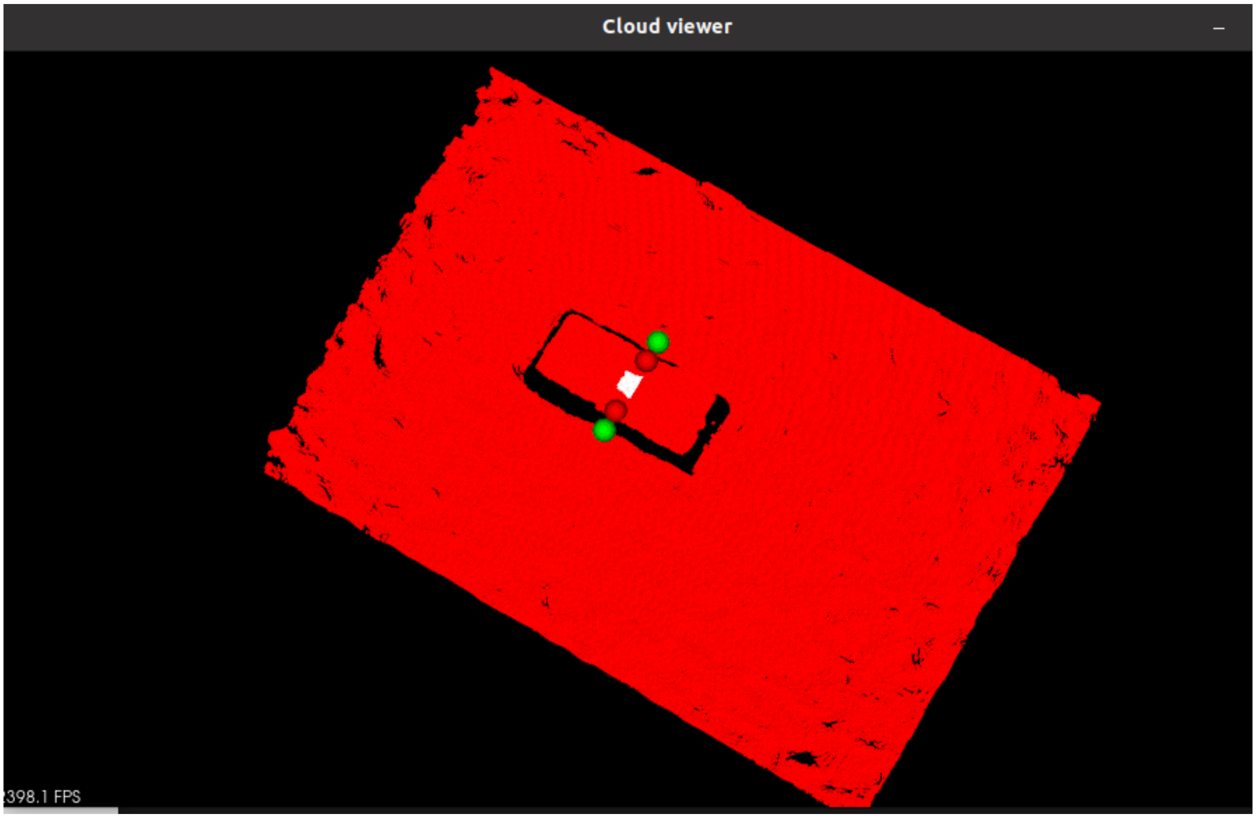
The Cloud Viewer window will then load the frame of the perception input data as shown below:

Pointcloud data



Object Bounding Box

Grasp visualization



The grasps are ranked based off the quality of their grasps. The pose and orientation of the top ranked grasp will then be published for *Grasp Execution Example*