Tesseract

Release 3cd485a843e521cb2c2344110d332ea69405f375

unknown

Mar 07, 2023

INTRO

The new planning framework (Tesseract) was designed to be lightweight, limiting the number of dependencies, mainly to only used standard library, eigen, boost, orocos and to the core packages below are ROS agnostic and have full python support.

CHAPTER

ONE

SECTIONS

1.1 Getting Started

Note: Instructions are intended for installation on the Ubuntu 20.04 Focal operating system.

Before installing Tesseract and it's dependencies, make sure you have the most up to date packages:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

1.1.1 Installing ROS Noetic (Optional)

The Robot Operating System (ROS) is an open source set of software libraries and tools that help you build robot applications. The *tesseract*, *tesseract_planning*, and *tesseract_python* repositories are all ROS-agnostic. The *tesseract_ros* and *tesseract_ros2* repositories contain tools that are specific to ROS and make integrating Tesseract and ROS extremely easy.

To install the latest version of ROS (Noetic) follow the instructions from the ROS website.

Note: It's easy to miss steps when going through the ROS installation tutorial. If you run into errors in the next few steps, a good place to start is to go back and make sure you have installed ROS correctly.

1.1.2 Installing Tools

Catkin

Catkin is the official build system for ROS. Catkin can also be used to build ROS-agnostic packages containing CMakeLists.txt and package.xml files.

You can install Catkin with the following command:

```
sudo apt-get install ros-noetic-catkin python3-catkin-tools
```

ROSDep (Optional)

ROSDep is the official ROS command line tool for installing system dependencies.

You can install ROSDep with the following commands:

```
sudo apt-get install python3-rosdep
sudo rosdep init
rosdep update
```

WSTool

WSTool is the official ROS workspace management tool.

You can install WSTool with the following command:

```
sudo apt-get install python3-wstool
```

1.1.3 Creating a Workspace

Create a workspace (in this case we'll name it *tesseract_ws*):

```
mkdir -p ~/tesseract_ws/src
```

1.1.4 Cloning Tesseract Repositories

Move to the source directory:

```
cd ~/tesseract_ws/src
```

Clone Tesseract repository into your workspace:

```
git clone https://github.com/tesseract-robotics/tesseract
```

Clone Tesseract Planning repository into your workspace:

```
git clone https://github.com/tesseract-robotics/tesseract_planning
```

Clone Tesseract Qt repository into your workspace:

```
git clone https://github.com/tesseract-robotics/tesseract_qt
```

If you are using the Robot Operating System (ROS), you can also clone the Tesseract ROS repository:

```
git clone https://github.com/tesseract-robotics/tesseract_ros
```

1.1.5 Installing Dependencies

Cloning Source Dependencies with WSTool

Clone the repositories in the dependencies.rosinstall file using wstool:

```
wstool init ~/tesseract_ws/src/ ~/tesseract_ws/src/tesseract_planning/dependencies.
↪rosinstall
```

Installing Debian Dependencies with ROSDep (Optional)

Run the following command to automatically install all debian dependencies listed in each package.xml file:

```
rosdep install -y --from-paths ~/tesseract_ws/src --ignore-src --rosdistro noetic
```

Note: If you don't use the ROSDep tool you will need to manually install (via *apt-get*) each debian dependency.

1.1.6 Building Your Workspace

Build your workspace using catkin tools:

```
cd ~/tesseract_ws/
source /opt/ros/noetic/setup.bash
catkin build
```

Source the catkin workspace:

```
source ~/tesseract_ws/devel/setup.bash
```

Note: To build with Clang-Tidy enabled you must pass the *-DTESSERACT_ENABLE_CLANG_TIDY=ON* to cmake when building. This is automatically enabled if cmake argument *-DTESSERACT_ENABLE_TESTING_ALL=ON* is passed.

1.2 URDF

Tesseract is compatible with URDFs (Universal Robot Description Format), the native format for describing robots in ROS. In this tutorial, you will find resources for the URDF and important tips.

Attention: Tesseract has extended capabilities for URDFs. For more information, see Tesseract URDF.

1.2.1 URDF Resources

- URDF ROS Wiki Page - The URDF ROS Wiki page is the source of most information about the URDF.
- URDF Tutorials - Tutorials for working with the URDF.
- SOLIDWORKS URDF Plugin - A plugin that lets you generate a URDF directly from a SOLIDWORKS model.

1.2.2 Important Tips

This section contains a set of tips on making sure that the URDF that you generate can be used with Tesseract. Make sure you go through all these tips before starting to use Tesseract with your robot.

Special Characters in Joint Names

Joint names should not contain any of the following special characters: -, [,], (,)

Safety Limits

Some URDFs have safety limits set in addition to the joint limits of the robot. Here's an example of the safety controller specified for the Panda robot head pan joint:

```
<safety_controller k_position="100" k_velocity="1.5" soft_lower_limit="-2.857" soft_
↪upper_limit="2.857"/>
```

The "soft_lower_limit" field and the "soft_upper_limit" field specify the joint position limits for this joint. Tesseract will compare these limits to the hard limits for the joint specified in the URDF and choose the limits that are more conservative.

Note: If the "soft_lower_limit" and the "soft_upper_limit" in the safety_controller are set to 0.0, your joint will be unable to move. Tesseract relies on you to specify the correct robot model.

Collision Checking

Tesseract uses the meshes specified in the URDF for collision checking. The URDF allows you to specify two sets of meshes separately for visualization and collision checking. In general, the visualization meshes can be detailed and pretty, but the collision meshes should be much less detailed. The number of triangles in a mesh affects the amount of time it takes to collision check a robot link. The number of triangles in the whole robot should be on the order of a few thousand.

Test your URDF

It is very important to test your URDF out and make sure things are okay. The ROS URDF packages provide a check_urdf tool. To verify your URDF using the check_urdf tool, follow the instructions here.

1.2.3 URDF Examples

There are lots of URDFs available for robots using ROS.

- URDF Examples - A list of URDFs from the ROS community.

1.3 SRDF

The SRDF or Semantic Robot Description Format complement the URDF and specifies joint groups, default robot configurations, additional collision checking information, and additional transforms that may be needed to completely specify the robot's pose. The recommended way to generate a SRDF is using the Tesseract Setup Assistant.

1.3.1 Groups

Groups (also referred to as kinematics groups) define collections of links and joints that are used for planning. Groups can be specified in several ways:

Collection of Joints

A group can be specified as a collection of joints. All the child links of each joint are automatically included in the group.

Serial Chain

A serial chain is specified using the base link and the tip link. The tip link in a chain is the child link of the last joint in the chain. The base link in a chain is the parent link for the first joint in the chain.

1.3.2 Group States

The SRDF can also store fixed configurations of the robot. A typical example of the SRDF in this case is in defining a HOME position for a manipulator. The configuration is stored with a string id, which can be used to recover the configuration later.

1.3.3 Group Tool Center Points

Store fixed tool center point definitions by string id, which can be used to recover the tool center point during operation like planning.

1.3.4 Add OPW Inverse Kinematics Solver

Add and assign a OPW inverse kinematics solver to an already defined group.

1.3.5 Add ROP Inverse Kinematics Solver

Add and assign a Robot on Positioner (ROP) inverse kinematics solver to an already defined group. This assumes a custom invserse kinematics solver already exists for the robot and the positioner is to be sample at some resolution to find a larger set of inverse kinematic solutions.

1.3.6 Add REP Inverse Kinematics Solver

Add and assign a Robot with External Positioner (REP) inverse kinematics solver to an already defined group. This assumes a custom invserse kinematics solver already exists for the robot and the positioner is to be sample at some resolution to find a larger set of inverse kinematic solutions.

1.3.7 Define Collision Margin Data

In most industrial applications a single contact margin distance is not sutable because there are objects that constantly work within close proximity. This would limit the contact distance to be smaller than desired for other links allowing all objects to operate close to one another. The Tesseract contact checkers allow for a default margin to be defined along with link pair collision margins elimating this issue. This is configurable from within the SRDF file shown below.

1.3.8 Allowed Collision Matrix

Define link pairs that are allowed to be in collision with each other. This is used during contact checking to avoid checking links that are allowed to be in collision and contact data should not be calculated.

1.3.9 SRDF Documentation

For information about the syntax for the Tesseract SRDF, read more details on the Tesseract SRDF Package page.

1.4 Tesseract Setup Wizard

1.4.1 Overview

The Tesseract Setup Wizard is a GUI that is designed to help you generate a Semantic Robot Description Format file (SRDF). This file defines your robot's:

- allowed collision matrix
- kinematics groups
- user defined joint states
- user defined tool center points (TCPs)
- OPW kinematics parameters

1.4.2 Getting Started

The Tesseract Setup Wizard is part of the Tesseract Ignition snap package.

- You can install this package by using the following command:

```
sudo snap install tesseract-ignition
```

1.4.3 Step 1: Start

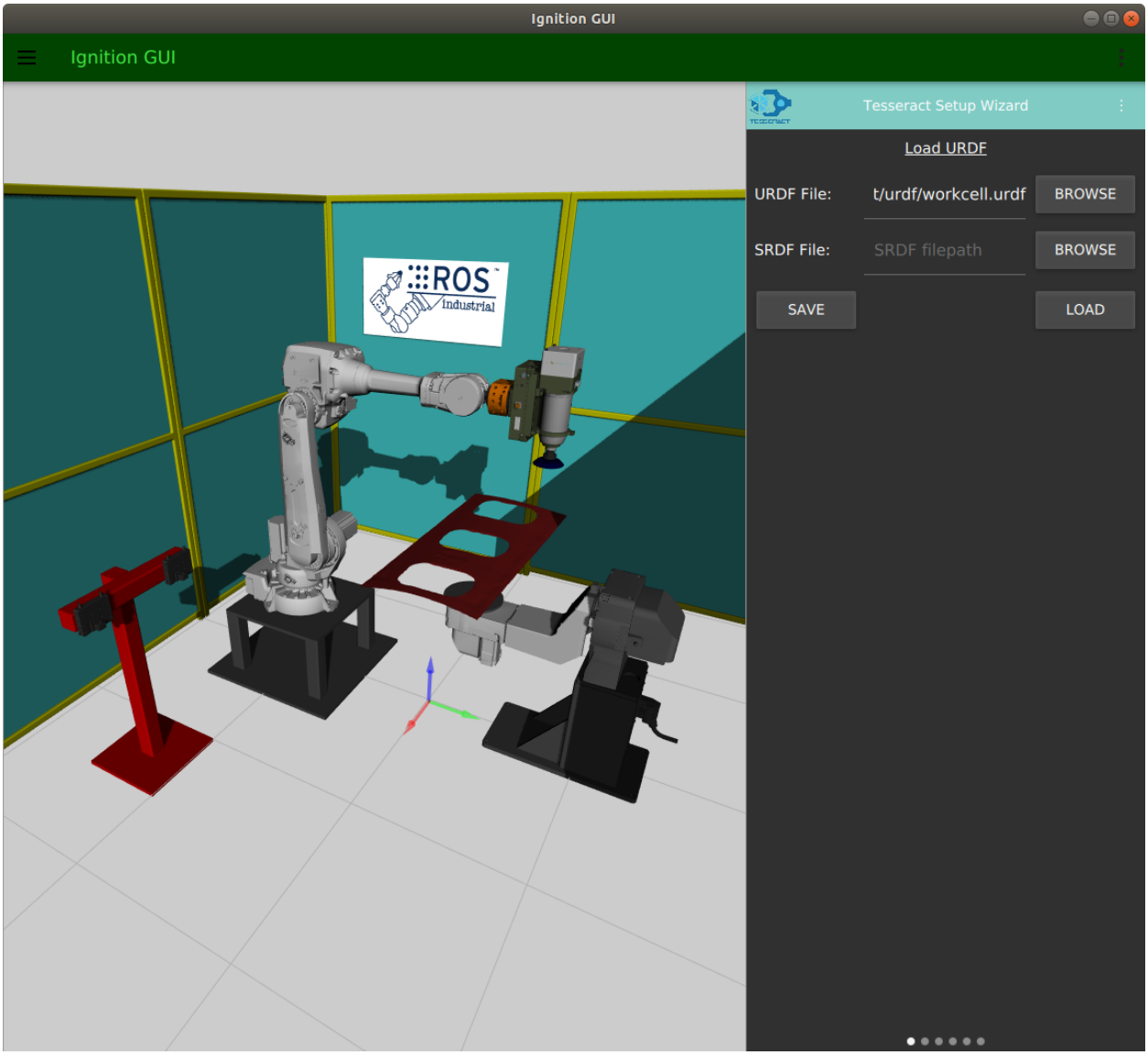
Note: If you are using ROS, make sure you source your workspace before launching the Tesseract Setup Wizard

- Launch the Tesseract Setup Wizard with the following command:

```
tesseract-ignition.tesseract-setup-wizard
```

Note: The Tesseract Setup Wizard contains a toolbar with pages for each component of the SRDF file. To navigate to different pages, you must click on the dark gray background of the tool bar, hold, and drag you mouse left or right.

1.4.4 Step 2: Load URDF and SRDF



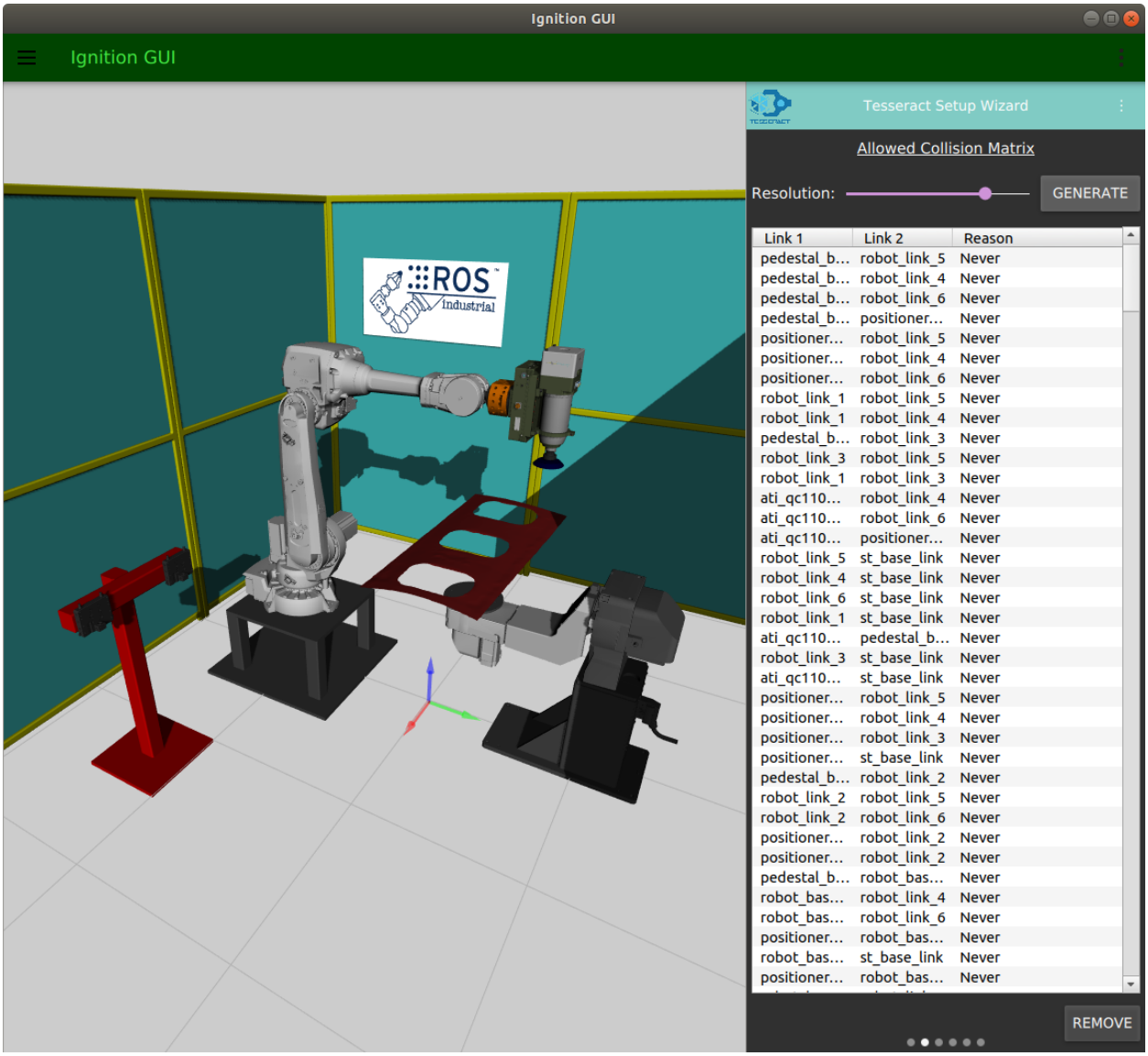
To use the Tesseract Setup Wizard and create/edit a SRDF file, we'll first need to load in your URDF file. If you'd like to edit an existing SRDF, you may also load that in as well.

- To load in your URDF file:
  1. Click the **BROWSE** button to the right of the *URDF File* field.
  2. Find and select your URDF file.
- If you would like to edit an existing SRDF file:
  1. Click the **BROWSE** button to the right of the *SRDF File* field.
  2. Find and select you SRDF file.
- Once you have selected your URDF file (required) and SRDF file (not required):
  1. Click **LOAD** to load your model into the setup wizard.
  2. At this point, you should see your model appear in the workspace to the left of the toolbar.

3. If your model did not appear, check the terminal to see error messages that will help you debug the issue.

- Click, hold, and drag left to move to the next page.

1.4.5 Step 3: Generate/Edit Allowed Collision Matrix



The allowed collision matrix specifies which links don't need to be checked for collisions. Links that are adjacent to each other or are never in collision should be ignored by the Tesseract Collision Manager.

Note: If you loaded in an existing SRDF file, the table should already be populated with your allowed collision matrix.

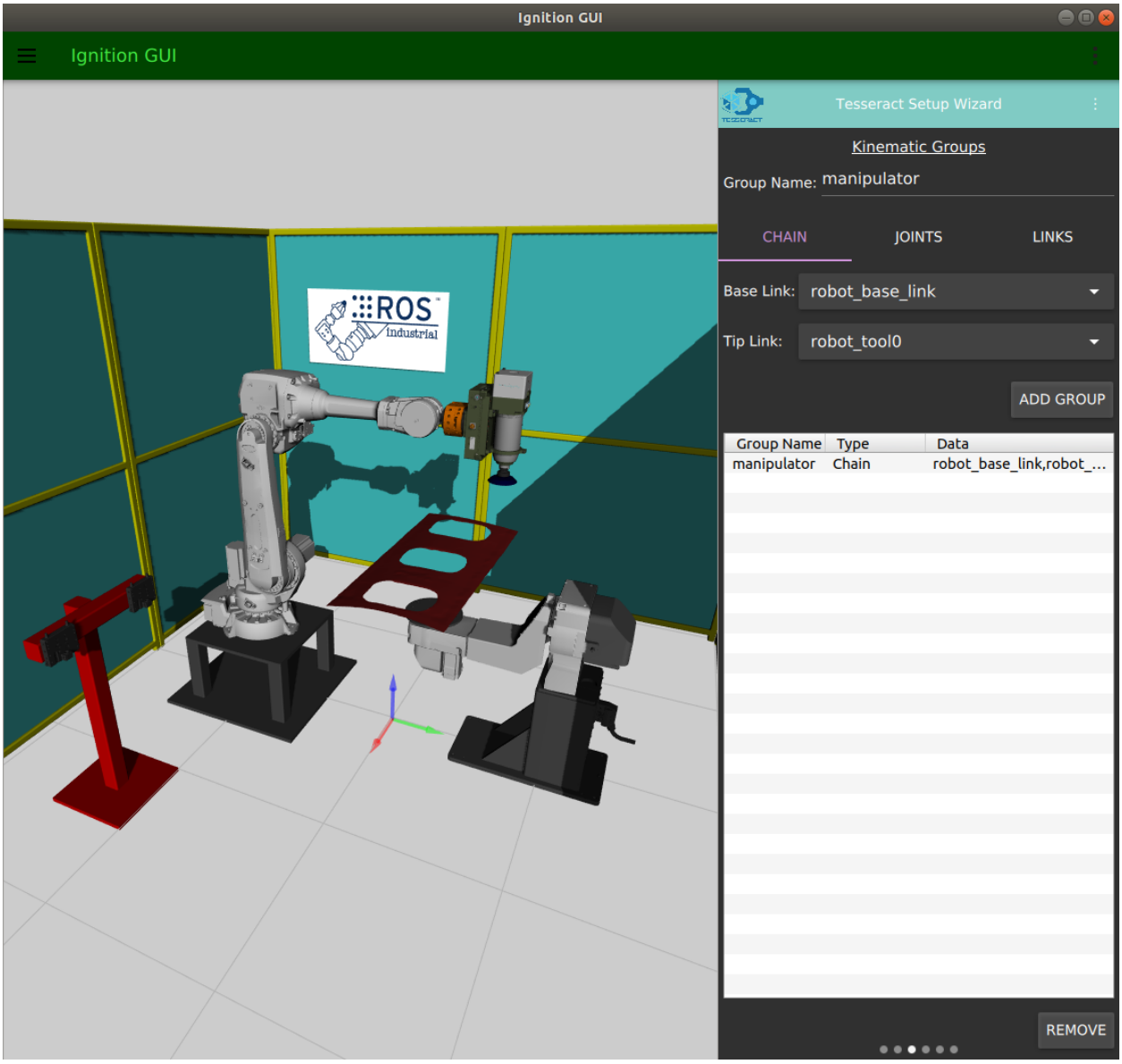
- To generate an allowed collision matrix:

  1. Click **GENERATE**. After a couple seconds you should see the table populate with your default collision matrix.

2. Inspect your collision matrix and ensure that there are no link pairs that should not be disabled for collision checking.
3. If you would like to remove a link pair (enabling it for collision checking) select the row in the table corresponding to the link pair, and click **REMOVE**.

- Click, hold, and drag left to move to the next page.

1.4.6 Step 4: Create/Edit Kinematics Groups



A kinematics group specifies a group of links that belong to a specific part of your robot. Usually, you'll want to specify the chain of links that represent the arm and will be used for motion planning.

- A kinematics group can be specified in three ways:
  1. By selecting a chain of links
  2. By selecting each joint in the group

3. By selecting each link in the group

- First, enter a name for your kinematics group in the *Group Name* field.

Method 1: Select Chain of Links

Specifying a chain is the most common way to specify kinematics group.

- To do this:
  1. Click the **CHAIN** tab.
  2. Click the *Base Link* drop down box.
  3. Select the link that represents the first link in your chain (usually this is named something like *base_link*).
4. Click the *Tip Link* drop down box.
5. Select the link that represents the end of your chain (usually this is named something link *tool0* or *tcp*).
6. Click **ADD GROUP** to add the chain to your list of kinematics groups.

Method 2: Select All Joints

You can also specify a kinematics group by specifying each joint in the group. To do this:

1. Click the **JOINTS** tab.
2. Click the *Joint Names* drop down box.
3. Select a joint to add to the group.
4. Click **ADD** to add the joint to the group.
5. Repeat steps 1-4 for each joint.
6. If you need to remove a joint, select the joint in the list and click **REMOVE**
7. Once you have added all joints to your group, click **ADD GROUP**.

Method 3: Select All Links

You can also specify a kinematics group by specifying each link in the group. To do this:

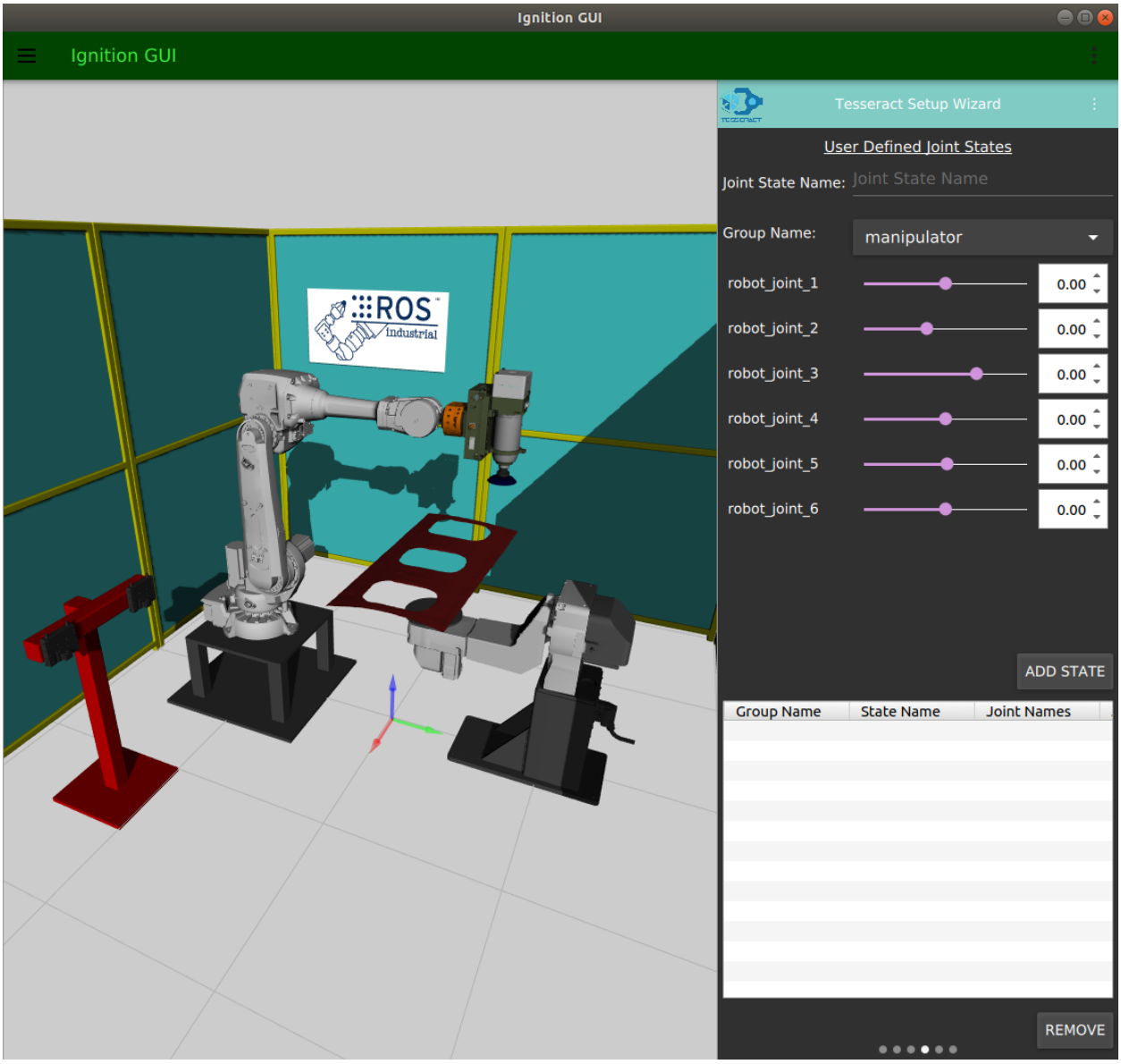
1. Click the **LINKS** tab.
2. Click the *Link Names* drop down box.
3. Select a link to add to the group.
4. Click **ADD** to add the link to the group.
5. Repeat the last three steps for each link.
6. If you need to remove a link, select the link in the list and click **REMOVE**
7. Once you have added all links for your group, click **ADD GROUP**.

Removing a Group

To remove a group, select the group in the table and click **REMOVE** (at the bottom of the tool bar).

- Click, hold, and drag left to move to the next page.

1.4.7 Step 5: Create/Edit Joint States



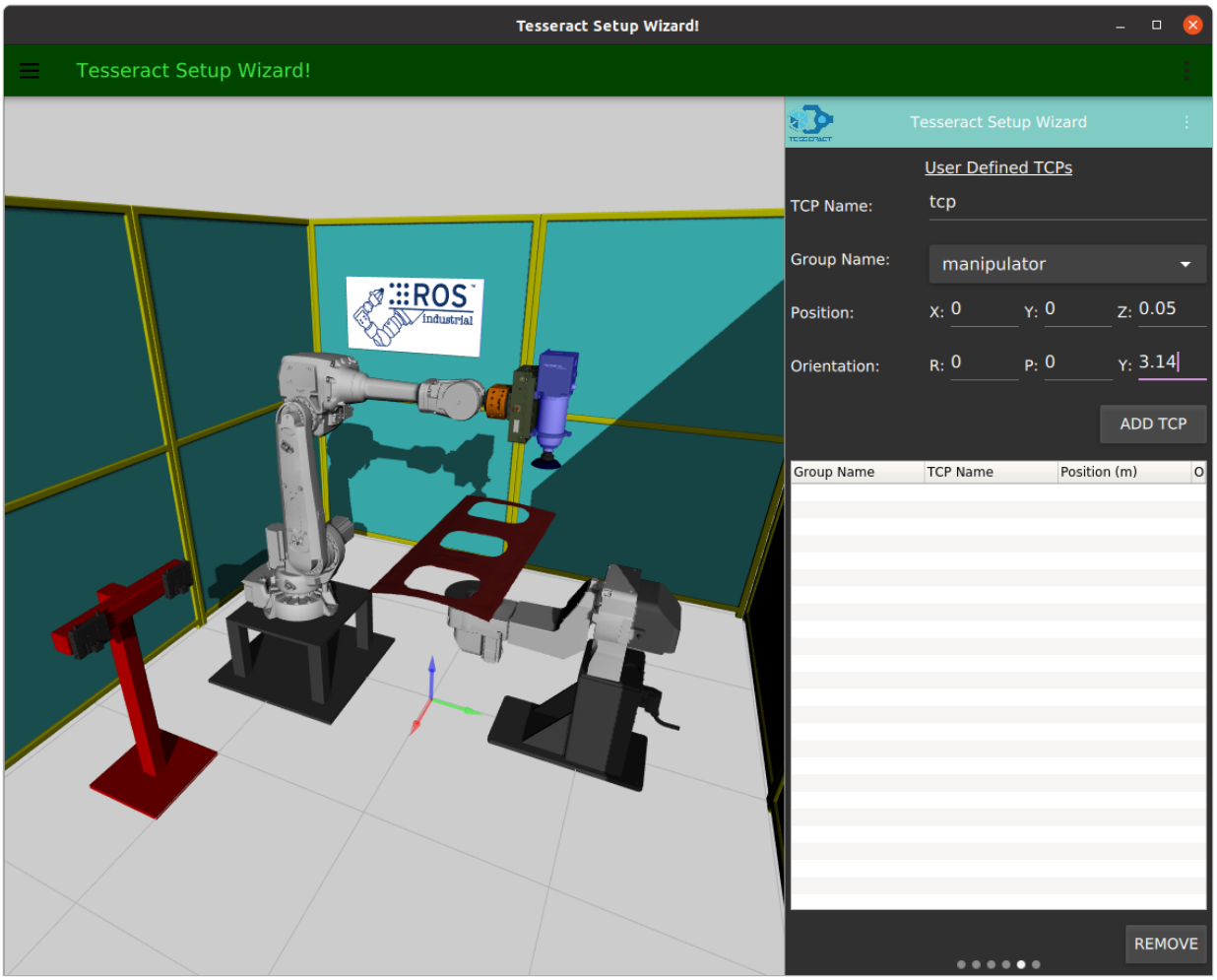
The *User Defined Joint States* page allows you to define different poses for your robot. For example, it is often useful to create a joint state called *Home* which contains the joint values for the starting/ending state of your robot.

- To define a joint state:
  1. Enter a name for your joint state in the *Joint State Name* field.
  2. Select the kinematic group that you would like to use from the *Group Name* drop down box.

3. After selecting a group name, a value field for each joint should appear bellow the *Group Name* drop down box. For each joint, select the value you'd like to set for the robot's position.
4. Once each joint value is set, click **ADD STATE**.

- To remove a joint state:
  1. Select the joint state from the table.
  2. Click **REMOVE**.
- Click, hold, and drag left to move to the next page.

1.4.8 Step 6: Create/Edit TCPs



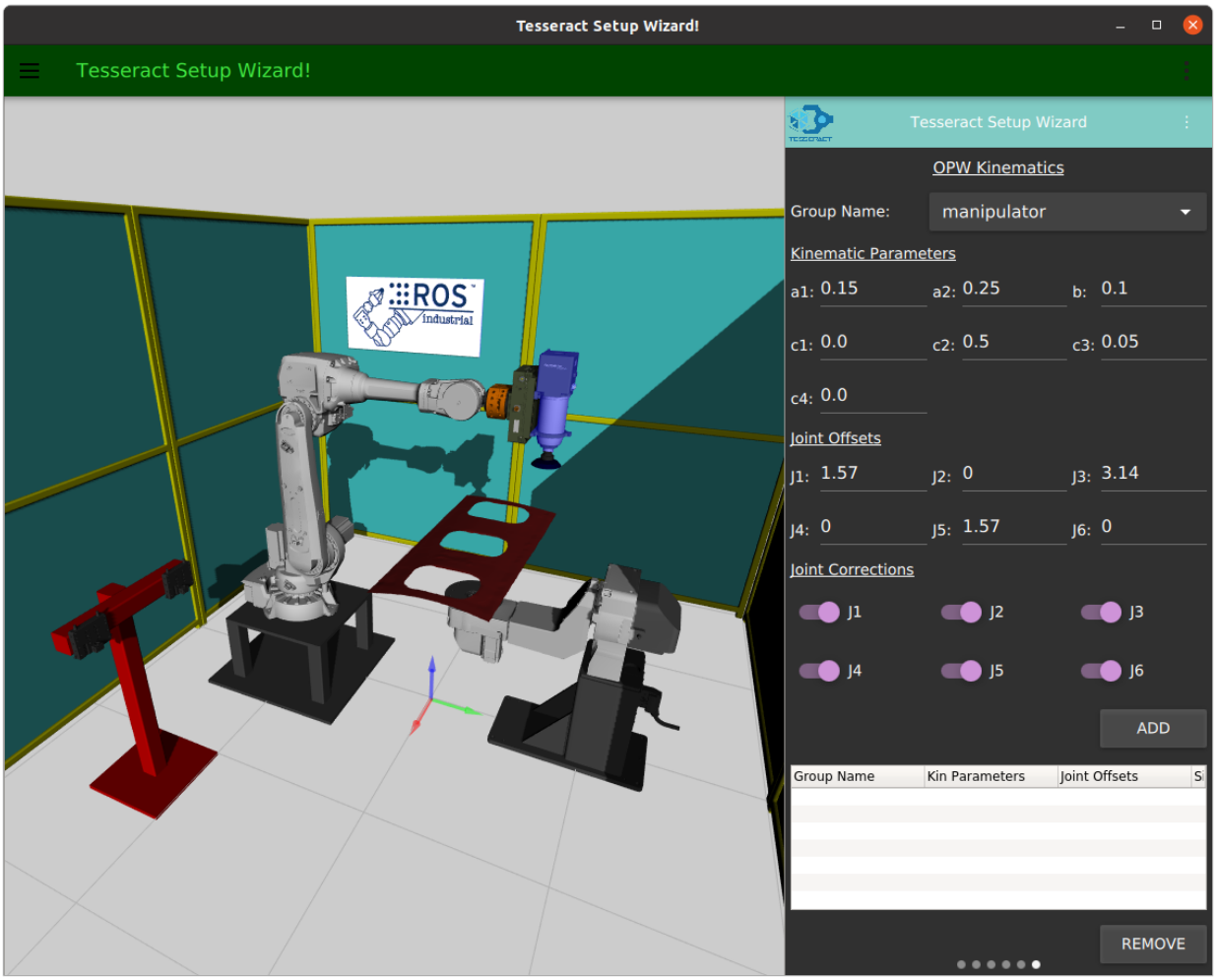
The User Defined TCPs page allows you to define Tool Center Points for your kinematic groups.

- To define a TCP:
  1. Enter a name for your TCP in the *TCP Name* field.
  2. Select the kinematic group that you would like to use from the *Group Name* drop down box.
  3. In the *Position* fields enter the *X*, *Y*, and *Z* positions of the TCP in reference to the last link in your kinematics group.

4. In the *Orientation* fields enter the Roll (*R*), Pitch (*P*), and Yaw (*Y*) of the TCP in reference to the last link in your kinematics group.
5. Click **ADD TCP**.

- To remove a TCP:
  1. Select the TCP from the table
  2. Click **REMOVE**.
- Click, hold, and drag left to move to the next page.

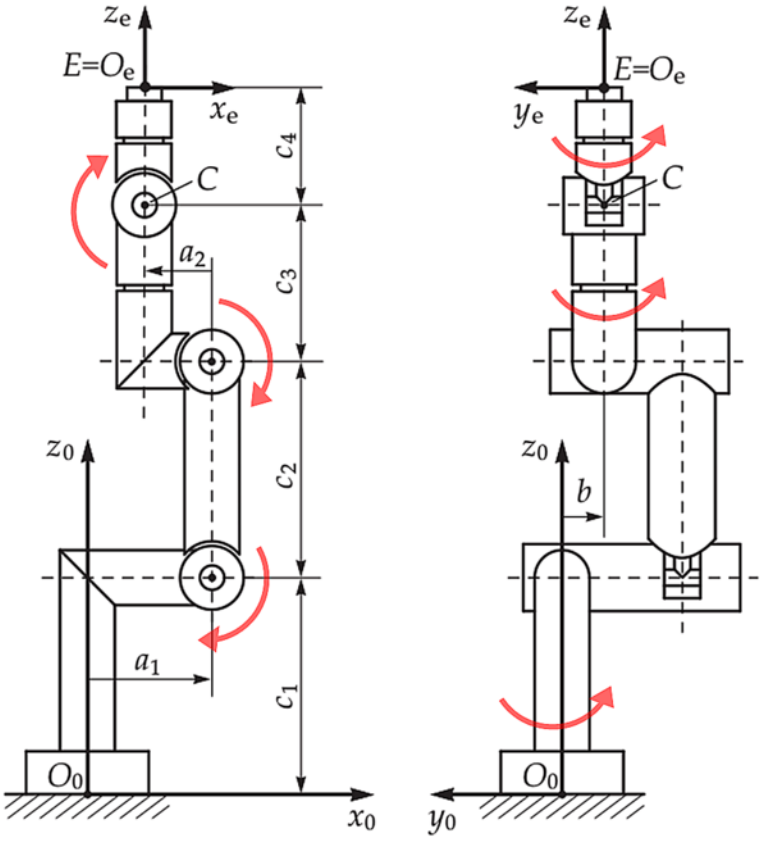
1.4.9 Step 7: Setting OPW Parameters



OPW is an efficient inverse kinematics solver for robots with parallel bases and spherical wrists. This algorithm requires 7 measurements from the robot's specification sheet to be defined here. These values will be stored in the SRDF and used by the OPW solver.

- To define your robot's OPW parameters:

1. Use the following diagram to determine each parameter

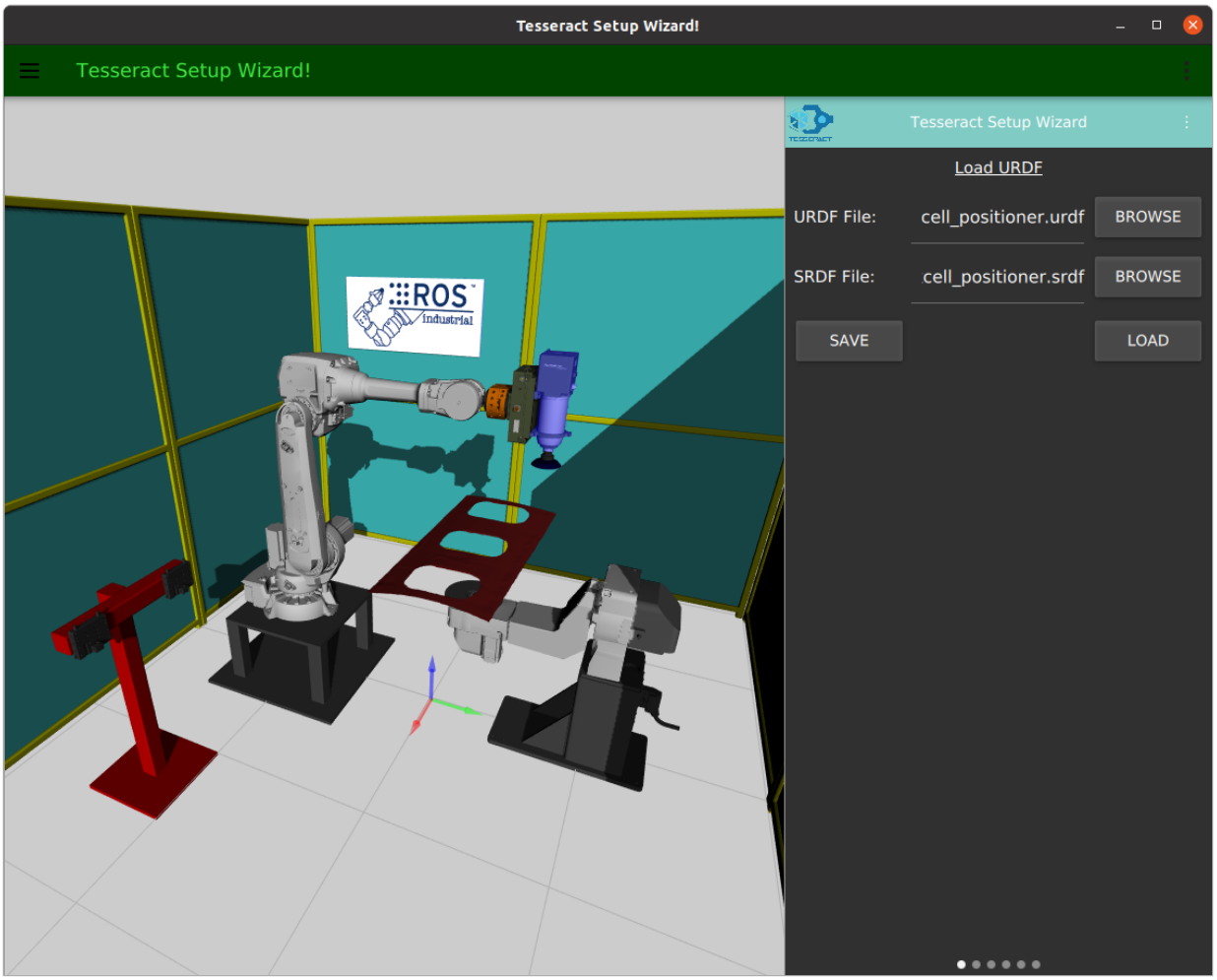


(a) Side view (b) Back view

2. Enter each value in it's respective field.

- For more details on the OPW algorithm, visit the opw_kinematics github repository.
- Click, hold, and drag left to move to the next page.

1.4.10 Step 8: Saving the SRDF File



Settings for the allowed collision matrix, kinematics groups, joint states, tcp values, and OPW parameters are all stored in a Semantic Robot Description Format file (SRDF).

- To save your SRDF file:
  1. Scroll back to the left most page where you originally loaded your URDF file.
  2. Click **SAVE** and select a file and location to save the SRDF to.

1.5 Overview

Tesseract is a motion planning framework developed by Southwest Research Institute for industrial automation applications focusing on quality, robustness and performance. This document serves an overview of its capabilities and for more details please refer to the wiki and API Documentation.

Before we begin, this is not a fork of MoveIt. It has been developed from the ground up but was inspired by MoveIt where we leveraged our knowledge of MoveIt to help shape the architecture to enable the necessary functionality needed in industrial automation.

Tesseract Planning Framework is currently broken out into four repositories.

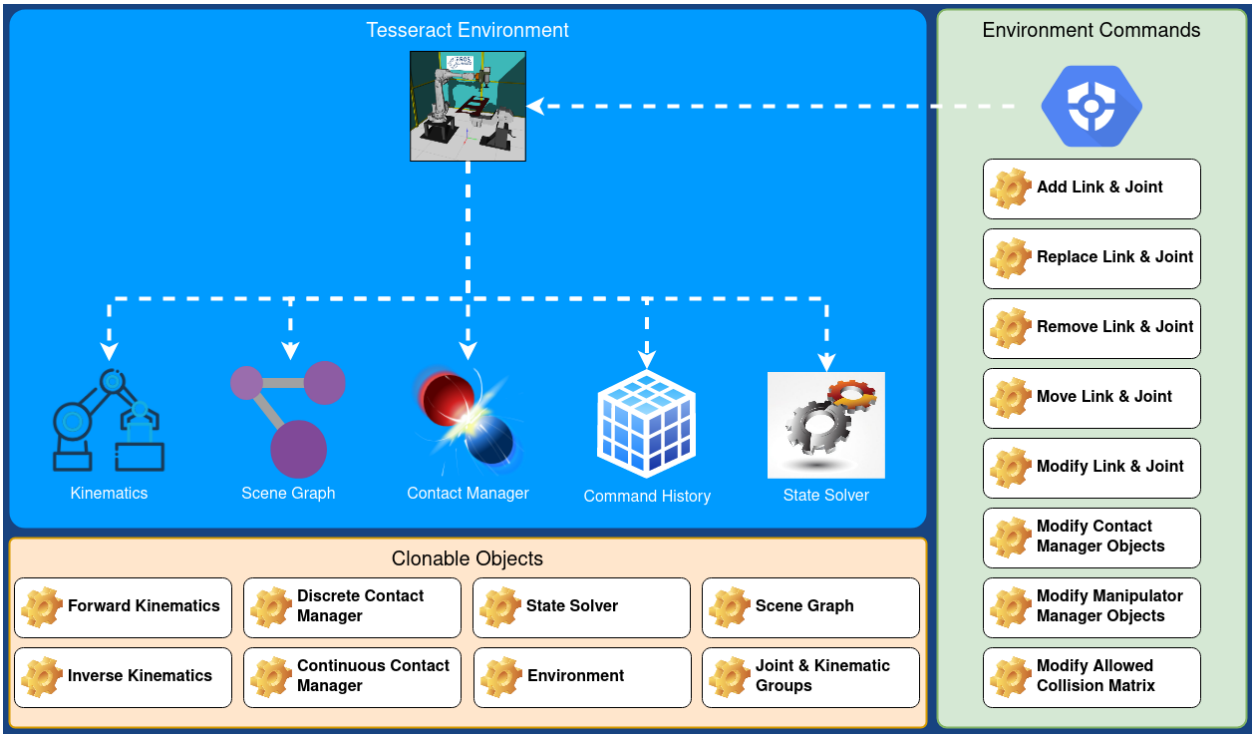
tesseract - Contain core package shown in the image below tesseract_planning - Contains motion and process planning packages (TrajOpt, OMPL, Descartes, etc.) tesseract_python - Python wrappers for tesseract and tesseract_planning repositories tesseract_ros - ROS wrappers exposing the capabilities of tessseract and tesseract_planning repositories

1.5.1 Tesseract Environment

The Tesseract Environment is a manager which interface with the State Solver, Scene Graph, Contact Managers, Manipulator Manager and Command History. It provides really no additional capability other than interfacing with these components in a thread safe way and providing additional restriction. For example, the scene graph support adding just links creating disconnected graphs, though the environment restricts things such that everything must be one connected graph. Where if you add just a link it will create a joint attaching it to the base link of the scene graph. In addition if you remove a link or joint it removes all children.

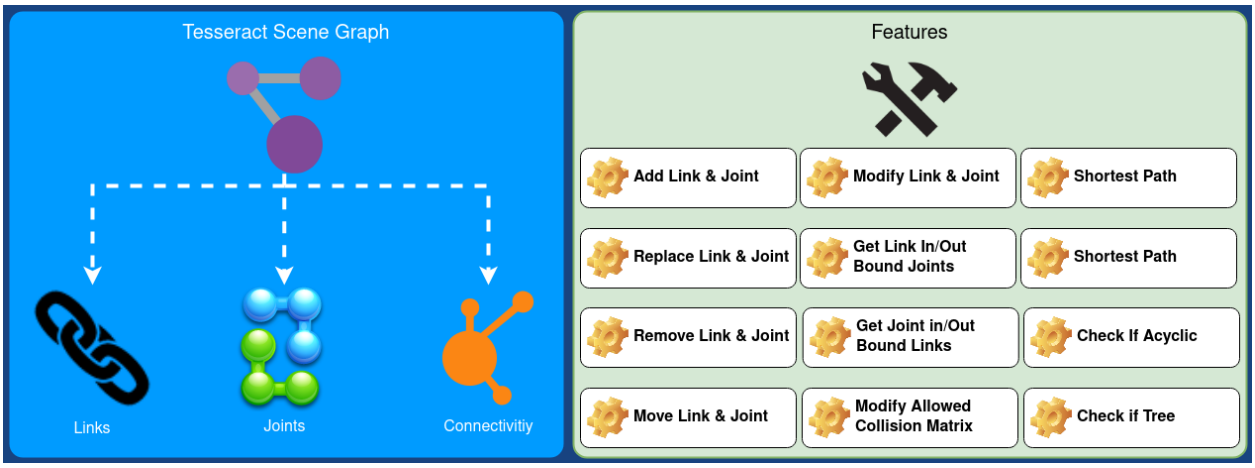
Most if not all components can be cloned (performing a deep copy). This was a critical performance decision which is different from MoveIt. MoveIt leverages the statement that all const function must be thread safe. This imposes a performance issue when it comes to motion planning requiring function to copy data. Instead Tesseract leverages the ability to clone to achieve optimal performance during motion planning and real-time execution and obstacle avoidance. It leverages cloning so process planners may allocate a cloned object per planner and thread along with allowing the object to be non const.

The Command History is another differentiating functionality. Almost every aspect of the environment can be changed live which adds additional complexity to the framework. In order to manage this capability the command pattern was leveraged to track the changes to the environment. This way all changes to the environment are tracked which is valuable in a production environment and facilitate the ability to leverage this Command History to create an exact replica. This also eliminates the need for having access to the URDF and SRDF all the time. Once the URDF and SRDF has been processed all information is stored in internal data structures (Scene Graph and Manipulator Manager) which are then stored in the Command History. This eliminates the URDF and SRDF as a critical component and are not required to leverage Tesseract.



1.5.2 Tesseract Scene Graph

The scene graph is used to manage connectivity of the environment leveraging boost graph and manages links and joints. It inherits from boost graph so you can leverage any of the boost graph capabilities for searching the scene graph. This should be used to gain access to link and joint information. The image below highlights some of the features provided but is not limited to what is shown.



1.5.3 Tesseract Collision

The tessseract collision package provides support for performing both discrete and continuous collision checking. It understand nothing about connectivity of the object it is managing.

Features

1. Add Links
2. Set link transforms
3. Enable/Disable links
4. Set default contact threshold
5. Set link pair contact threshold
6. Perform differnt contact test types (FIRST, CLOSEST, ALL)

Contact Test Types

Contact Test Type	Description
ContactTestType::FIRST	Exit on first contact. This is good for planners like Descartes and OMPL.
ContactTestType::CLOSEST	Store only closets for each collision object. This is good for planners like TrajOpt.
ContactTestType::ALL	Store all contacts for each collision object. This is good for planners like TrajOpt.

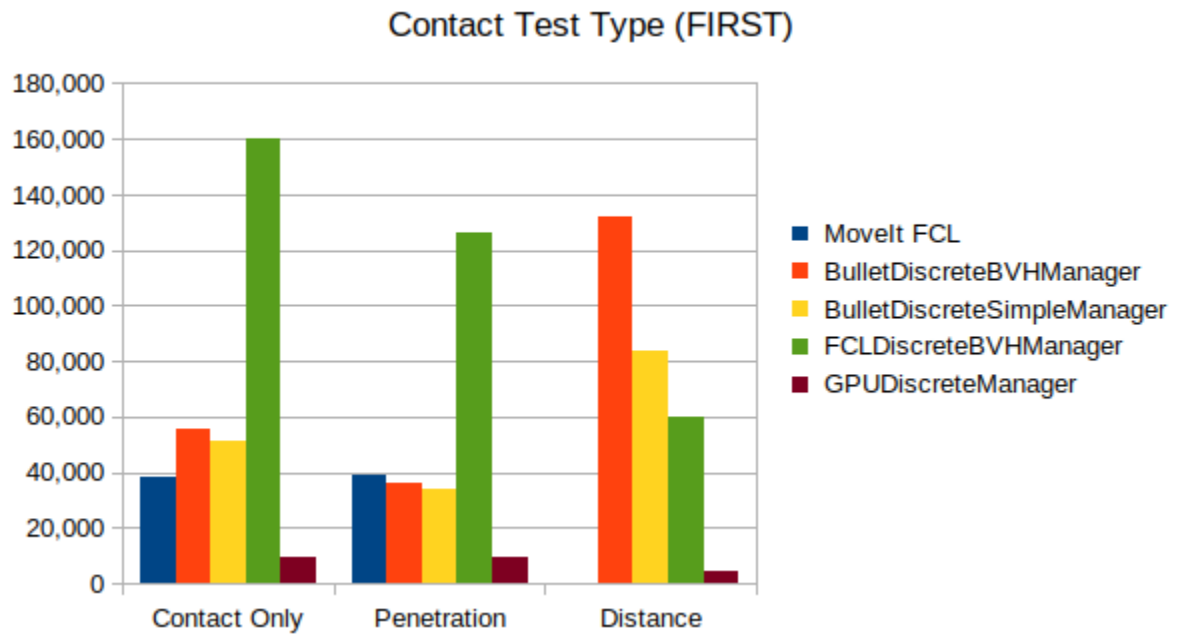
Available Contact Checkers

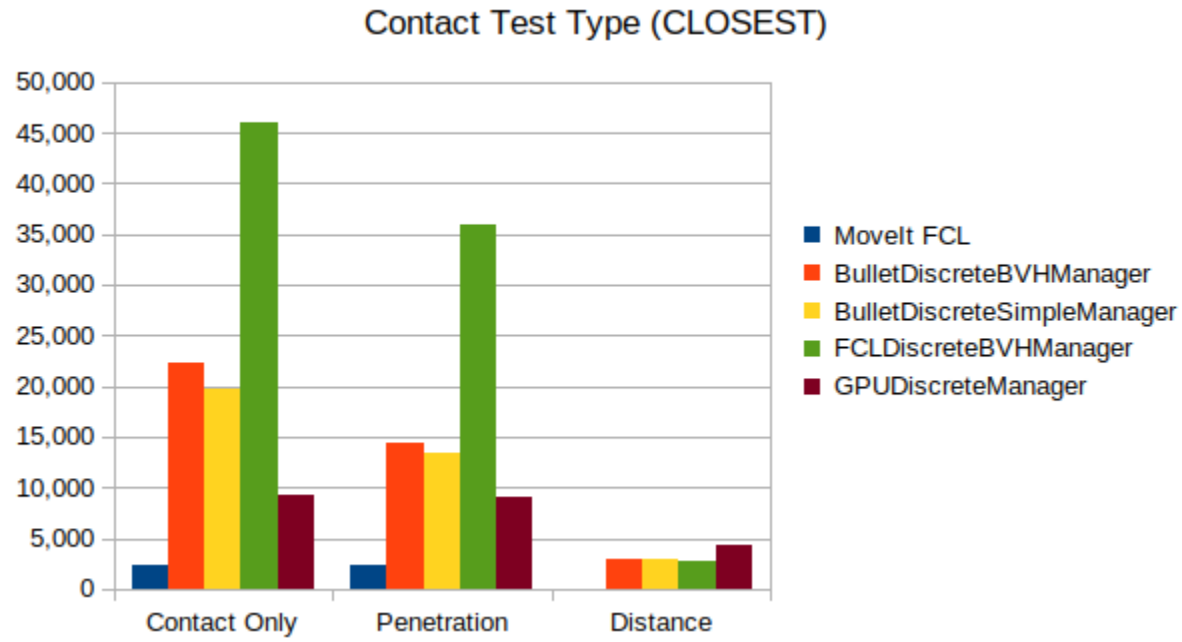
Contact Checker	Type	Description
BulletDiscreteBVHManager	Discrete	This leverages the Bullet Physics BVH to perform discrete contact checking
BulletCastBVHManager	Continuous	This leverages the Bullet Physics BVH to perform continuous contact checking by creating a casted convex hull between two link locations.
BulletDiscreteSimpleManager	Discrete	This leverages the Bullet Physics to perform discrete contact checking. It is a naive implementation where it loops over all contact pairs and checks for contact. In relatively small environments this can be faster than leveraging the BVH implementation.
BulletCastSimpleManager	Continuous	This leverages the Bullet Physics to perform continuous contact checking by creating a casted convex hull between two link locations.. It is a naive implementation where it loops over all contact pairs and checks for contact. In relatively small environments this can be faster than leveraging the BVH implementation.
FCLDiscreteBVHManager	Discrete	This leverages the Flexible Collision Library to perform discrete contact checking.
GPUDiscreteBVHManager	Discrete	This leverages an library for performing GPU/CPU contact checking

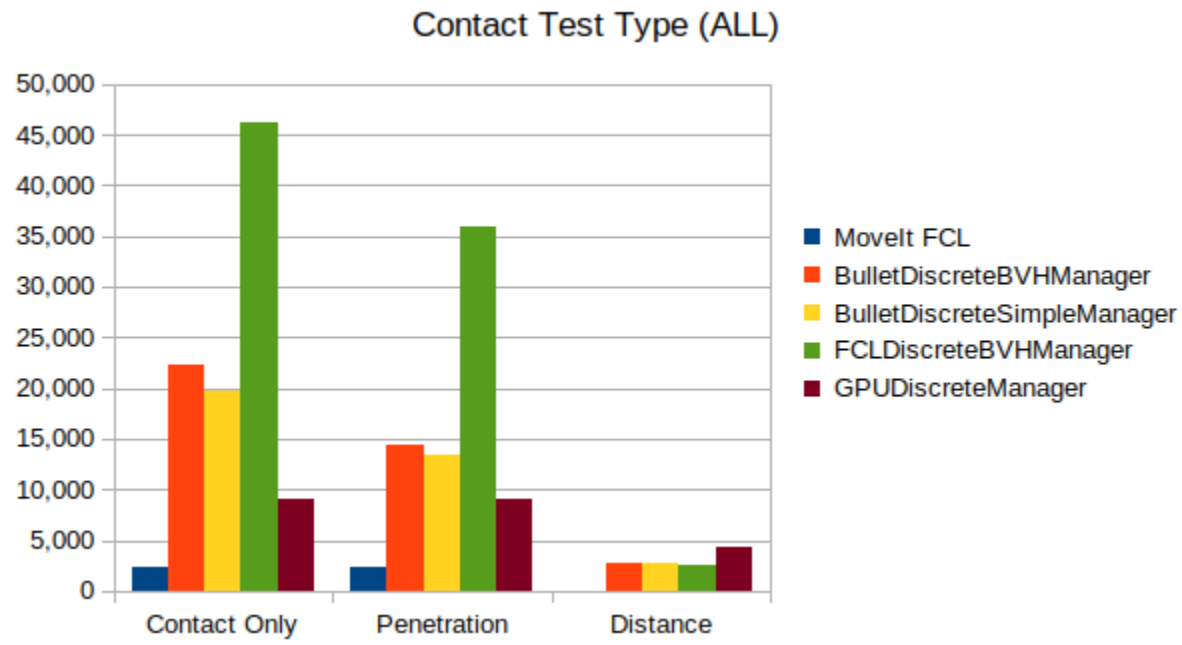
Benchmark Data

FCL Comparison

Type	FIRST	CLOSEST	ALL
Contact Only	3x Faster	19x Faster	19x Faster
Penetration	2x Faster	15x Faster	15x Faster
Distance	6,000x Faster	265x Faster	265x Faster


Contact Test Type (FIRST)
180,000
160,000
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
Contact Only
Penetration
Distance
MoveIt FCL
BulletDiscreteBVHManager
BulletDiscreteSimpleManager
FCLDiscreteBVHManager
GPUDiscreteManager


Contact Test Type (CLOSEST)
50,000
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
Contact Only
Penetration
Distance
MoveIt FCL
BulletDiscreteBVHManager
BulletDiscreteSimpleManager
FCLDiscreteBVHManager
GPUDiscreteManager



1.5.4 Tesseract Kinematics

This contains an interface for both forward and inverse kinematics. One design decision made is that the kinematics are not environment aware to simplify the process of implementing both forward and inverse kinematics. It is setup to support kinematic chains, trees and graphs but currently only has implementations for chains and trees.

It also includes specialized support for the following: - IKFast generated inverse kinematics - OPW Kinematics - Robot on a positioner - Robot with an external positioner - Universal Robot

The kinematics are all loaded as plugins which can be defined in a yaml file and added to the SRDF.

1.5.5 Building

Build and Run All

Run the following command:

```
catkin build --force-cmake -DTESSERACT_ENABLE_TESTS=ON
```

Build and Run One Package

Run the following command:

```
catkin build pkg_name --force-cmake -DTESSERACT_ENABLE_TESTS=ON
```

Make Output Verbose

Run the following command:

```
catkin build pkg_name --v --force-cmake -DTESSERACT_ENABLE_TESTS=ON
```

Building with Clang-Tidy

Note: Clang-tidy is automatically enabled if cmake argument -DTESSERACT_ENABLE_TESTING_ALL=ON is passed.

To build with clang-tidy, you must pass the -DTESSERACT_ENABLE_CLANG_TIDY=ON to cmake when building:

```
catkin build --force-cmake -DTESSERACT_ENABLE_CLANG_TIDY=ON
```

1.6 Quickstart

1.7 Examples

1.8 Tesseract Core Packages

Warning: These packages are under heavy development and are subject to change.

1.8.1 Tesseract Package

This is the main class that manages the major component Environment, Forward Kinematics, Inverse Kinematics and loading from various data.

1.8.2 Tesseract Collision Package

Background

This package is used for performing both discrete and continuous collision checking. It understands nothing about connectivity of the object within. It purely allows for the user to add objects to the checker, set object transforms, enable/disable objects, set contact distance per object, and perform collision checks.

Note: The contact managers are load as plugins through a yaml config file which is added to the SRDF file.

Contact Manager Plugin Config

The contact manager config file has four sections:

Section	Description
search_paths	A list of locations to search for libraries. It searches in the order provided.
search_libraries	A list of libraries to search for classes
discrete_plugins	The discrete contact manater plugins plugins.
continuous_plugins	The continuous contact manater plugins plugins.

Example Config file:

```
contact_manager_plugins:
  search_paths:
    - /usr/local/lib
  search_libraries:
    - tesseract_collision_bullet_factories
    - tesseract_collision_fcl_factories
  discrete_plugins:
    default: BulletDiscreteBVHManager
    plugins:
      BulletDiscreteBVHManager:
        class: BulletDiscreteBVHManagerFactory
      BulletDiscreteSimpleManager:
        class: BulletDiscreteSimpleManagerFactory
      FCLDiscreteBVHManager:
        class: FCLDiscreteBVHManagerFactory
  continuous_plugins:
    default: BulletCastBVHManager
    plugins:
      BulletCastBVHManager:
        class: BulletCastBVHManagerFactory
      BulletCastSimpleManager:
        class: BulletCastSimpleManagerFactory
```

Features

1. Add Links
2. Set link transforms
3. Enable/Disable links
4. Set default contact threshold
5. Set link pair contact threshold
6. Perform differnt contact test types (FIRST, CLOSEST, ALL)

Contact Test Types

Contact Test Type	Description
ContactTestType::FIRST	Exit on first contact. This is good for planners like Descartes and OMPL.
ContactTestType::CLOSEST	Store only closets for each collision object. This is good for planners like TrajOpt.
ContactTestType::ALL	Store all contacts for each collision object. This is good for planners like TrajOpt.

Available Contact Checkers

Contact Checker	Type	Description
BulletDiscreteBVHManager	Discrete	This leverages the Bullet Physics BVH to perform discrete contact checking
BulletCastBVHManager	Continuous	This leverages the Bullet Physics BVH to perform continuous contact checking by creating a casted convex hull between two link locations.
BulletDiscreteSimpleManager	Discrete	This leverages the Bullet Physics to perform discrete contact checking. It is a naive implementation where it loops over all contact pairs and checks for contact. In relatively small environments this can be faster than leveraging the BVH implementation.
BulletCastSimpleManager	Continuous	This leverages the Bullet Physics to perform continuous contact checking by creating a casted convex hull between two link locations.. It is a naive implementation where it loops over all contact pairs and checks for contact. In relatively small environments this can be faster than leveraging the BVH implementation.
FCLDiscreteBVHManager	Discrete	This leverages the Flexible Collision Library to perform discrete contact checking.
GPUDiscreteBVHManager	Discrete	This leverages an library for performing GPU/CPU contact checking

Discrete Collision Checker Example

```
#include <tesseract_common/macros.h>
TESSERACT_COMMON_IGNORE_WARNINGS_PUSH
#include <console_bridge/console.h>
TESSERACT_COMMON_IGNORE_WARNINGS_POP

#include <tesseract_collision/bullet/bullet_discrete_bvh_manager.h>
#include <tesseract_collision/bullet/convex_hull_utils.h>

using namespace tesseract_collision;
using namespace tesseract_geometry;
```

(continues on next page)

(continued from previous page)

```
std::string toString(const Eigen::MatrixXd& a)
{
  std::stringstream ss;
  ss << a;
  return ss.str();
}

std::string toString(bool b) { return b ? "true" : "false"; }

int main(int /*argc*/, char** /*argv*/)
{
  // documentation:start:1: Create Collision Manager
  tesseract_collision_bullet::BulletDiscreteBVHManager checker;
  // documentation:end:1: Create Collision Manager

  // documentation:start:2: Add box to checker
  // Create a box
  CollisionShapePtr box = std::make_shared<Box>(1, 1, 1);
  Eigen::Isometry3d box_pose;
  box_pose.setIdentity();

  CollisionShapesConst obj1_shapes;
  tesseract_common::VectorIsometry3d obj1_poses;
  obj1_shapes.push_back(box);
  obj1_poses.push_back(box_pose);

  // Add box to checker in enabled state
  checker.addCollisionObject("box_link", 0, obj1_shapes, obj1_poses);
  // documentation:end:2: Add box to checker

  // documentation:start:3: Add thin box
  // Create a thin box
  CollisionShapePtr thin_box = std::make_shared<Box>(0.1, 1, 1);
  Eigen::Isometry3d thin_box_pose;
  thin_box_pose.setIdentity();

  CollisionShapesConst obj2_shapes;
  tesseract_common::VectorIsometry3d obj2_poses;
  obj2_shapes.push_back(thin_box);
  obj2_poses.push_back(thin_box_pose);

  // Add thin box to checker in disabled state
  checker.addCollisionObject("thin_box_link", 0, obj2_shapes, obj2_poses, false);
  // documentation:end:3: Add thin box

  // documentation:start:4: Add convex hull
  // Add second box to checker, but convert to convex hull mesh
  CollisionShapePtr second_box;

  auto mesh_vertices = std::make_shared<tesseract_common::VectorVector3d>();
  auto mesh_faces = std::make_shared<Eigen::VectorXi>();
```

(continues on next page)

(continued from previous page)

```
  loadSimplePlyFile(std::string(TESSERACT_SUPPORT_DIR) + "/meshes/box_2m.ply", *mesh_
↪vertices, *mesh_faces, true);

  auto mesh = std::make_shared<tesseract_geometry::Mesh>(mesh_vertices, mesh_faces);
  second_box = makeConvexMesh(*mesh);
  // documentation:end:4: Add convex hull

  // documentation:start:5: Add convex hull collision
  Eigen::Isometry3d second_box_pose;
  second_box_pose.setIdentity();

  CollisionShapesConst obj3_shapes;
  tesseract_common::VectorIsometry3d obj3_poses;
  obj3_shapes.push_back(second_box);
  obj3_poses.push_back(second_box_pose);

  checker.addCollisionObject("second_box_link", 0, obj3_shapes, obj3_poses);
  // documentation:end:5: Add convex hull collision

  CONSOLE_BRIDGE_logInform("Test when object is inside another");
  // documentation:start:6: Set active collision object
  checker.setActiveCollisionObjects({ "box_link", "second_box_link" });
  // documentation:end:6: Set active collision object

  // documentation:start:7: Set contact distance threshold
  checker.setCollisionMarginData(CollisionMarginData(0.1));
  // documentation:end:7: Set contact distance threshold

  // documentation:start:8: Set collision object transform
  // Set the collision object transforms
  tesseract_common::TransformMap location;
  location["box_link"] = Eigen::Isometry3d::Identity();
  location["box_link"].translation()(0) = 0.2;
  location["box_link"].translation()(1) = 0.1;
  location["second_box_link"] = Eigen::Isometry3d::Identity();

  checker.setCollisionObjectsTransform(location);
  // documentation:end:8: Set collision object transform

  // documentation:start:9: Perform collision check
  ContactResultMap result;
  ContactRequest request(ContactTestType::CLOSEST);
  checker.contactTest(result, request);

  ContactResultVector result_vector;
  flattenMoveResults(std::move(result), result_vector);

  CONSOLE_BRIDGE_logInform("Has collision: %s", toString(result_vector.empty()).c_str());
  CONSOLE_BRIDGE_logInform("Distance: %f", result_vector[0].distance);
  CONSOLE_BRIDGE_logInform("Link %s nearest point: %s",
                           result_vector[0].link_names[0].c_str(),
                           toString(result_vector[0].nearest_points[0]).c_str());
```

(continues on next page)

(continued from previous page)

```
  CONSOLE_BRIDGE_logInform("Link %s nearest point: %s",
                           result_vector[0].link_names[1].c_str(),
                           toString(result_vector[0].nearest_points[1]).c_str());
  CONSOLE_BRIDGE_logInform("Direction to move Link %s out of collision with Link %s: %s",
                           result_vector[0].link_names[0].c_str(),
                           result_vector[0].link_names[1].c_str(),
                           toString(result_vector[0].normal).c_str());
  // documentation:end:9: Perform collision check

  // documentation:start:10: Set collision object transform
  CONSOLE_BRIDGE_logInform("Test object is out side the contact distance");
  location["box_link"].translation() = Eigen::Vector3d(1.60, 0, 0);
  checker.setCollisionObjectsTransform(location);
  // documentation:end:10: Set collision object transform

  // documentation:start:11: Perform collision check
  result = ContactResultMap();
  result.clear();
  result_vector.clear();

  // Check for collision after moving object
  checker.contactTest(result, request);
  flattenMoveResults(std::move(result), result_vector);
  CONSOLE_BRIDGE_logInform("Has collision: %s", toString(result_vector.empty()).c_str());
  // documentation:end:11: Perform collision check

  // documentation:start:12: Change contact distance threshold
  // Set higher contact distance threshold
  checker.setDefaultCollisionMarginData(0.25);
  // documentation:end:12: Change contact distance threshold

  // documentation:start:13: Perform collision check
  CONSOLE_BRIDGE_logInform("Test object inside the contact distance");
  result = ContactResultMap();
  result.clear();
  result_vector.clear();

  // Check for contact with new threshold
  checker.contactTest(result, request);
  flattenMoveResults(std::move(result), result_vector);

  CONSOLE_BRIDGE_logInform("Has collision: %s", toString(result_vector.empty()).c_str());
  CONSOLE_BRIDGE_logInform("Distance: %f", result_vector[0].distance);
  CONSOLE_BRIDGE_logInform("Link %s nearest point: %s",
                           result_vector[0].link_names[0].c_str(),
                           toString(result_vector[0].nearest_points[0]).c_str());
  CONSOLE_BRIDGE_logInform("Link %s nearest point: %s",
                           result_vector[0].link_names[1].c_str(),
                           toString(result_vector[0].nearest_points[1]).c_str());
  CONSOLE_BRIDGE_logInform("Direction to move Link %s further from Link %s: %s",
                           result_vector[0].link_names[0].c_str(),
                           result_vector[0].link_names[1].c_str(),
```

(continues on next page)

(continued from previous page)

```
                          toString(result_vector[0].normal).c_str());
  // documentation:end:13: Perform collision check
}
```

You can find this example here.

Example Explanation

Create Contact Checker

```
  tesseract_collision_bullet::BulletDiscreteBVHManager checker;
```

Add Collision Objects to Contact Checker

Add a collision object in a enabled state

```
  // Create a box
  CollisionShapePtr box = std::make_shared<Box>(1, 1, 1);
  Eigen::Isometry3d box_pose;
  box_pose.setIdentity();

  CollisionShapesConst obj1_shapes;
  tesseract_common::VectorIsometry3d obj1_poses;
  obj1_shapes.push_back(box);
  obj1_poses.push_back(box_pose);

  // Add box to checker in enabled state
  checker.addCollisionObject("box_link", 0, obj1_shapes, obj1_poses);
```

Note: A collision object can consist of multiple collision shapes.

Add collision object in a disabled state

```
  // Create a thin box
  CollisionShapePtr thin_box = std::make_shared<Box>(0.1, 1, 1);
  Eigen::Isometry3d thin_box_pose;
  thin_box_pose.setIdentity();

  CollisionShapesConst obj2_shapes;
  tesseract_common::VectorIsometry3d obj2_poses;
  obj2_shapes.push_back(thin_box);
  obj2_poses.push_back(thin_box_pose);

  // Add thin box to checker in disabled state
  checker.addCollisionObject("thin_box_link", 0, obj2_shapes, obj2_poses, false);
```

Create convex hull from mesh file

```
// Add second box to checker, but convert to convex hull mesh
CollisionShapePtr second_box;

auto mesh_vertices = std::make_shared<tesseract_common::VectorVector3d>();
auto mesh_faces = std::make_shared<Eigen::VectorXi>();
loadSimplePlyFile(std::string(TESSERACT_SUPPORT_DIR) + "/meshes/box_2m.ply", *mesh_
↪vertices, *mesh_faces, true);

auto mesh = std::make_shared<tesseract_geometry::Mesh>(mesh_vertices, mesh_faces);
second_box = makeConvexMesh(*mesh);
```

Add convex hull collision object

```
Eigen::Isometry3d second_box_pose;
second_box_pose.setIdentity();

CollisionShapesConst obj3_shapes;
tesseract_common::VectorIsometry3d obj3_poses;
obj3_shapes.push_back(second_box);
obj3_poses.push_back(second_box_pose);

checker.addCollisionObject("second_box_link", 0, obj3_shapes, obj3_poses);
```

Set the active collision objects

```
checker.setActiveCollisionObjects({ "box_link", "second_box_link" });
```

Set the contact distance threshold

```
checker.setCollisionMarginData(CollisionMarginData(0.1));
```

Set the collision object's transform

```
// Set the collision object transforms
tesseract_common::TransformMap location;
location["box_link"] = Eigen::Isometry3d::Identity();
location["box_link"].translation()(0) = 0.2;
location["box_link"].translation()(1) = 0.1;
location["second_box_link"] = Eigen::Isometry3d::Identity();

checker.setCollisionObjectsTransform(location);
```

Perform collision check

Note: One object is inside another object

```
ContactResultMap result;
ContactRequest request(ContactTestType::CLOSEST);
checker.contactTest(result, request);

ContactResultVector result_vector;
flattenMoveResults(std::move(result), result_vector);

CONSOLE_BRIDGE_logInform("Has collision: %s", toString(result_vector.empty()).c_str());
CONSOLE_BRIDGE_logInform("Distance: %f", result_vector[0].distance);
CONSOLE_BRIDGE_logInform("Link %s nearest point: %s",
                         result_vector[0].link_names[0].c_str(),
                         toString(result_vector[0].nearest_points[0]).c_str());
CONSOLE_BRIDGE_logInform("Link %s nearest point: %s",
                         result_vector[0].link_names[1].c_str(),
                         toString(result_vector[0].nearest_points[1]).c_str());
CONSOLE_BRIDGE_logInform("Direction to move Link %s out of collision with Link %s: %s",
                         result_vector[0].link_names[0].c_str(),
                         result_vector[0].link_names[1].c_str(),
                         toString(result_vector[0].normal).c_str());
```

Set the collision object's transform

```
CONSOLE_BRIDGE_logInform("Test object is out side the contact distance");
location["box_link"].translation() = Eigen::Vector3d(1.60, 0, 0);
checker.setCollisionObjectsTransform(location);
```

Perform collision check

Note: The objects are outside the contact threshold

```
result = ContactResultMap();
result.clear();
result_vector.clear();

// Check for collision after moving object
checker.contactTest(result, request);
flattenMoveResults(std::move(result), result_vector);
CONSOLE_BRIDGE_logInform("Has collision: %s", toString(result_vector.empty()).c_str());
```

Change contact distance threshold

```
// Set higher contact distance threshold
checker.setDefaultCollisionMarginData(0.25);
```

Perform collision check

Note: The objects are inside the contact threshold

```
CONSOLE_BRIDGE_logInform("Test object inside the contact distance");
result = ContactResultMap();
result.clear();
result_vector.clear();

// Check for contact with new threshold
checker.contactTest(result, request);
flattenMoveResults(std::move(result), result_vector);

CONSOLE_BRIDGE_logInform("Has collision: %s", toString(result_vector.empty()).c_str());
CONSOLE_BRIDGE_logInform("Distance: %f", result_vector[0].distance);
CONSOLE_BRIDGE_logInform("Link %s nearest point: %s",
                         result_vector[0].link_names[0].c_str(),
                         toString(result_vector[0].nearest_points[0]).c_str());
CONSOLE_BRIDGE_logInform("Link %s nearest point: %s",
                         result_vector[0].link_names[1].c_str(),
                         toString(result_vector[0].nearest_points[1]).c_str());
CONSOLE_BRIDGE_logInform("Direction to move Link %s further from Link %s: %s",
                         result_vector[0].link_names[0].c_str(),
                         result_vector[0].link_names[1].c_str(),
                         toString(result_vector[0].normal).c_str());
```

Running the Example

Build the Tesseract Workspace:

```
catkin build
```

Navigate to the build folder containing the executable:

```
cd build/tesseract_collision/examples
```

Run the executable:

```
./tesseract_collision_box_box_example
```

1.8.3 Tesseract Common Package

This package contains common functionality needed by the majority of the packages.

1.8.4 Tesseract Environment Package

This package contains the Tesseract Environment which provides functionality to add, remove, move and modify links and joint. It also manages adding object to the contact managers and provides the ability.

The Tesseract Environment is a manager which interface with the State Solver, Scene Graph, Contact Managers, Manipulator Manager and Command History. It provides really no additional capability other than interfacing with these components in a thread safe way and providing additional restriction. For example, the scene graph support adding just links creating disconnected graphs, though the environment restricts things such that everything must be one connected graph. Where if you add just a link it will create a joint attaching it to the base link of the scene graph. In addition if you remove a link or joint it removes all children.

Most if not all components can be cloned (performing a deep copy). This was a critical performance decision which is different from MoveIt. MoveIt leverages the statement that all const function must be thread safe. This imposes a performance issue when it comes to motion planning requiring function to copy data. Instead Tesseract leverages the ability to clone to achieve optimal performance during motion planning and real-time execution and obstacle avoidance. It leverages cloning so process planners may allocate a cloned object per planner and thread along with allowing the object to be non const.

The Command History is another differentiating functionality. Almost every aspect of the environment can be changed live which adds additional complexity to the framework. In order to manage this capability the command pattern was leveraged to track the changes to the environment. This way all changes to the environment are tracked which is valuable in a production environment and facilitate the ability to leverage this Command History to create an exact replica. This also eliminates the need for having access to the URDF and SRDF all the time. Once the URDF and SRDF has been processed all information is stored in internal data structures (Scene Graph and Manipulator Manager) which are then stored in the Command History. This eliminates the URDF and SRDF as a critical component and are not required to leverage Tesseract.

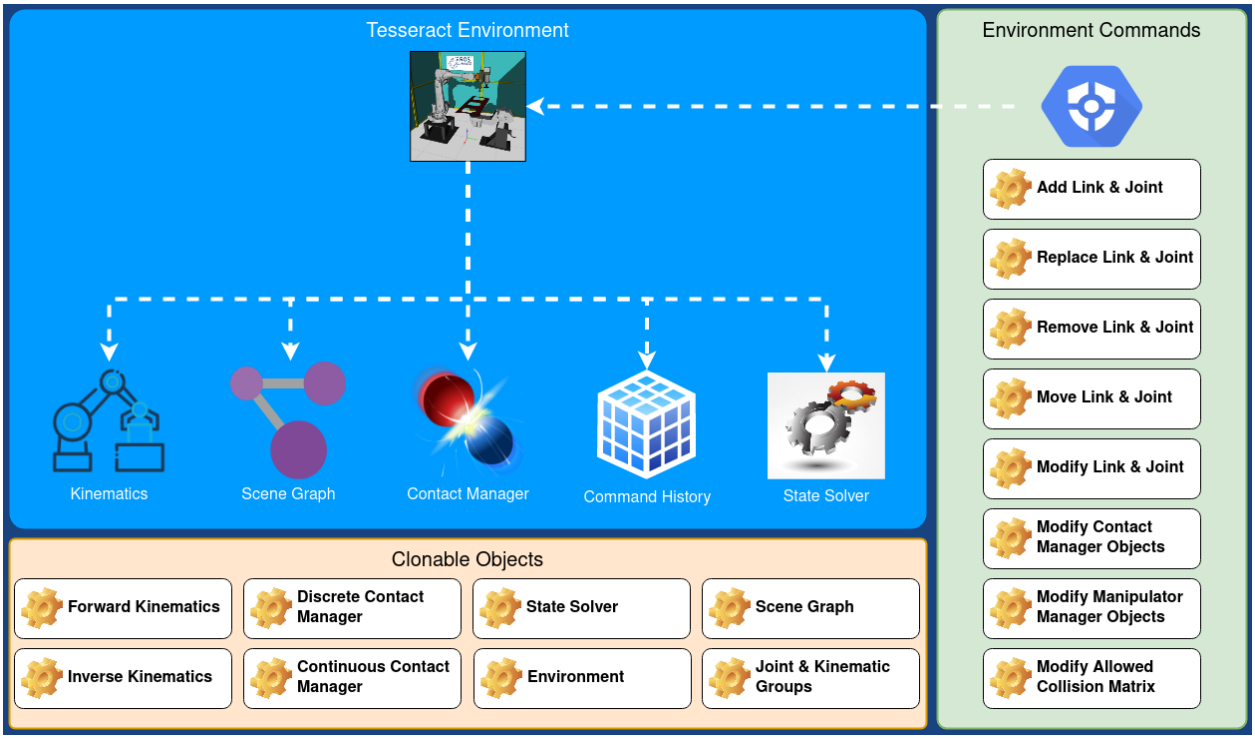
1.8.5 Tesseract Geometry Package

Background

This package contains geometry types used by Tesseract including primitive shapes, mesh, convex hull mesh, octomap and signed distance field.

Features

1. Primitive Shapes
    - Box
    - Cone
    - Capsule
    - Cylinder
    - Plane
    - Sphere
2. Mesh



3. Convex Mesh
4. SDF Mesh
5. Octree

Creating Geometry Shapes

```cpp
#include <console_bridge/console.h>
#include <tesseract_geometry/geometries.h>
#include <iostream>

using namespace tesseract_geometry;

int main(int /*argc*/, char** /*argv*/)
{
  // Shape Box
  auto box = std::make_shared<tesseract_geometry::Box>(1, 1, 1);
  // Shape Cone
  auto cone = std::make_shared<tesseract_geometry::Cone>(1, 1);
  // Shape Capsule
  auto capsule = std::make_shared<tesseract_geometry::Capsule>(1, 1);
  // Shape Cylinder
  auto cylinder = std::make_shared<tesseract_geometry::Cylinder>(1, 1);
  // Shape Plane
  auto plane = std::make_shared<tesseract_geometry::Plane>(1, 1, 1, 1);
  // Shape Sphere
  auto sphere = std::make_shared<tesseract_geometry::Sphere>(1);
```

(continues on next page)

(continued from previous page)

```
  // Manually create mesh
  std::shared_ptr<const tesseract_common::VectorVector3d> mesh_vertices =
      std::make_shared<const tesseract_common::VectorVector3d>();
  std::shared_ptr<const Eigen::VectorXi> mesh_faces = std::make_shared<const␣
↪Eigen::VectorXi>();
  // Next fill out vertices and triangles
  auto mesh = std::make_shared<tesseract_geometry::Mesh>(mesh_vertices, mesh_
↪faces);

  // Manually create signed distance field mesh
  std::shared_ptr<const tesseract_common::VectorVector3d> sdf_vertices =
      std::make_shared<const tesseract_common::VectorVector3d>();
  std::shared_ptr<const Eigen::VectorXi> sdf_faces = std::make_shared<const␣
↪Eigen::VectorXi>();
  // Next fill out vertices and triangles
  auto sdf_mesh = std::make_shared<tesseract_geometry::SDFMesh>(sdf_vertices,␣
↪sdf_faces);

  // Manually create convex mesh
  std::shared_ptr<const tesseract_common::VectorVector3d> convex_vertices =
      std::make_shared<const tesseract_common::VectorVector3d>();
  std::shared_ptr<const Eigen::VectorXi> convex_faces = std::make_shared<const␣
↪Eigen::VectorXi>();
  // Next fill out vertices and triangles
  auto convex_mesh = std::make_shared<tesseract_geometry::ConvexMesh>(convex_
↪vertices, convex_faces);

  // Create an octree
  std::shared_ptr<const octomap::OcTree> octree;
  auto octree_t = std::make_shared<tesseract_geometry::Octree>(octree,␣
↪tesseract_geometry::Octree::SubType::BOX);
}
```

Example Explanation

1. Create a box.

```
  auto box = std::make_shared<tesseract_geometry::Box>(1, 1, 1);
```

2. Create a cone.

```
  auto cone = std::make_shared<tesseract_geometry::Cone>(1, 1);
```

3. Create a capsule.

```
  auto capsule = std::make_shared<tesseract_geometry::Capsule>(1, 1);
```

4. Create a cylinder.

```
  auto cylinder = std::make_shared<tesseract_geometry::Cylinder>(1, 1);
```

5. Create a plane.

```
auto plane = std::make_shared<tesseract_geometry::Plane>(1, 1, 1, 1);
```

6. Create a sphere.

```
auto sphere = std::make_shared<tesseract_geometry::Sphere>(1);
```

7. Create a mesh.

```
std::shared_ptr<const tesseract_common::VectorVector3d> mesh_vertices =
    std::make_shared<const tesseract_common::VectorVector3d>();
std::shared_ptr<const Eigen::VectorXi> mesh_faces = std::make_shared<const␣
↪Eigen::VectorXi>();
// Next fill out vertices and triangles
auto mesh = std::make_shared<tesseract_geometry::Mesh>(mesh_vertices, mesh_faces);
```

Note: This shows how to create a mesh provided vertices and faces. You may also use utilities in tesseract_scene_graph mesh parser to load meshes from file.

---

8. Create a signed distance field mesh.

Note: This should be the same as a mesh, but when interpreted as the collision object it will be encoded as a signed distance field.

```
std::shared_ptr<const tesseract_common::VectorVector3d> sdf_vertices =
    std::make_shared<const tesseract_common::VectorVector3d>();
std::shared_ptr<const Eigen::VectorXi> sdf_faces = std::make_shared<const␣
↪Eigen::VectorXi>();
// Next fill out vertices and triangles
auto sdf_mesh = std::make_shared<tesseract_geometry::SDFMesh>(sdf_vertices, sdf_
↪faces);
```

Note: This shows how to create a SDF mesh provided vertices and faces. You may also use utilities in tesseract_scene_graph mesh parser to load meshes from file.

---

9. Create a convex mesh.

Warning: This expects the data to already represent a convex mesh. If yours does not load as a mesh and then use tesseract utility to convert to a convex mesh.

```
std::shared_ptr<const tesseract_common::VectorVector3d> convex_vertices =
    std::make_shared<const tesseract_common::VectorVector3d>();
std::shared_ptr<const Eigen::VectorXi> convex_faces = std::make_shared<const␣
↪Eigen::VectorXi>();
// Next fill out vertices and triangles
auto convex_mesh = std::make_shared<tesseract_geometry::ConvexMesh>(convex_
↪vertices, convex_faces);
```

Note: This shows how to create a convex mesh provided vertices and faces. You may also use utilities in tesseract_scene_graph mesh parser to load meshes from file.

10. Create an octree.

```
  std::shared_ptr<const octomap::OcTree> octree;
  auto octree_t = std::make_shared<tesseract_geometry::Octree>(octree, tesseract_
↪geometry::Octree::SubType::BOX);
```

Note: It is beneficial to prune the octree prior to creating the tesseract octree shape to simplify

Octree support multiple shape types to represent a cell in the octree.

- BOX **tesseract_geometry::Octree::SubType::BOX**
- SPHERE_INSIDE **tesseract_geometry::Octree::SubType::SPHERE_INSIDE**
- SPHERE_OUTSIDE **tesseract_geometry::Octree::SubType::SPHERE_OUTSIDE**

1.8.6 Tesseract Kinematics Package

This package contains a common interface for Forward and Inverse kinematics for Chain, Tree's and Graphs including implementation using KDL and OPW Kinematics.

Note: The kinematics are load as plugins through a yaml config file which is added to the SRDF file.

Kinematics Plugin Config

The kinematics config file has four sections:

Section	Description
search_paths	A list of locations to search for libraries. It searches in the order provided.
search_libraries	A list of libraries to search for classes
fwd_kin_plugins	A map of group names to list of forward kinematic plugins.
inv_kin_plugins	A map of group names to list of inverse kinematic plugins.

Example Config file:

```
kinematic_plugins:
  search_paths:
    - /usr/local/lib
  search_libraries:
    - tesseract_kinematics_kdl_factories
  fwd_kin_plugins:
    iiwa_manipulator:
      default: KDLFwdKinChain
      plugins:
        KDLFwdKinChain:
```

(continues on next page)

(continued from previous page)

```
        class: KDLFwdKinChainFactory
        config:
          base_link: base_link
          tip_link: tool0
inv_kin_plugins:
  iiwa_manipulator:
    default: KDLInvKinChainLMA
    plugins:
      KDLInvKinChainLMA:
        class: KDLInvKinChainLMAFactory
        config:
          base_link: base_link
          tip_link: tool0
      KDLInvKinChainNR:
        class: KDLInvKinChainNRFactory
        config:
          base_link: base_link
          tip_link: tool0
  abb_manipulator:
    default: OPWInvKin
    plugins:
      OPWInvKin:
        class: OPWInvKinFactory
        config:
          base_link: base_link
          tip_link: tool0
          params:
            a1: 0.100
            a2: -0.135
            b: 0.00
            c1: 0.615
            c2: 0.705
            c3: 0.755
            c4: 0.086
            offsets: [0, 0, -1.57079632679, 0, 0, 0]
            sign_corrections: [1, 1, 1, 1, 1, 1]
  ur_manipulator:
    default: URInvKin
    plugins:
      URInvKin:
        class: URInvKinFactory
        config:
          base_link: base_link
          tip_link: tool0
          model: UR10 #Current Options: UR3, UR5, UR10, UR3e, UR5e, UR10e
  rop_manipulator:
    default: ROPInvKin
    plugins:
      ROPInvKin:
        class: ROPInvKinFactory
        config:
          manipulator_reach: 2.0
```

(continues on next page)

(continued from previous page)

```
          positioner_sample_resolution:
            - name: positioner_joint_1
              value: 0.1
          positioner:
            class: KDLFwdKinChainFactory
            config:
              base_link: positioner_base_link
              tip_link: positioner_tool0
          manipulator:
            class: OPWInvKinFactory
            config:
              base_link: base_link
              tip_link: tool0
              params:
                a1: 0.100
                a2: -0.135
                b: 0.00
                c1: 0.615
                c2: 0.705
                c3: 0.755
                c4: 0.086
                offsets: [0, 0, -1.57079632679, 0, 0, 0]
                sign_corrections: [1, 1, 1, 1, 1, 1]
  rep_manipulator:
    default: REPInvKin
    plugins:
      REPInvKin:
        class: REPInvKinFactory
        config:
          manipulator_reach: 2.0
          positioner_sample_resolution:
            - name: positioner_joint_1
              value: 0.1
            - name: positioner_joint_2
              value: 0.1
          positioner:
            class: KDLFwdKinChainFactory
            config:
              base_link: positioner_base_link
              tip_link: positioner_tool0
          manipulator:
            class: OPWInvKinFactory
            config:
              base_link: base_link
              tip_link: tool0
              params:
                a1: 0.100
                a2: -0.135
                b: 0.00
                c1: 0.615
                c2: 0.705
                c3: 0.755
```

(continues on next page)

(continued from previous page)

```
                c4: 0.086
                offsets: [0, 0, -1.57079632679, 0, 0, 0]
                sign_corrections: [1, 1, 1, 1, 1, 1]
```

KDL Forward Kinematic Plugin

```
kinematic_plugins:
  fwd_kin_plugins:
    manipulator:
      default: KDLFwdKinChain
      plugins:
        KDLFwdKinChain:
          class: KDLFwdKinChainFactory
          config:
            base_link: base_link
            tip_link: tool0
```

KDL Inverse Kinematic Plugin

```
kinematic_plugins:
  inv_kin_plugins:
    iiwa_manipulator:
      default: KDLInvKinChainLMA
      plugins:
        KDLInvKinChainLMA:
          class: KDLInvKinChainLMAFactory
          config:
            base_link: base_link
            tip_link: tool0
        KDLInvKinChainNR:
          class: KDLInvKinChainNRFactory
          config:
            base_link: base_link
            tip_link: tool0
```

OPW Inverse Kinematic Plugin

```
kinematic_plugins:
  inv_kin_plugins:
    manipulator:
      default: OPWInvKin
      plugins:
        OPWInvKin:
          class: OPWInvKinFactory
          config:
            base_link: base_link
            tip_link: tool0
```

(continues on next page)

(continued from previous page)

```
            params:
              a1: 0.100
              a2: -0.135
              b: 0.00
              c1: 0.615
              c2: 0.705
              c3: 0.755
              c4: 0.086
              offsets: [0, 0, -1.57079632679, 0, 0, 0]
              sign_corrections: [1, 1, 1, 1, 1, 1]
```

UR Inverse Kinematic Plugin

Using preconfigured parameters:

```
kinematic_plugins:
  inv_kin_plugins:
    manipulator:
      default: URInvKin
      plugins:
        URInvKin:
          class: URInvKinFactory
          config:
            base_link: base_link
            tip_link: tool0
            model: UR10 #Current Options: UR3, UR5, UR10, UR3e, UR5e, UR10e
```

Using userdefined parameters:

```
kinematic_plugins:
  inv_kin_plugins:
    manipulator:
      default: URInvKin
      plugins:
        URInvKin:
          class: URInvKinFactory
          config:
            base_link: base_link
            tip_link: tool0
            params:
              d1: 0.1273
              a2: -0.612
              a3: -0.5723
              d4: 0.163941
              d5: 0.1157
              d6: 0.0922
```

Robot On Positioner (ROP) Inverse Kinematic Plugin

```yaml
kinematic_plugins:
  inv_kin_plugins:
    manipulator:
      default: ROPInvKin
      plugins:
        ROPInvKin:
          class: ROPInvKinFactory
          config:
            manipulator_reach: 2.0
            positioner_sample_resolution:
              - name: positioner_joint_1
                value: 0.1
            positioner:
              class: KDLFwdKinChainFactory
              config:
                base_link: positioner_base_link
                tip_link: positioner_tool0
            manipulator:
              class: OPWInvKinFactory
              config:
                base_link: base_link
                tip_link: tool0
                params:
                  a1: 0.100
                  a2: -0.135
                  b: 0.00
                  c1: 0.615
                  c2: 0.705
                  c3: 0.755
                  c4: 0.086
                  offsets: [0, 0, -1.57079632679, 0, 0, 0]
                  sign_corrections: [1, 1, 1, 1, 1, 1]
```

Robot wit External Positioner (REP) Inverse Kinematic Plugin

```yaml
kinematic_plugins:
  inv_kin_plugins:
    manipulator:
      default: REPInvKin
      plugins:
        REPInvKin:
          class: REPInvKinFactory
          config:
            manipulator_reach: 2.0
            positioner_sample_resolution:
              - name: positioner_joint_1
                value: 0.1
              - name: positioner_joint_2
                value: 0.1
```

(continues on next page)

(continued from previous page)

```
        positioner:
          class: KDLFwdKinChainFactory
          config:
            base_link: positioner_base_link
            tip_link: positioner_tool0
        manipulator:
          class: OPWInvKinFactory
          config:
            base_link: base_link
            tip_link: tool0
            params:
              a1: 0.100
              a2: -0.135
              b: 0.00
              c1: 0.615
              c2: 0.705
              c3: 0.755
              c4: 0.086
              offsets: [0, 0, -1.57079632679, 0, 0, 0]
              sign_corrections: [1, 1, 1, 1, 1, 1]
```

Creating IKFast Plugin

Prerequisites

1. Install docker and add a user group with appropriate permissions.
2. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
3. sudo apt-key fingerprint 0EBFCD88
4. sudo add-apt-repository -y "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
5. sudo apt update
6. sudo apt install -y docker-ce
7. sudo groupadd docker
8. sudo usermod -aG docker $USER
9. Verify that docker is installed correctly by running a test container using docker run hello-world. If this produces an error like "docker: Got permission denied while trying to connect to the Docker daemon socket," try rebooting your computer. This problem is caused by the docker usergroup not existing or not having sufficient permissions.
10. Grab the ikfast docker container we'll be using (you can take a look at what goes into it here): `docker pull hamzamerzic/openrave`

Converting URDF to .dae

OpenRAVE uses the .dae file format, so we'll need to convert our URDF to .dae. Replace catkin_ws with the path to the workspace containing the URDF, replace /path/to/dir/my_robot.urdf with the actual path to the URDF, and replace /path/to/dir/robot_full.dae with the filename and path where you want the new .dae to be written. I've called it robot_full.dae to emphasize that the transforms in it have an unnecessary degree of precision, but we'll rectify this in a later step

```
sudo apt install ros-<rosdistro>-collada-urdf
source catkin_ws/devel/setup.bash
rosrun collada_urdf urdf_to_collada /path/to/dir/my_robot.urdf /path/to/dir/robot_full.
↪dae
# Round off the precision to 6 decimal places:
rosrun moveit_kinematics round_collada_numbers.py /path/to/dir/robot_full.dae /path/to/
↪dir/robot.dae 6
```

Find Robot Link Indices

The robot links in the .dae are assigned indices, which can be somewhat opaque. OpenRAVE provides a utility to get info about the .dae, which we can access through the docker container. You'll need to map the container's /out directory to the folder containing your robot.dae, and pass the name of robot.dae as an argument to the Python script being run by the docker container.

```
docker run --rm --env PYTHONPATH=/usr/local/lib/python2.7/dist-packages -v /path/to/dir:/
↪out hamzamerzic/openrave /bin/bash -c "cd /out; openrave-robot.py robot.dae --info␣
↪links"
```

The output should look like this, with link names dependent on the names in the URDF:

```
name                        index parents

-------------------------------------------

world                       0

base_link                   1     world

base                        2     base_link

shoulder_link               3     base_link

upper_arm_link              4     shoulder_link

forearm_link                5     upper_arm_link

wrist_1_link                6     forearm_link

wrist_2_link                7     wrist_1_link

wrist_3_link                8     wrist_2_link

tool0                       9    wrist_3_link
```

(continues on next page)

(continued from previous page)

```
ee_link                      10    tool0

tool_control_point           11    ee_link

-----------------------------------------

name                         index parents
```

Run the IKFast Plugin Generator

Now we have enough information to compose the command to the ikfast generator script in the docker container:

```
docker run --rm --env PYTHONPATH=/usr/local/lib/python2.7/dist-packages -v /path/to/dir:/
↪out hamzamerzic/openrave /bin/bash -c "cd /out; python /usr/local/lib/python2.7/dist-
↪packages/openravepy/_openravepy_/ikfast.py --robot=robot.dae --iktype=transform6d --
↪baselink=1 --eelink=9 --savefile=robot_ikfast.cpp"
```

More information about the arguments for this script can be found in the [openravepy documentation](http://openrave.org/docs/0.8.2/openravepy/ikfast/). The important ones here are:

```
--robot: The name of the robot's .dae file in the path mapped to the container's /out␣
↪directory.

--iktype: The inverse kinematics model used to solve the kinematic chain. Since we have␣
↪a 6-dof robot the transform6d solver is most appropriate, but others are available for␣
↪different cases.

--baselink: The index of the robot's base link, from the table generated in the previous␣
↪step.

--eelink: The index of the robot's end effector link, also from the table. We usually␣
↪set this to correspond to the tool0 link, since the transform to the TCP frame can be␣
↪set outside the ikfast solver.

--savefile: The filename for the output .cpp file.
```

Another potentially-important argument not used here is –freeindex. If your robot has more than 6 DOFs, such as the 7-axis Kuka iiwa7 or a 6-DOF robot mounted on a rail, you'll need to pick a link that will have its position explicitly set prior to solving inverse kinematics. In the example above if I wanted to set the wrist_3_link to be a free axis I would add the argument –freeindex=8. I haven't tried this personally yet but I think that multiple free indices can be specified using, for example, –freeindex=[7,8].

This command might take a while to run (up to about 20 minutes) depending on the arrangement of the kinematic chain, and will produce a lot of output. The result will be a several-tens-of-thousand-LOC C++ source file containing the ikfast kinematic solver plugin for your robot.

Create Tesseract IKFast Solver

Header file:

```
#include <Eigen/Geometry>
#include <vector>
#include <tesseract_kinematics/ikfast/ikfast_inv_kin.h>

namespace fanuc_p50ib_15_ikfast_wrapper
{
class FanucP50iBInvKinematics : public tesseract_kinematics::IKFastInvKin
{
public:
  FanucP50iBInvKinematics(const std::string base_link_name,
                          const std::string tip_link_name,
                          const std::vector<std::string> joint_names,
                          const std::string name)
};
```

Source file:

The order of the includes matter.

```
#include <tesseract_kinematics/ikfast/impl/ikfast_inv_kin.hpp>
#include <fanuc_p50ib_15_ikfast_wrapper/impl/fanuc_p50ib_15_ikfast.hpp>
#include <fanuc_p50ib_15_ikfast_wrapper/tesseract_fanuc_p50ib_kinematics.h>

namespace fanuc_p50ib_15_ikfast_wrapper
{
  FanucP50iBInvKinematics::FanucP50iBInvKinematics(const std::string base_link_name,
                                                   const std::string tip_link_name,
                                                   const std::vector<std::string> joint_
↪names
                                                   const std::string name)
  : FanucP50iBInvKinematics(base_link_name, tip_link_name, joint_names, name, joint_
↪limits)
  {}
}
```

Create Tesseract IKFast Plugin

Header file:

```
#include <tesseract_kinematics/core/kinematics_plugin_factory.h>
namespace fanuc_p50ib_15_ikfast_wrapper
{
  class FanucP50iBInvKinFactory : public tesseract_kinematics::InvKinFactory
  {
    tesseract_kinematics::InverseKinematics::UPtr create(const std::string& solver_name,
```

(continues on next page)

(continued from previous page)

```
                                                                const tesseract_scene_
↪graph::SceneGraph& scene_graph,
                                                                const tesseract_scene_
↪graph::SceneState& scene_state,
                                                                const tesseract_
↪kinematics::KinematicsPluginFactory& plugin_factory,
                                                                const YAML::Node& config) const␣
↪override final;
  };
}
```

Source file:

```
#include <fanuc_p50ib_15_ikfast_wrapper/tesseract_fanuc_p50ib_kinematics.h>
#include <fanuc_p50ib_15_ikfast_wrapper/tesseract_fanuc_p50ib_factory.h>

namespace fanuc_p50ib_15_ikfast_wrapper
{
  tesseract_kinematics::InverseKinematics::UPtr
  FanucP50iBInvKinFactory::create(const std::string& solver_name,
                                  const tesseract_scene_graph::SceneGraph& scene_graph,
                                  const tesseract_scene_graph::SceneState& /*scene_
↪state*/,
                                  const tesseract_kinematics::KinematicsPluginFactory& /
↪*plugin_factory*/,
                                  const YAML::Node& config) const
  {
    std::string base_link;
    std::string tip_link;

    try
    {
      if (YAML::Node n = config["base_link"])
        base_link = n.as<std::string>();
      else
        throw std::runtime_error("KDLInvKinChainLMAFactory, missing 'base_link' entry");

      if (YAML::Node n = config["tip_link"])
        tip_link = n.as<std::string>();
      else
        throw std::runtime_error("KDLInvKinChainLMAFactory, missing 'tip_link' entry");
    }
    catch (const std::exception& e)
    {
      CONSOLE_BRIDGE_logError("KDLInvKinChainLMAFactory: Failed to parse yaml config␣
↪data! Details: %s", e.what());
      return nullptr;
    }

    auto shortest_path = scene_graph.getShortestPath(base_link, tip_link);
```

(continues on next page)

(continued from previous page)

```
    return std::make_unique<FanucP50iBInvKinematics>(base_link, tip_link, shortest_path.
↪active_joint_names);
  }
}

TESSERACT_ADD_PLUGIN(fanuc_p50ib_15_ikfast_wrapper::FanucP50iBInvKinFactory,␣
↪FanucP50iBInvKinFactory);
```

Add Additional items to CMakeLists.txt file:

```
find_package(tesseract_kinematics REQUIRED)
find_package(Eigen3 REQUIRED)
find_package(LAPACK REQUIRED) # Requried for ikfast

add_library(${PROJECT_NAME} src/tesseract_fanuc_p50ib_kinematics.cpp )
target_link_libraries(${PROJECT_NAME} PUBLIC tesseract::tesseract_kinematics_ikfast␣
↪console_bridge Eigen3::Eigen ${LAPACK_LIBRARIES})
target_include_directories(${PROJECT_NAME} PUBLIC
    "$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>"
    "$<INSTALL_INTERFACE:include>")
target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
    ${LAPACK_INCLUDE_DIRS})

add_library(${PROJECT_NAME}_factory src/tesseract_fanuc_p50ib_factory.cpp )
target_link_libraries(${PROJECT_NAME}_factory PUBLIC ${PROJECT_NAME}␣
↪tesseract::tesseract_kinematics_core)
target_include_directories(${PROJECT_NAME}_factory PUBLIC
    "$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>"
    "$<INSTALL_INTERFACE:include>")
```

Add Tesseract IKFast Plugin Config

Bellow shows an example kinematic plugin config file using the IKFast plugin.

```
kinematic_plugins:
  search_paths:
    - <path to your workspace lib directory>
  search_libraries:
    - <package_name>_factory
  fwd_kin_plugins:
    manipulator:
      default: KDLFwdKinChain
      plugins:
        KDLFwdKinChain:
          class: KDLFwdKinChainFactory
          config:
            base_link: base_link
            tip_link: tool0
  inv_kin_plugins:
```

(continues on next page)

(continued from previous page)

```
  manipulator:
    default: FanucP50iBInvKinematics
    plugins:
      KDLInvKinChainLMA:
        class: FanucP50iBInvKinFactory
        config:
          base_link: base_link
          tip_link: tool0
```

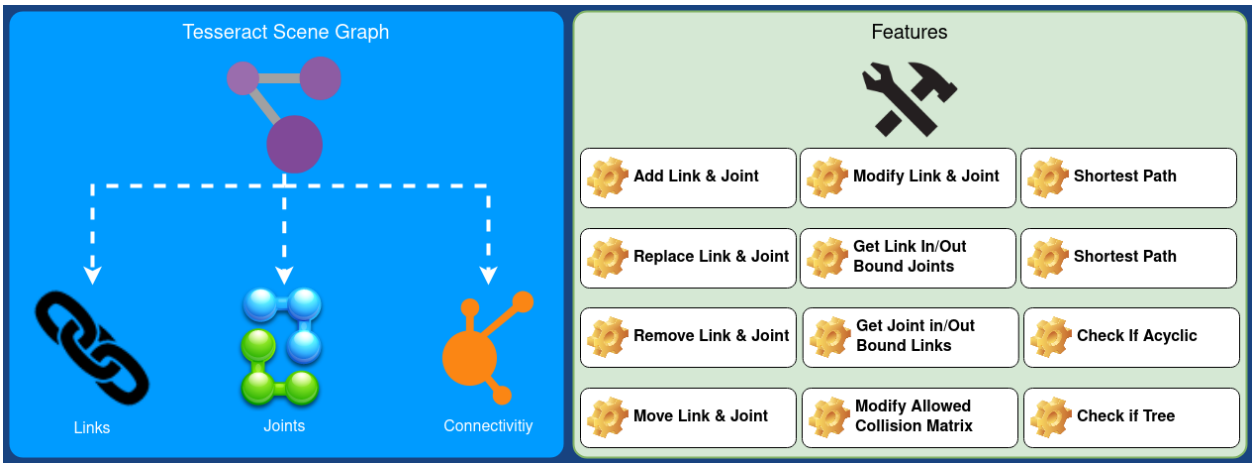
1.8.7 Tesseract Planners Package

This package contains a common interface for Planners and includes implementation for OMPL, TrajOpt and Descartes.

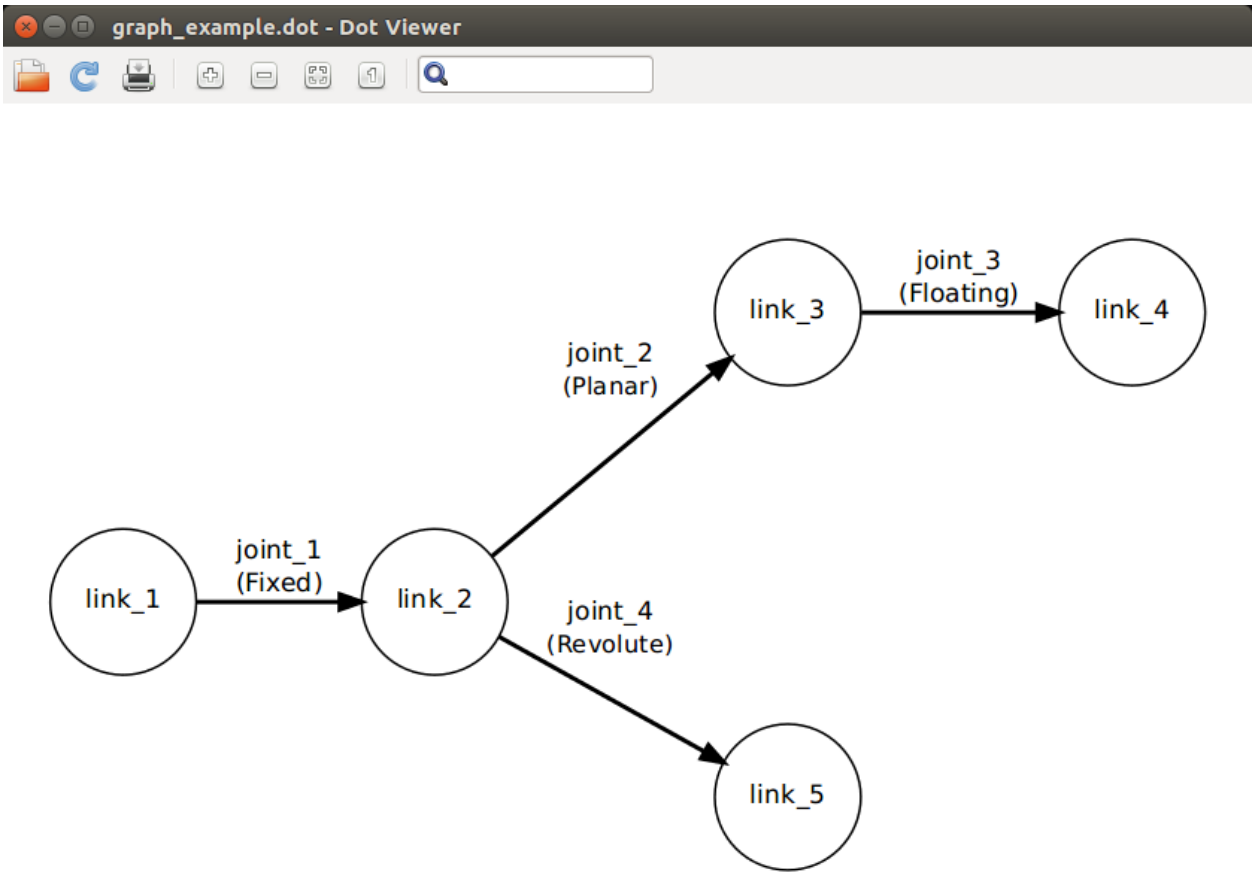
1.8.8 Tesseract Scene Graph Package

Background

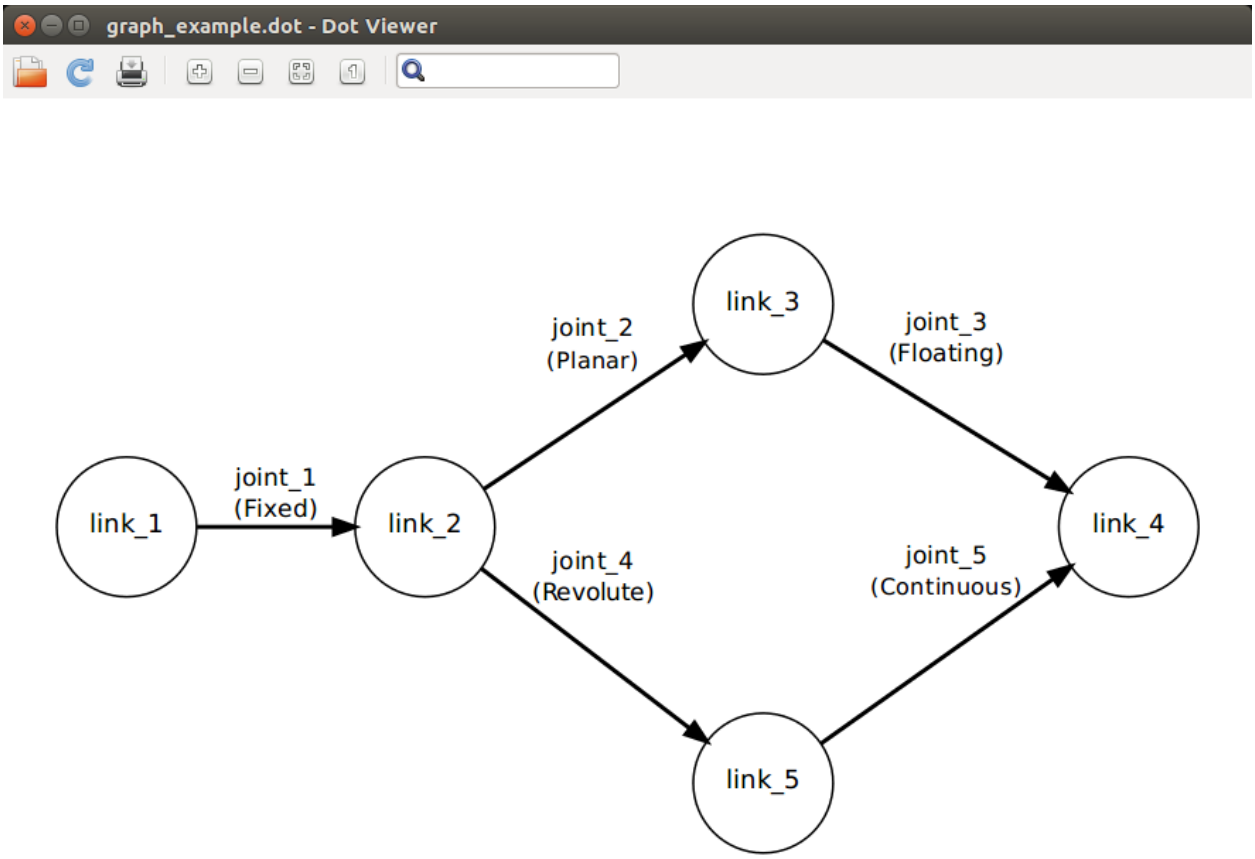
This package contains the scene graph and parsers. The scene graph is used to manage the connectivity of the environment. The scene graph inherits from boost graph so you are able to leverage boost graph utilities for searching.



Scene Graph (Tree)



Scene Graph (Acyclic)



Features

1. Links - Get, Add, Remove, Modify, Show/Hide, and Enable/Disable Collision
2. Joints - Get, Add, Remove, Move and Modify
3. Allowed Collision Matrix - Get, Add, Remove
4. Graph Functions
    - Get Inbound/Outbound Joints for Link
    - Check if acyclic
    - Check if tree
    - Get Adjacent/InvAdjacent Links for Joint
5. Utility Functions
    - Save to Graph to Graph Description Language (DOT)
    - Get shortest path between two Links

6. Parsers
    - URDF Parser
    - SRDF Parser
    - KDL Parser
    - Mesh Parser

Examples

1. *Building A Scene Graph*
2. *Create Scene Graph from URDF*
3. *Parse SRDF adding ACM to Scene Graph*
4. *Parse Mesh*

Building A Scene Graph

```
#include <tesseract_common/macros.h>
TESSERACT_COMMON_IGNORE_WARNINGS_PUSH
#include <console_bridge/console.h>
#include <iostream>
TESSERACT_COMMON_IGNORE_WARNINGS_POP

#include <tesseract_scene_graph/graph.h>
#include <tesseract_common/utils.h>

using namespace tesseract_scene_graph;

std::string toString(const ShortestPath& path)
{
  std::stringstream ss;
  ss << path;
  return ss.str();
}

std::string toString(bool b) { return b ? "true" : "false"; }

int main(int /*argc*/, char** /*argv*/)
{
  console_bridge::setLogLevel(console_bridge::LogLevel::CONSOLE_BRIDGE_LOG_INFO);

  // documentation:start:1: Create scene graph
  SceneGraph g;
  // documentation:end:1: Create scene graph

  // documentation:start:2: Create links
  Link link_1("link_1");
  Link link_2("link_2");
  Link link_3("link_3");
  Link link_4("link_4");
```

(continues on next page)

(continued from previous page)

```
  Link link_5("link_5");
  // documentation:end:2: Create links

  // documentation:start:3: Add links
  g.addLink(link_1);
  g.addLink(link_2);
  g.addLink(link_3);
  g.addLink(link_4);
  g.addLink(link_5);
  // documentation:end:3: Add links

  // documentation:start:4: Create joints
  Joint joint_1("joint_1");
  joint_1.parent_to_joint_origin_transform.translation()(0) = 1.25;
  joint_1.parent_link_name = "link_1";
  joint_1.child_link_name = "link_2";
  joint_1.type = JointType::FIXED;

  Joint joint_2("joint_2");
  joint_2.parent_to_joint_origin_transform.translation()(0) = 1.25;
  joint_2.parent_link_name = "link_2";
  joint_2.child_link_name = "link_3";
  joint_2.type = JointType::PLANAR;

  Joint joint_3("joint_3");
  joint_3.parent_to_joint_origin_transform.translation()(0) = 1.25;
  joint_3.parent_link_name = "link_3";
  joint_3.child_link_name = "link_4";
  joint_3.type = JointType::FLOATING;

  Joint joint_4("joint_4");
  joint_4.parent_to_joint_origin_transform.translation()(1) = 1.25;
  joint_4.parent_link_name = "link_2";
  joint_4.child_link_name = "link_5";
  joint_4.type = JointType::REVOLUTE;
  // documentation:end:4: Create joints

  // documentation:start:5: Add joints
  g.addJoint(joint_1);
  g.addJoint(joint_2);
  g.addJoint(joint_3);
  g.addJoint(joint_4);
  // documentation:end:5: Add joints

  // documentation:start:6: Check getAdjacentLinkNames Method
  std::vector<std::string> adjacent_links = g.getAdjacentLinkNames("link_3");
  for (const auto& adj : adjacent_links)
    CONSOLE_BRIDGE_logInform(adj.c_str());
  // documentation:end:6: Check getAdjacentLinkNames Method

  // documentation:start:7: Check getInvAdjacentLinkNames Method
  std::vector<std::string> inv_adjacent_links = g.getInvAdjacentLinkNames("link_3");
```

(continues on next page)

(continued from previous page)

```
  for (const auto& inv_adj : inv_adjacent_links)
    CONSOLE_BRIDGE_logInform(inv_adj.c_str());
  // documentation:end:7: Check getInvAdjacentLinkNames Method

  // documentation:start:8: Check getLinkChildrenNames
  std::vector<std::string> child_link_names = g.getLinkChildrenNames("link_2");
  for (const auto& child_link : child_link_names)
    CONSOLE_BRIDGE_logInform(child_link.c_str());
  // documentation:end:8: Check getLinkChildrenNames

  // documentation:start:9: Check getJointChildrenNames
  child_link_names = g.getJointChildrenNames("joint_1");
  for (const auto& child_link : child_link_names)
    CONSOLE_BRIDGE_logInform(child_link.c_str());
  // documentation:end:9: Check getJointChildrenNames

  // documentation:start:10: Save Graph
  g.saveDOT(tesseract_common::getTempPath() + "graph_acyclic_tree_example.dot");
  // documentation:end:10: Save Graph

  // documentation:start:11: Test if the graph is Acyclic
  bool is_acyclic = g.isAcyclic();
  CONSOLE_BRIDGE_logInform(toString(is_acyclic).c_str());
  // documentation:end:11: Test if the graph is Acyclic

  // documentation:start:12: Test if the graph is Tree
  bool is_tree = g.isTree();
  CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
  // documentation:end:12: Test if the graph is Tree

  // documentation:start:13: Test for unused links
  Link link_6("link_6");
  g.addLink(link_6);
  is_tree = g.isTree();
  CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
  // documentation:end:13: Test for unused links

  // documentation:start:14: Remove unused link
  g.removeLink("link_6");
  is_tree = g.isTree();
  CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
  // documentation:end:14: Remove unused link

  // documentation:start:15: Add new joint
  Joint joint_5("joint_5");
  joint_5.parent_to_joint_origin_transform.translation()(1) = 1.25;
  joint_5.parent_link_name = "link_5";
  joint_5.child_link_name = "link_4";
  joint_5.type = JointType::CONTINUOUS;
  g.addJoint(joint_5);
  // documentation:end:15: Add new joint
```

(continues on next page)

(continued from previous page)

```
  // documentation:start:16: Save new graph
  g.saveDOT(tesseract_common::getTempPath() + "graph_acyclic_not_tree_example.dot");
  // documentation:end:16: Save new graph

  // documentation:start:17: Test again if the graph is Acyclic
  is_acyclic = g.isAcyclic();
  CONSOLE_BRIDGE_logInform(toString(is_acyclic).c_str());
  // documentation:end:17: Test again if the graph is Acyclic

  // documentation:start:18: Test again if the graph is Tree
  is_tree = g.isTree();
  CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
  // documentation:end:18: Test again if the graph is Tree

  // documentation:start:19: Get Shortest Path
  ShortestPath path = g.getShortestPath("link_1", "link_4");
  CONSOLE_BRIDGE_logInform(toString(path).c_str());
  // documentation:end:19: Get Shortest Path
}
```

You can find this example https://github.com/tesseract-robotics/tesseract/blob/master/tesseract_scene_graph/examples/build_scene_graph_example.cpp

Example Explanation

Create Scene Graph

```
  SceneGraph g;
```

Add Links

Create the links. The links are able to be configured see Link documentation.

```
  Link link_1("link_1");
  Link link_2("link_2");
  Link link_3("link_3");
  Link link_4("link_4");
  Link link_5("link_5");
```

Add the links to the scene graph

```
  g.addLink(link_1);
  g.addLink(link_2);
  g.addLink(link_3);
  g.addLink(link_4);
  g.addLink(link_5);
```

Add Joints

Create the joints. The links are able to be configured see Joint documentation.

```
Joint joint_1("joint_1");
joint_1.parent_to_joint_origin_transform.translation()(0) = 1.25;
joint_1.parent_link_name = "link_1";
joint_1.child_link_name = "link_2";
joint_1.type = JointType::FIXED;

Joint joint_2("joint_2");
joint_2.parent_to_joint_origin_transform.translation()(0) = 1.25;
joint_2.parent_link_name = "link_2";
joint_2.child_link_name = "link_3";
joint_2.type = JointType::PLANAR;

Joint joint_3("joint_3");
joint_3.parent_to_joint_origin_transform.translation()(0) = 1.25;
joint_3.parent_link_name = "link_3";
joint_3.child_link_name = "link_4";
joint_3.type = JointType::FLOATING;

Joint joint_4("joint_4");
joint_4.parent_to_joint_origin_transform.translation()(1) = 1.25;
joint_4.parent_link_name = "link_2";
joint_4.child_link_name = "link_5";
joint_4.type = JointType::REVOLUTE;
```

Add the joints to the scene graph_acyclic_tree_example

```
g.addJoint(joint_1);
g.addJoint(joint_2);
g.addJoint(joint_3);
g.addJoint(joint_4);
```

Inspect Scene Graph

Get the adjacent links for **link_3** and print to terminal

```
std::vector<std::string> adjacent_links = g.getAdjacentLinkNames("link_3");
for (const auto& adj : adjacent_links)
  CONSOLE_BRIDGE_logInform(adj.c_str());
```

Get the inverse adjacent links for **link_3** and print to terminal

```
std::vector<std::string> inv_adjacent_links = g.getInvAdjacentLinkNames("link_3");
for (const auto& inv_adj : inv_adjacent_links)
  CONSOLE_BRIDGE_logInform(inv_adj.c_str());
```

Get child link names for link **link_3** and print to terminal

```
std::vector<std::string> child_link_names = g.getLinkChildrenNames("link_2");
for (const auto& child_link : child_link_names)
  CONSOLE_BRIDGE_logInform(child_link.c_str());
```

Get child link names for joint **joint_1** and print to terminal

```
child_link_names = g.getJointChildrenNames("joint_1");
for (const auto& child_link : child_link_names)
  CONSOLE_BRIDGE_logInform(child_link.c_str());
```

Save the graph to a file for visualization

```
g.saveDOT(tesseract_common::getTempPath() + "graph_acyclic_tree_example.dot");
```

Test if the graph is Acyclic and print to terminal

```
bool is_acyclic = g.isAcyclic();
CONSOLE_BRIDGE_logInform(toString(is_acyclic).c_str());
```

Test if the graph is a tree and print to terminal

```
bool is_tree = g.isTree();
CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
```

Detect Unused Links

First add a link but do not create joint and check if it is a tree. It should return false because the link is not associated with a joint.

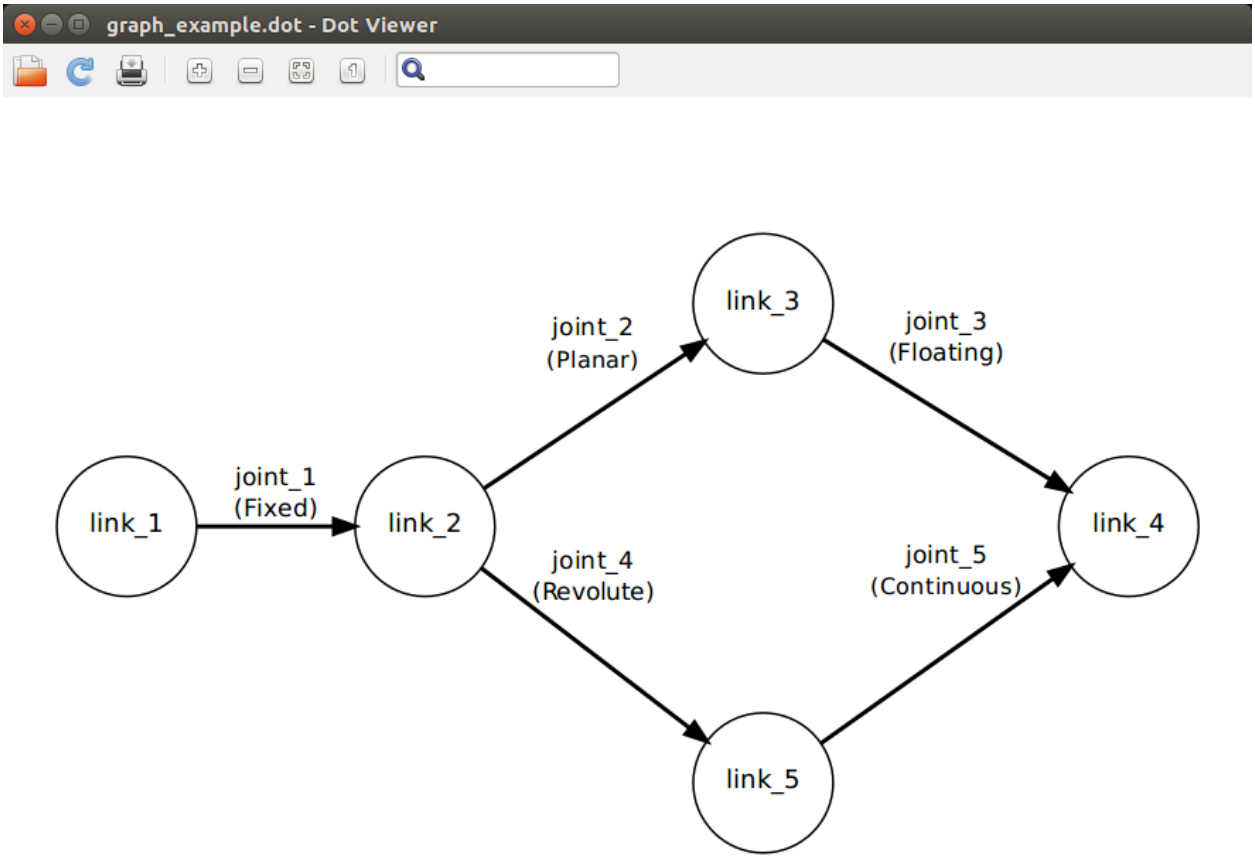
```
Link link_6("link_6");
g.addLink(link_6);
is_tree = g.isTree();
CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
```

Remove link and check if it is a tree. It should return true.

```
g.removeLink("link_6");
is_tree = g.isTree();
CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
```

Create Acyclic Graph

Add joint connecting **link_5** and **link_4** to create an Acyclic graph_acyclic_tree_example




```
Joint joint_5("joint_5");
joint_5.parent_to_joint_origin_transform.translation()(1) = 1.25;
joint_5.parent_link_name = "link_5";
joint_5.child_link_name = "link_4";
joint_5.type = JointType::CONTINUOUS;
g.addJoint(joint_5);
```

Save the Acyclic graph

```
g.saveDOT(tesseract_common::getTempPath() + "graph_acyclic_not_tree_example.dot");
```

Test to confirm it is acyclic, should return true.

```
is_acyclic = g.isAcyclic();
CONSOLE_BRIDGE_logInform(toString(is_acyclic).c_str());
```

Test if it is a tree, should return false.

```
is_tree = g.isTree();
CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
```

Get Shortest Path

```
  ShortestPath path = g.getShortestPath("link_1", "link_4");
  CONSOLE_BRIDGE_logInform(toString(path).c_str());
```

Running the Example

Build the Tesseract Workspace:

```
catkin build
```

Navigate to the build folder containing the executable:

```
cd build/tesseract_scene_graph/examples
```

Run the executable:

```
./tesseract_scene_graph_build_graph_example
```

Create Scene Graph from URDF

```
#include <tesseract_common/macros.h>
TESSERACT_COMMON_IGNORE_WARNINGS_PUSH
#include <iostream>
#include <console_bridge/console.h>
#include <tesseract_scene_graph/graph.h>
#include <tesseract_common/utils.h>
TESSERACT_COMMON_IGNORE_WARNINGS_POP

#include <tesseract_urdf/urdf_parser.h>
#include <tesseract_support/tesseract_support_resource_locator.h>

using namespace tesseract_scene_graph;
using namespace tesseract_urdf;

std::string toString(const ShortestPath& path)
{
  std::stringstream ss;
  ss << path;
  return ss.str();
}

std::string toString(bool b) { return b ? "true" : "false"; }

int main(int /*argc*/, char** /*argv*/)
{
  // documentation:start:2: Get the urdf file path
  std::string urdf_file = std::string(TESSERACT_SUPPORT_DIR) + "/urdf/lbr_iiwa_14_r820.
↪urdf";
  // documentation:end:2: Get the urdf file path
```

(continues on next page)

(continued from previous page)

```
  // documentation:start:3: Create scene graph
  tesseract_common::TesseractSupportResourceLocator locator;
  SceneGraph::Ptr g = parseURDFFile(urdf_file, locator);
  // documentation:end:3: Create scene graph

  // documentation:start:4: Print information
  CONSOLE_BRIDGE_logInform(std::to_string(g->getJoints().size()).c_str());
  CONSOLE_BRIDGE_logInform(std::to_string(g->getLinks().size()).c_str());
  CONSOLE_BRIDGE_logInform(toString(g->isTree()).c_str());
  CONSOLE_BRIDGE_logInform(toString(g->isAcyclic()).c_str());
  // documentation:end:4: Print information

  // documentation:start:5: Save graph
  g->saveDOT(tesseract_common::getTempPath() + "tesseract_urdf_import.dot");
  // documentation:end:5: Save graph
}
```

You can find this example https://github.com/tesseract-robotics/tesseract/blob/master/tesseract_urdf/examples/load_urdf_example.cpp

Example Explanation

Create Resource Locator

Because this is ROS agnostic you need to provide a resource locator for interpreting **package:/**.

Load URDF

Get the file path to the urdf file

```
  std::string urdf_file = std::string(TESSERACT_SUPPORT_DIR) + "/urdf/lbr_iiwa_14_r820.
↪urdf";
```

Create scene graph from urdf

```
  tesseract_common::TesseractSupportResourceLocator locator;
  SceneGraph::Ptr g = parseURDFFile(urdf_file, locator);
```

Print information about the scene graph to the terminal

```
  CONSOLE_BRIDGE_logInform(std::to_string(g->getJoints().size()).c_str());
  CONSOLE_BRIDGE_logInform(std::to_string(g->getLinks().size()).c_str());
  CONSOLE_BRIDGE_logInform(toString(g->isTree()).c_str());
  CONSOLE_BRIDGE_logInform(toString(g->isAcyclic()).c_str());
```

Save the graph to a file.

```
  g->saveDOT(tesseract_common::getTempPath() + "tesseract_urdf_import.dot");
```

Running the Example

Build the Tesseract Workspace:

```
catkin build
```

Navigate to the build folder containing the executable:

```
cd build/tesseract_urdf/examples
```

Run the executable:

```
./tesseract_urdf_load_urdf_example
```

Parse SRDF adding Allowed Collision Matrix to Graph

```
#include <console_bridge/console.h>
#include <tesseract_scene_graph/graph.h>
#include <tesseract_common/allowed_collision_matrix.h>
#include <tesseract_common/resource_locator.h>
#include <tesseract_srdf/srdf_model.h>
#include <tesseract_srdf/utils.h>
#include <iostream>
#include <tesseract_support/tesseract_support_resource_locator.h>

using namespace tesseract_scene_graph;
using namespace tesseract_srdf;

std::string toString(const ShortestPath& path)
{
  std::stringstream ss;
  ss << path;
  return ss.str();
}

std::string toString(bool b) { return b ? "true" : "false"; }

int main(int /*argc*/, char** /*argv*/)
{
  // documentation:start:1: Create scene graph
  SceneGraph g;

  g.setName("kuka_lbr_iiwa_14_r820");

  Link base_link("base_link");
  Link link_1("link_1");
  Link link_2("link_2");
  Link link_3("link_3");
  Link link_4("link_4");
  Link link_5("link_5");
  Link link_6("link_6");
  Link link_7("link_7");
```

(continues on next page)

(continued from previous page)

```
Link tool0("tool0");

g.addLink(base_link);
g.addLink(link_1);
g.addLink(link_2);
g.addLink(link_3);
g.addLink(link_4);
g.addLink(link_5);
g.addLink(link_6);
g.addLink(link_7);
g.addLink(tool0);

Joint base_joint("base_joint");
base_joint.parent_link_name = "base_link";
base_joint.child_link_name = "link_1";
base_joint.type = JointType::FIXED;
g.addJoint(base_joint);

Joint joint_1("joint_1");
joint_1.parent_link_name = "link_1";
joint_1.child_link_name = "link_2";
joint_1.type = JointType::REVOLUTE;
g.addJoint(joint_1);

Joint joint_2("joint_2");
joint_2.parent_to_joint_origin_transform.translation()(0) = 1.25;
joint_2.parent_link_name = "link_2";
joint_2.child_link_name = "link_3";
joint_2.type = JointType::REVOLUTE;
g.addJoint(joint_2);

Joint joint_3("joint_3");
joint_3.parent_to_joint_origin_transform.translation()(0) = 1.25;
joint_3.parent_link_name = "link_3";
joint_3.child_link_name = "link_4";
joint_3.type = JointType::REVOLUTE;
g.addJoint(joint_3);

Joint joint_4("joint_4");
joint_4.parent_to_joint_origin_transform.translation()(1) = 1.25;
joint_4.parent_link_name = "link_4";
joint_4.child_link_name = "link_5";
joint_4.type = JointType::REVOLUTE;
g.addJoint(joint_4);

Joint joint_5("joint_5");
joint_5.parent_to_joint_origin_transform.translation()(1) = 1.25;
joint_5.parent_link_name = "link_5";
joint_5.child_link_name = "link_6";
joint_5.type = JointType::REVOLUTE;
g.addJoint(joint_5);
```

(continues on next page)

(continued from previous page)

```
  Joint joint_6("joint_6");
  joint_6.parent_to_joint_origin_transform.translation()(1) = 1.25;
  joint_6.parent_link_name = "link_6";
  joint_6.child_link_name = "link_7";
  joint_6.type = JointType::REVOLUTE;
  g.addJoint(joint_6);

  Joint joint_tool0("joint_tool0");
  joint_tool0.parent_link_name = "link_7";
  joint_tool0.child_link_name = "tool0";
  joint_tool0.type = JointType::FIXED;
  g.addJoint(joint_tool0);
  // documentation:end:1: Create scene graph

  // documentation:start:2: Get the srdf file path
  tesseract_common::TesseractSupportResourceLocator locator;
  std::string srdf_file =
      locator.locateResource("package://tesseract_support/urdf/lbr_iiwa_14_r820.srdf")->
↪getFilePath();
  // documentation:end:2: Get the srdf file path

  // documentation:start:3: Parse the srdf
  SRDFModel srdf;
  try
  {
    srdf.initFile(g, srdf_file, locator);
  }
  catch (const std::exception& e)
  {
    CONSOLE_BRIDGE_logError("Failed to parse SRDF.");
    tesseract_common::printNestedException(e);
    return 1;
  }
  // documentation:end:3: Parse the srdf

  // documentation:start:4: Add allowed collision matrix to scene graph
  g.addAllowedCollision("link_1", "link_2", "adjacent");

  processSRDFAllowedCollisions(g, srdf);
  // documentation:end:4: Add allowed collision matrix to scene graph

  // documentation:start:5: Get info about allowed collision matrix
  tesseract_common::AllowedCollisionMatrix::ConstPtr acm = g.getAllowedCollisionMatrix();
  const tesseract_common::AllowedCollisionEntries& acm_entries = acm->
↪getAllAllowedCollisions();
  CONSOLE_BRIDGE_logInform("ACM Number of entries: %d", acm_entries.size());
  // documentation:end:5: Get info about allowed collision matrix
}
```

You can find this example at https://github.com/tesseract-robotics/tesseract/blob/master/tesseract_srdf/examples/parse_srdf_example.cpp

Example Explanation

Create Scene Graph

```
SceneGraph g;

g.setName("kuka_lbr_iiwa_14_r820");

Link base_link("base_link");
Link link_1("link_1");
Link link_2("link_2");
Link link_3("link_3");
Link link_4("link_4");
Link link_5("link_5");
Link link_6("link_6");
Link link_7("link_7");
Link tool0("tool0");

g.addLink(base_link);
g.addLink(link_1);
g.addLink(link_2);
g.addLink(link_3);
g.addLink(link_4);
g.addLink(link_5);
g.addLink(link_6);
g.addLink(link_7);
g.addLink(tool0);

Joint base_joint("base_joint");
base_joint.parent_link_name = "base_link";
base_joint.child_link_name = "link_1";
base_joint.type = JointType::FIXED;
g.addJoint(base_joint);

Joint joint_1("joint_1");
joint_1.parent_link_name = "link_1";
joint_1.child_link_name = "link_2";
joint_1.type = JointType::REVOLUTE;
g.addJoint(joint_1);

Joint joint_2("joint_2");
joint_2.parent_to_joint_origin_transform.translation()(0) = 1.25;
joint_2.parent_link_name = "link_2";
joint_2.child_link_name = "link_3";
joint_2.type = JointType::REVOLUTE;
g.addJoint(joint_2);

Joint joint_3("joint_3");
joint_3.parent_to_joint_origin_transform.translation()(0) = 1.25;
joint_3.parent_link_name = "link_3";
joint_3.child_link_name = "link_4";
joint_3.type = JointType::REVOLUTE;
```

(continues on next page)

(continued from previous page)

```
  g.addJoint(joint_3);

  Joint joint_4("joint_4");
  joint_4.parent_to_joint_origin_transform.translation()(1) = 1.25;
  joint_4.parent_link_name = "link_4";
  joint_4.child_link_name = "link_5";
  joint_4.type = JointType::REVOLUTE;
  g.addJoint(joint_4);

  Joint joint_5("joint_5");
  joint_5.parent_to_joint_origin_transform.translation()(1) = 1.25;
  joint_5.parent_link_name = "link_5";
  joint_5.child_link_name = "link_6";
  joint_5.type = JointType::REVOLUTE;
  g.addJoint(joint_5);

  Joint joint_6("joint_6");
  joint_6.parent_to_joint_origin_transform.translation()(1) = 1.25;
  joint_6.parent_link_name = "link_6";
  joint_6.child_link_name = "link_7";
  joint_6.type = JointType::REVOLUTE;
  g.addJoint(joint_6);

  Joint joint_tool0("joint_tool0");
  joint_tool0.parent_link_name = "link_7";
  joint_tool0.child_link_name = "tool0";
  joint_tool0.type = JointType::FIXED;
  g.addJoint(joint_tool0);
```

Load SRDF

Get the file path to the SRDF file

```
  tesseract_common::TesseractSupportResourceLocator locator;
  std::string srdf_file =
      locator.locateResource("package://tesseract_support/urdf/lbr_iiwa_14_r820.srdf")->
↪getFilePath();
```

Parse SRDF

```
  SRDFModel srdf;
  try
  {
    srdf.initFile(g, srdf_file, locator);
  }
  catch (const std::exception& e)
  {
    CONSOLE_BRIDGE_logError("Failed to parse SRDF.");
    tesseract_common::printNestedException(e);
    return 1;
  }
```

Add Allowed Collision Matrix to Scene Graph

```
  g.addAllowedCollision("link_1", "link_2", "adjacent");

  processSRDFAllowedCollisions(g, srdf);
```

Methods for getting Allowed Collision Matrix from Scene Graph

```
  tesseract_common::AllowedCollisionMatrix::ConstPtr acm = g.getAllowedCollisionMatrix();
  const tesseract_common::AllowedCollisionEntries& acm_entries = acm->
↪getAllAllowedCollisions();
  CONSOLE_BRIDGE_logInform("ACM Number of entries: %d", acm_entries.size());
```

Parse Mesh from file

```
#include <console_bridge/console.h>
#include <tesseract_geometry/impl/mesh.h>
#include <tesseract_geometry/mesh_parser.h>
#include <iostream>

using namespace tesseract_geometry;

int main(int /*argc*/, char** /*argv*/)
{
  // documentation:start:1: Create meshes
  std::string mesh_file = std::string(TESSERACT_SUPPORT_DIR) + "/meshes/sphere_p25m.dae";
  std::vector<Mesh::Ptr> meshes = createMeshFromPath<Mesh>(mesh_file);
  // documentation:end:1: Create meshes

  // documentation:start:2: Print mesh information
  CONSOLE_BRIDGE_logInform("Number of meshes: %f", meshes.size());
  CONSOLE_BRIDGE_logInform("Mesh #1 Triangle Count: %f", meshes[0]->getFaceCount());
  CONSOLE_BRIDGE_logInform("Mesh #1 Triangle Count: %f", meshes[0]->getVertexCount());
  CONSOLE_BRIDGE_logInform("Mesh #2 Triangle Count: %f", meshes[1]->getFaceCount());
  CONSOLE_BRIDGE_logInform("Mesh #2 Triangle Count: %f", meshes[1]->getVertexCount());
  // documentation:end:2: Print mesh information
}
```

Example Explanation

Parse Mesh from File

Mesh files can contain multiple meshes. This is a critical difference between MoveIt! which merges all shapes in to a single triangle list for collision checking. By keeping each mesh independent, each will have its own bounding box and if you want to convert to a convex hull you will get a closer representation of the geometry.

```
  std::string mesh_file = std::string(TESSERACT_SUPPORT_DIR) + "/meshes/sphere_p25m.dae";
  std::vector<Mesh::Ptr> meshes = createMeshFromPath<Mesh>(mesh_file);
```

Print Mesh Information to Terminal

```
  CONSOLE_BRIDGE_logInform("Number of meshes: %f", meshes.size());
  CONSOLE_BRIDGE_logInform("Mesh #1 Triangle Count: %f", meshes[0]->getFaceCount());
  CONSOLE_BRIDGE_logInform("Mesh #1 Triangle Count: %f", meshes[0]->getVertexCount());
  CONSOLE_BRIDGE_logInform("Mesh #2 Triangle Count: %f", meshes[1]->getFaceCount());
  CONSOLE_BRIDGE_logInform("Mesh #2 Triangle Count: %f", meshes[1]->getVertexCount());
```

Running the Example

Build the Tesseract Workspace:

```
catkin build
```

Navigate to the build folder containing the executable:

```
cd build/tesseract_geometry/examples
```

Run the executable:

```
./tesseract_geometry_parse_mesh_example
```

1.8.9 Tesseract SRDF Format

Background

Tesseract has its own SRDF format which is similar to the one used through ROS, but includes features specific to Tesseract.

Features

Feature	Description
Groups	Groups define collections of links and joints that are used for planning.
Group States	Define a state for a group by name for easy access during operation like planning.
Group Tool Center Points	Define a TCP for a group by name for easy access during operation like planning.
Kinematics Plugin Config	A config file defining kinematics plugins: (OPW, IKFast, KDL, ROP and REP)
Contact Manager Plugin Config	A config file defining contact manager plugins
Calibration Config	A config file defining calibration information
Collision Margins	Define default collision margin along with link pair collision margin data
Allowed Collisions	Define link pair allowed collisions

Example File

```
<robot name="abb_irb2400" version="1.0.0">
    <group name="manipulator_chain">
        <chain base_link="base_link" tip_link="tool0"/>
    </group>

    <group name="positioner">
      <chain base_link="world" tip_link="base_link" />
    </group>
```

(continues on next page)

(continued from previous page)

```
    <group name="gantry">
      <chain base_link="world" tip_link="tool0" />
    </group>

    <group name="manipulator_joints">
        <joint name="joint_1"/>
        <joint name="joint_2"/>
        <joint name="joint_3"/>
        <joint name="joint_4"/>
        <joint name="joint_5"/>
        <joint name="joint_6"/>
    </group>

    <group_state name="zeros" group="manipulator_joints">
        <joint name="joint_6" value="0"/>
        <joint name="joint_4" value="0"/>
        <joint name="joint_5" value="0"/>
        <joint name="joint_3" value="0"/>
        <joint name="joint_1" value="0"/>
        <joint name="joint_2" value="0"/>
    </group_state>

    <group_state name="zeros" group="manipulator_chain">
        <joint name="joint_6" value="0"/>
        <joint name="joint_4" value="0"/>
        <joint name="joint_5" value="0"/>
        <joint name="joint_3" value="0"/>
        <joint name="joint_1" value="0"/>
        <joint name="joint_2" value="0"/>
    </group_state>

    <group_tcps group="manipulator_chain">
        <tcp name="scanner" xyz="  0   0 0.2" wxyz="1 0 0 0"/>
    </group_tcps>

    <group_tcps group="manipulator_joints">
        <tcp name="scanner" xyz="  0   0 0.2" wxyz="1 0 0 0"/>
    </group_tcps>

    <!--Groups kinematic plugins -->
    <kinematics_plugin_config filename="package://tesseract_support/urdf/abb_irb2400_
↪plugins.yaml"/>

    <!--Contact Managers plugins -->
    <contact_managers_plugin_config filename="package://tesseract_support/urdf/contact_
↪manager_plugins.yaml"/>

    <!--Calibration config information -->
    <calibration_config filename="calibration_config.yaml"/>

    <collision_margins default_margin="0.025">
```

(continues on next page)

(continued from previous page)

```
      <pair_margin link1="link_6" link2="link_5" margin="0.01"/>
      <pair_margin link1="link_5" link2="link_4" margin="0.015"/>
    </collision_margins>

    <disable_collisions link1="link_3" link2="link_5" reason="Never"/>
    <disable_collisions link1="link_3" link2="link_6" reason="Never"/>
    <disable_collisions link1="link_2" link2="link_5" reason="Never"/>
    <disable_collisions link1="link_2" link2="link_4" reason="Never"/>
    <disable_collisions link1="link_4" link2="link_6" reason="Allways"/>
    <disable_collisions link1="link_1" link2="link_5" reason="Never"/>
    <disable_collisions link1="link_3" link2="link_4" reason="Adjacent"/>
    <disable_collisions link1="link_2" link2="link_3" reason="Adjacent"/>
    <disable_collisions link1="base_link" link2="link_1" reason="Adjacent"/>
    <disable_collisions link1="link_1" link2="link_2" reason="Adjacent"/>
    <disable_collisions link1="link_1" link2="link_4" reason="Never"/>
    <disable_collisions link1="base_link" link2="link_4" reason="Never"/>
    <disable_collisions link1="link_1" link2="link_6" reason="Never"/>
    <disable_collisions link1="link_5" link2="link_6" reason="Adjacent"/>
    <disable_collisions link1="base_link" link2="link_5" reason="Never"/>
    <disable_collisions link1="link_1" link2="link_3" reason="Never"/>
    <disable_collisions link1="base_link" link2="link_2" reason="Never"/>
    <disable_collisions link1="link_2" link2="link_6" reason="Never"/>
    <disable_collisions link1="link_4" link2="link_5" reason="Adjacent"/>
    <disable_collisions link1="base_link" link2="link_6" reason="Never"/>
    <disable_collisions link1="base_link" link2="link_3" reason="Never"/>
</robot>
```

Example Explanation

Create Chain Groups

A serial chain is specified using the base link and the tip link. The tip link in a chain is the child link of the last joint in the chain. The base link in a chain is the parent link for the first joint in the chain.

```
<group name="manipulator_chain">
    <chain base_link="base_link" tip_link="tool0"/>
</group>

<group name="positioner">
  <chain base_link="world" tip_link="base_link" />
</group>

<group name="gantry">
  <chain base_link="world" tip_link="tool0" />
</group>
```

Create Joint Groups

A group can be specified as a collection of joints. All the child links of each joint are automatically included in the group.

```
<group name="manipulator_joints">
    <joint name="joint_1"/>
    <joint name="joint_2"/>
    <joint name="joint_3"/>
    <joint name="joint_4"/>
    <joint name="joint_5"/>
    <joint name="joint_6"/>
</group>
```

Create Group States

Store fixed configurations of the robot. A typical use case is in defining a HOME position for a manipulator. The configuration is stored with a string id, which can be used to recover the configuration later.

```
<group_state name="zeros" group="manipulator_joints">
    <joint name="joint_6" value="0"/>
    <joint name="joint_4" value="0"/>
    <joint name="joint_5" value="0"/>
    <joint name="joint_3" value="0"/>
    <joint name="joint_1" value="0"/>
    <joint name="joint_2" value="0"/>
</group_state>

<group_state name="zeros" group="manipulator_chain">
    <joint name="joint_6" value="0"/>
    <joint name="joint_4" value="0"/>
    <joint name="joint_5" value="0"/>
    <joint name="joint_3" value="0"/>
    <joint name="joint_1" value="0"/>
    <joint name="joint_2" value="0"/>
</group_state>
```

Create Group Tool Center Points

Store fixed tool center point definitions by string id, which can be used to recover the tool center point during operation like planning.

```
<group_tcps group="manipulator_chain">
    <tcp name="scanner" xyz="  0   0 0.2" wxyz="1 0 0 0"/>
</group_tcps>

<group_tcps group="manipulator_joints">
    <tcp name="scanner" xyz="  0   0 0.2" wxyz="1 0 0 0"/>
</group_tcps>
```

Add Kinematics Plugin Config

Add an entry to the SRDF for loading a yaml config file defining the kinematics plugins.

Note: Reference the kinematics documentation on the creation of the yaml file.

```
<kinematics_plugin_config filename="kinematics_plugin_config.yaml"/>
```

Add Contact Manager Plugin Config

Add an entry to the SRDF for loading a yaml config file defining the contact manager plugins.

Note: Reference the collision documentation on the creation of the yaml file.

```
<contact_managers_plugin_config filename="contact_managers_plugin_config.yaml"/>
```

Add Calibration Config

Add an entry to the SRDF for loading a yaml config file defining the calibration information. The config file defines new origin for joints.

```
<calibration_config filename="calibration_config.yaml"/>
```

Example Config File: *calibration_config.yaml*

```
calibration:
  joints:
    joint_1:
      position:
        x: 1
        y: 2
        z: 3
      orientation:
        x: 0
        y: 0
        z: 0
        w: 1
    joint_2:
      position:
        x: 4
        y: 5
        z: 6
      orientation:
        x: 0
        y: 0
        z: 0
        w: 1
```

Define Collision Margin Data

In most industrial applications a single contact margin distance is not sutable because there are objects that constantly work within close proximity. This would limit the contact distance to be smaller than desired for other links allowing all objects to operate close to one another. The Tesseract contact checkers allow for a default margin to be defined along with link pair collision margins elimating this issue. This is configurable from within the SRDF file shown below.

```
<collision_margins default_margin="0.025">
  <pair_margin link1="link_6" link2="link_5" margin="0.01"/>
  <pair_margin link1="link_5" link2="link_4" margin="0.015"/>
</collision_margins>
```

Define Allowed Collision

Define link pairs that are allowed to be in collision with each other. This is used during contact checking to avoid checking links that are allowed to be in collision and contact data should not be calculated.

```
<disable_collisions link1="link_3" link2="link_5" reason="Never"/>
<disable_collisions link1="link_3" link2="link_6" reason="Never"/>
<disable_collisions link1="link_2" link2="link_5" reason="Never"/>
<disable_collisions link1="link_2" link2="link_4" reason="Never"/>
<disable_collisions link1="link_4" link2="link_6" reason="Allways"/>
<disable_collisions link1="link_1" link2="link_5" reason="Never"/>
<disable_collisions link1="link_3" link2="link_4" reason="Adjacent"/>
<disable_collisions link1="link_2" link2="link_3" reason="Adjacent"/>
<disable_collisions link1="base_link" link2="link_1" reason="Adjacent"/>
<disable_collisions link1="link_1" link2="link_2" reason="Adjacent"/>
<disable_collisions link1="link_1" link2="link_4" reason="Never"/>
<disable_collisions link1="base_link" link2="link_4" reason="Never"/>
<disable_collisions link1="link_1" link2="link_6" reason="Never"/>
<disable_collisions link1="link_5" link2="link_6" reason="Adjacent"/>
<disable_collisions link1="base_link" link2="link_5" reason="Never"/>
<disable_collisions link1="link_1" link2="link_3" reason="Never"/>
<disable_collisions link1="base_link" link2="link_2" reason="Never"/>
<disable_collisions link1="link_2" link2="link_6" reason="Never"/>
<disable_collisions link1="link_4" link2="link_5" reason="Adjacent"/>
<disable_collisions link1="base_link" link2="link_6" reason="Never"/>
<disable_collisions link1="base_link" link2="link_3" reason="Never"/>
```

1.8.10 Tesseract Support Package

This package contains support data used for unit tests and examples throughout Tesseract.

1.8.11 Tesseract URDF Package

Background

This package contains urdf parser used by Tesseract. It supports additional shape and features not supported by urdfdom. This wiki only contains additional items and for more information please refer to http://wiki.ros.org/urdf/XML.

Features

1. New Shapes
    - Capsule
    - Cone
    - Mesh
    - Convex Mesh
    - SDF Mesh
    - Octomap
2. Origin
    - Quaternion
3. Limits
    - Acceleration - Oddly this was not supported by the original urdf specification
4. URDF Version
    - The original implementation of Tesseract interpreted mesh tags different than what is called version 2. It originally converted mesh geometry types to convex hull because there was no way to distinguish different types of meshes. Now in version 2 it supports the shape types (mesh, convex_mesh, sdf_mesh, etc.), therefore in version 2 the mesh tag is now interpreted as a detailed mesh and is no longer converted to a convex hull. To get the same behavior using version 2 change the tag to convex_mesh and set convert equal to true. For backwards compatibility any URDF without a version is assumed version 1 and mesh tags will be converted to convex hulls.

Change URDF Version

```
<robot name="kuka_iiwa" version="2">
</robot>
```

Defining New Shapes

Create Capsule

```
<capsule radius="1" length="2"/>
```

The total height is the **length + 2 * radius**, so the length is just the height between the center of each sphere of the capsule caps.

Create Cone

```
<cone radius="1" length="2"/>
```

The cone is like the cylinder. It is around z-axis and centered at the origin.

Create Convex Mesh

```
<convex_mesh filename="package://tesseract_support/meshes/box_2m.ply" scale="1 2 1"␣
↪convert="false"/>
```

This will create a convex hull shape type. This shape is more efficient than a regular mesh for collision checking. Also it provides an accurate penetration distance where in the case of mesh type you only get the penetration of one triangle into another.

Parameter	Required/Optional	Description
filename	Required	If convert is false (default) the mesh must be a convex hull represented by a polygon mesh. If it is triangulated such that multiple triangles represent the same surface you will get undefined behavior from collision checking.
scale	Optional	Scales the mesh axis aligned bounding box. Default scale = [1, 1, 1].
convert	Optional	If true the mesh is converted to a convex hull. Default convert = false.

Create SDF Mesh

```
<sdf_mesh filename="package://tesseract_support/meshes/box_2m.ply" scale="1 2 1" />
```

This will create a signed distance field shape type, which only affects collision shapes. This shape is more efficient than a regular mesh for collision checking, but not as efficient as convex hull.

Parameter	Required/Optional	Description
filename	Required	A path to a convex or non-convex mesh.
scale	Optional	Scales the mesh axis aligned bounding box. Default scale = [1, 1, 1].

Create Octree/Octomap

There are two methods for creating an octomap collision object. The first is to provide and octree file (.bt | .ot) and the second option is to provide a point cloud file (.pcd) with a resolution.

```
<octomap shape_type="box" prune="false" >
  <octree filename="package://tesseract_support/meshes/box_2m.bt"/>
</octomap>

<octomap shape_type="box" prune="false" >
  <point_cloud filename="package://tesseract_support/meshes/box_2m.pcd" resolution="0.1"/
↪>
</octomap>
```

This will create an octomap shape type. Each occupied cell is represented by either a box, sphere outside, or sphere inside shape.

Table 1: Octomap Element

Parameter	Required/Optional	Description
shape_type	Required	Currently three shape types (box, sphere_inside, sphere_outside).
prune	Optional	This executes the octree *toMaxLikelihood()* the *prune()* method prior to creating shape which will combine adjacent occupied cell into target cells resulting in fewer shapes.

Table 2: Octree Element

Parameter	Required/Optional	Description
filename	Required	A path to a binary or ascii octree file.

Table 3: Point Cloud Element

Parameter	Required/Optional	Description
filename	Required	A path to a PCL point cloud file.
resolution	Required	The resolution of the octree populated by the provided point cloud

Create Origin

```
<origin xyz="0 0 0" rpy="0 0 0" wxyz="1 0 0 0"/>;
```

This allows the ability to use a quaternion instead of roll, pitch and yaw values. It is acceptable to have both to allow backwards compatibility with other parsers, but the quaternion will take preference over rpy.

Parameter	Required/Optional	Description
wxyz	Optional	A Quaternion = [w, x, y, z]. It will be normalized on creation.

Acceleration Limits

```
<limit effort="30" velocity="1.0" acceleration="1.0" lower="-2.2" upper="0.7" />
```

Note: For backwards compatibility acceleration is required. If not provided it is assigned to be 0.5 * velocity.

1.8.12 Tesseract Visualization Package

This package contains visualization utilities and libraries.

1.9 Overview

1.9.1 Tesseract Motion Planning API

Overview

Tesseract contains an easy to use motion planning API that enables developers to quickly create custom-tailored motion plans. Tesseract's interface includes plugins for several popular planning algorithms such as OMPL, TrajOpt, TrajOpt IFOPT and Descartes.

In this tutorial, we will define each of the key components of Tesseract's motion planning API, then we will step through a simple freespace example.

Components

Environment Class

The *Environment* class is responsible for storing all information about the environment. This includes the robot and collision objects defined by the URDF and SRDF as well as any links/joints that are dynamically added to the environment.

The typical workflow for defining and initializing a *Environment* object is:

- Create a *Environment* object:

```
tesseract_environment::Environment::Ptr env = std::make_shared<tesseract_
↪environment::Environment>();
```

- Initialize the *Environment* object with the URDF and SRDF files:

```
tesseract_common::fs::path urdf_path("/path/to/urdf/my_robot.urdf");
tesseract_common::fs::path srdf_path("/path/to/srdf/my_robot.srdf");
env->init<tesseract_environment::OFKTStateSolver>(urdf_path, srdf_path, locator);
```

For more information about the *Environment* class see [TODO: add link to Tesseract Environment page].

Instruction Class

The *Instruction* class is a base class for all instruction type classes.

PlanInstruction Class

The *PlanInstruction* class inherits from the *Instruction* base class. This class allows the user to specify information about a singular movement in the trajectory. A *PlanInstruction* object holds information about the target pose and type of movement (*START*, *FREESPACE*, *LINEAR*, or *CIRCULAR*).

CompositeInstruction Class

The *CompositeInstruction* class inherits from the *Instruction* base class. This class allows the user to combine multiple *Instruction* objects into one *Instruction*. A *CompositeInstruction* object holds a list of instructions and enum indicating the type of ordering for the instructions (*ORDERED*, *UNORDERED*, *ORDERED_AND_REVERABLE*).

ProcessPlanningRequest Class

The *ProcessPlanningRequest* class allows the user to specify information about the process plan that they would like to solve.

The typical workflow for creating and initializing *ProcessPlanningRequest* objects is:

- Create a *ProcessPlanningRequest* object:

```
ProcessPlanningRequest request;
```

- Specify what type of planner to user:

```
request.name = process_planner_names::FREESPACE_PLANNER_NAME;
```

- Create *Instruction* and/or *CompositeInstruction* object(s):

```
CompositeInstruction program = freespaceExampleProgramIIWA();
```

- Add *Instruction* object to request:

```
request.instructions = Instruction(program);
```

ProcessPlanningServer Class

The *ProcessPlanningServer* class is responsible for accepting *ProcessPlanningRequest* objects and returning the *ProcessPlanningFuture* objects.

The typical workflow for creating a *ProcessPlanningServer* object and using it to solve a *ProcessPlanningRequest* is:

- Create a *ProcessPlanningServer* object:

```
ProcessPlanningServer planning_server(std::make_shared<ProcessEnvironmentCache>
↪(env), 1);
```

- Load default process planners:

```
planning_server.loadDefaultProcessPlanners();
```

- Create a *ProcessPlanningRequest* object (see section above).
- Run the planning server and pass in the *ProcessPlanningRequest* object:

```
ProcessPlanningFuture response = planning_server.run(request);
```

- Wait for the planning server to finish solving the process plan:

```
planning_server.waitForAll();
```

ProcessPlanningFuture Class

The *ProcessPlanningFuture* class is the type that gets returned when the *ProcessPlanningServer* solves a process plan. After calling *run()* the *ProcessPlanningServer* asynchronously solves the process plan and initializes the *ProcessPlanningFuture* object with the process plan results upon completion.

Running the Freespace Example

- Run the executable:

```
./devel/bin/tesseract_process_managers_freespace_manager_example
```

The Full Freespace Example

Stepping Through the Freespace Example

Initial Setup

Define resource locator function:

Create resource locator object:

Create environment object:

Initialize environment with URDF and SRDF files:

Dynamically load in ignition visualizer if exists:

Visualize the environment:

Defining the Process Plan

Create process planning server:

Create process planning request:

Define the program:

Solving the Process Plan

Solve the process plan:

Visualizing Results

Plot the process trajectory:

1.9.2 Tesseract Command Language

This is a high level programming language used throughout the Tesseract Planning Framework. The goal is to provide an abstraction between motion commands and motion planner specific configurations similar to programming on an industrial teach pendant.

At the lowest level the command language is a set of instruction where on an industrial teach pendant these include move instructions, set I/O instruction and set Analog instruction. These are all included in the command language with a few additional instruction like plan instruction and environment instruction to start. These instructions are specific to motion planning and environment management.

This high level language allows a novice user to create complex programs without the knowledge of planner specifics which is designed to encode the the whole process including motion planning, motion execution, environment management, sensor signals and much more.

Overview

The command language begins by defining your waypoints. The current supported waypoints are CartesianWaypoint, JointWaypoint and StateWaypoint.

- CartesianWaypoint: This is a Cartesian pose leveraged within the Plan Instruction
- JointWaypoint: This is a joint space pose leveraged within the Plan Instruction
- StateWaypoint: This not only includes joint names and positions but also includes velocity, acceleration, and time from start. It is primarily used in the Move Instruction but may be used in the Plan Instruction.

```
Waypoint wp0 = JointWaypoint(fwd_kin->getJointNames(), Eigen::VectorXd::Zero(6));
Waypoint wp1 = CartesianWaypoint(Eigen::Isometry3d::Identity() * Eigen::Translation3d(0.
↪8, -0.3, 0.8));
Waypoint wp2 = CartesianWaypoint(Eigen::Isometry3d::Identity() * Eigen::Translation3d(0.
↪8, -0.2, 0.8));
Waypoint wp3 = CartesianWaypoint(Eigen::Isometry3d::Identity() * Eigen::Translation3d(0.
↪8, -0.1, 0.8));
Waypoint wp4 = CartesianWaypoint(Eigen::Isometry3d::Identity() * Eigen::Translation3d(0.
↪8, 0.0, 0.8));
Waypoint wp5 = CartesianWaypoint(Eigen::Isometry3d::Identity() * Eigen::Translation3d(0.
↪8, 0.1, 0.8));
Waypoint wp6 = CartesianWaypoint(Eigen::Isometry3d::Identity() * Eigen::Translation3d(0.
↪8, 0.2, 0.8));
Waypoint wp7 = CartesianWaypoint(Eigen::Isometry3d::Identity() * Eigen::Translation3d(0.
↪8, 0.3, 0.8));
```

After the definitions of the waypoints, the plan instructions need to be defined. As shown below the plan instruction requires three inputs, the first is the waypoint, the second is the type of motion LINEAR, FREESPACE or START and

last is the profile name. The profile name corresponds to a set of planner specific parameters on the motion planning side.

```
PlanInstruction plan_f0(wp1, PlanInstructionType::FREESPACE, "FREESPACE");
PlanInstruction plan_c0(wp2, PlanInstructionType::LINEAR, "RASTER");
PlanInstruction plan_c1(wp3, PlanInstructionType::LINEAR, "RASTER");
PlanInstruction plan_c2(wp4, PlanInstructionType::LINEAR, "RASTER");
PlanInstruction plan_c3(wp5, PlanInstructionType::LINEAR, "RASTER");
PlanInstruction plan_c4(wp6, PlanInstructionType::LINEAR, "RASTER");
PlanInstruction plan_c5(wp7, PlanInstructionType::LINEAR, "RASTER");
```

The last component is to create the program which would be sent to the motion planning server. This involves the use of a Composite Instruction which is a vector of instructions with additional properties to be set. One of those additional parameters is the ManipulatorInfo which includes information about the manipulator along with working frame and tool center point information.

```
CompositeInstruction program("raster_program", CompositeInstructionOrder::ORDERED,␣
↪ManipulatorInfo("manipulator"));

PlanInstruction start_instruction(wp0, PlanInstructionType::START);
program.setStartInstruction(start_instruction);

program.push_back(plan_f0);
program.push_back(plan_c0);
program.push_back(plan_c1);
program.push_back(plan_c2);
program.push_back(plan_c3);
program.push_back(plan_c4);
program.push_back(plan_c5);
```

Now that the program has been created additional utility function will be discussed.

One useful feature of the Command Language, is it can be serialized to xml and deserialized from xml.

```
std::string xml_string = toXMLString(programn);
Instruction from_xml_string = fromXMLString(xml_string);
```

The Tesseract Command Language is unique within Tesseract because it leverages Type Erasures instead of inheritance. This was chosen because Type Erasures obey copy semantics allowing for full use of the stl libraries.

In addition, the Instruction and Waypoint Type Erasure provide a cast and cast_const for casting the type to the erased type leveraging the getType() method.

```
if (isComposite(from_xml_string))
{
 const auto* const_composite = from_xml_string.cast_const<CompositeInstruction>();
 auto* composite = from_xml_string.const<CompositeInstruction>();
}
```

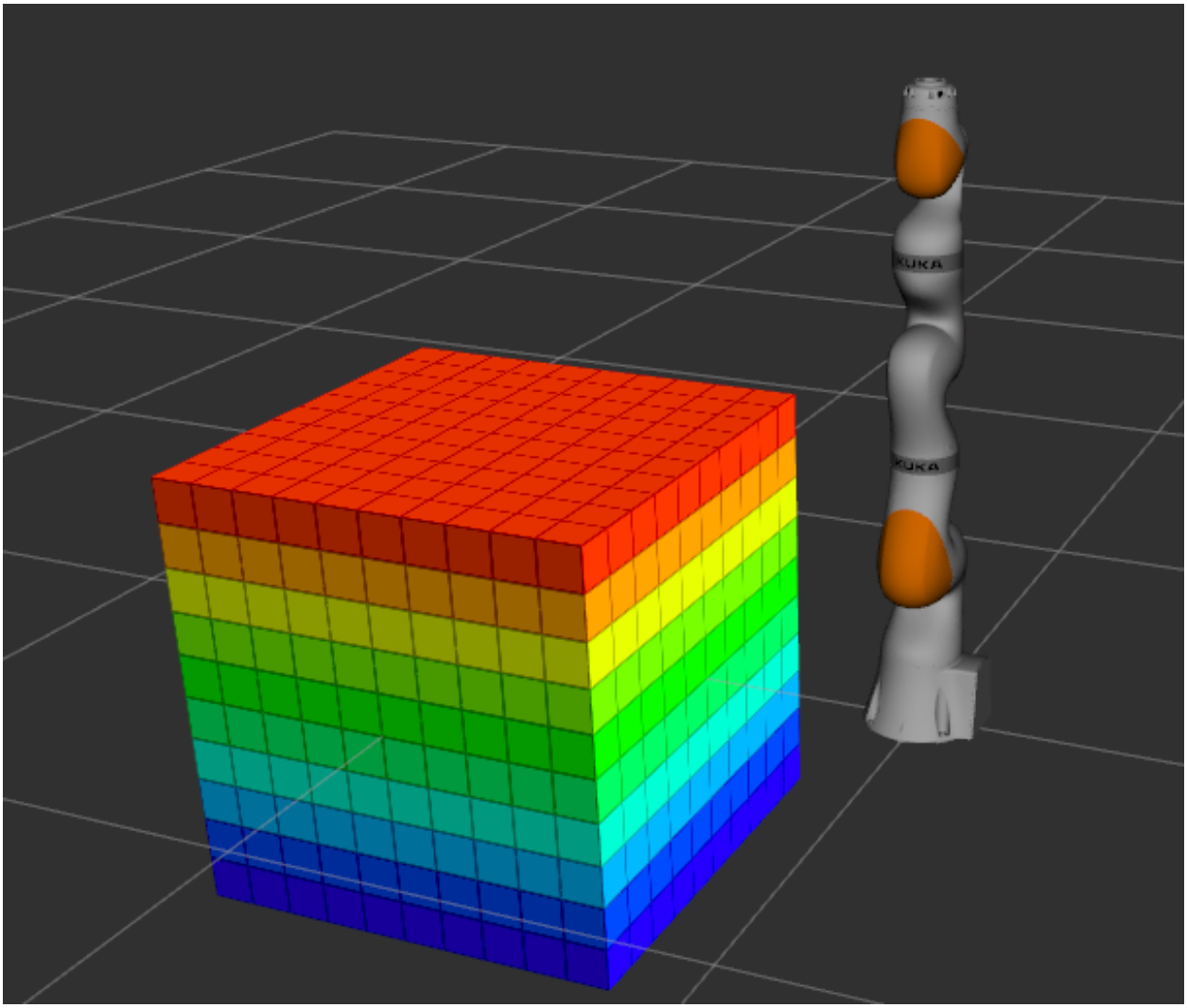
1.10 Quickstart

1.11 Examples

1.12 Tesseract Planning Packages

1.13 Overview

1.13.1 Tesseract ROSCPP Interface



Note: This tutorial was heavily inspired from MoveIt! tutorials. Credit should be recognized for the tutorial structure and some text content.

For Tesseract, the most common interface will be through ROS with the tesseract_ros package. It provides functionality for common operations such as setting joint or pose goals, creating motion plans, moving the robot, adding objects into

the environment, etc.

Getting Started

If you haven't already done so, make sure you've completed the steps in Getting Started.

The Launch File

The entire launch file is here on GitHub

Running the Code

Open a terminal, and launch the executable and Rviz. Wait for Rviz to finish loading:

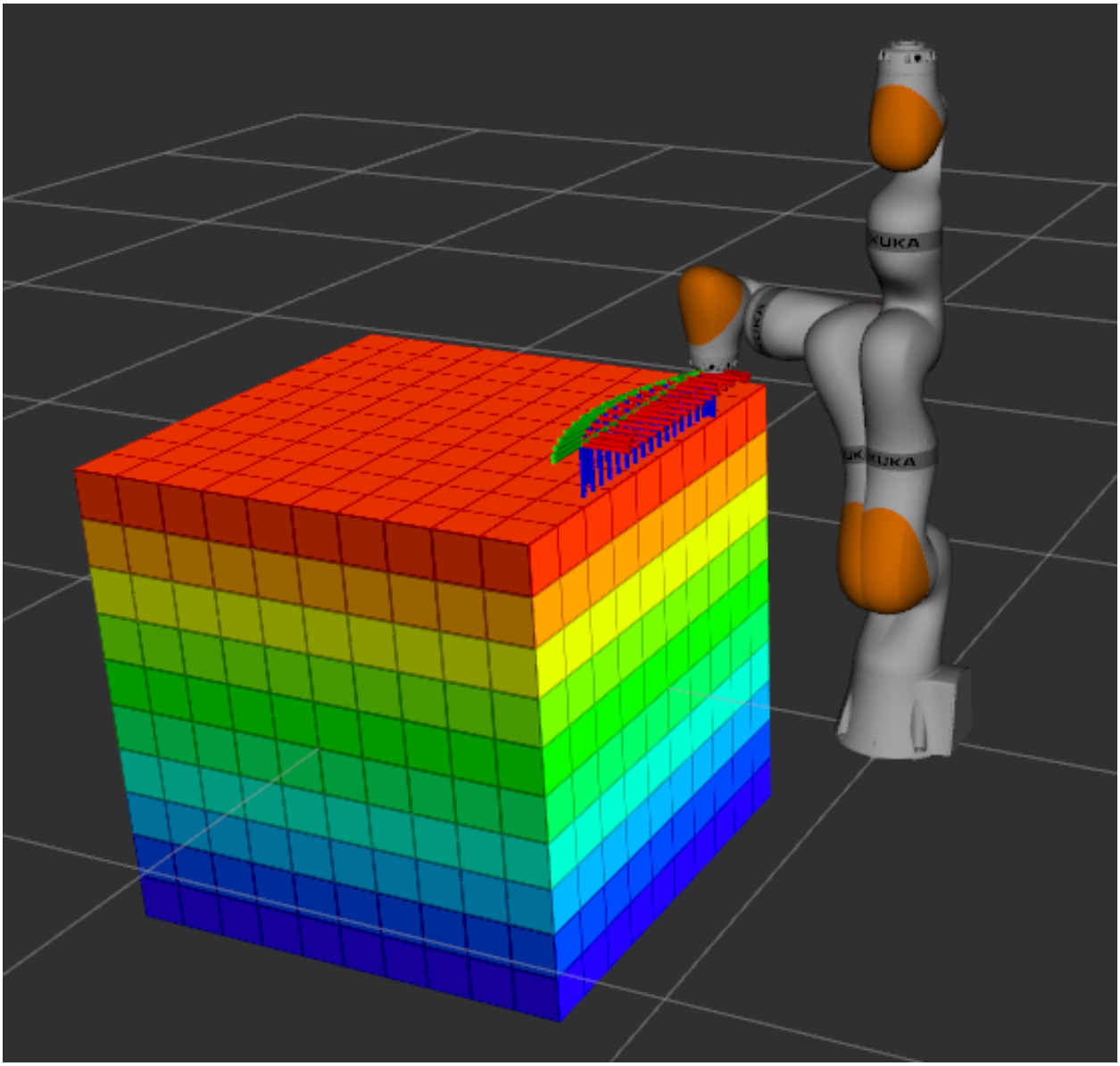
```
roslaunch tesseract_ros_examples basic_cartesian_example.launch
```

After Rviz opens, go back to the terminal where the launch file was executed, and the following text should be visible. Robot motion in Rviz begins immediately upon pressing any key.

```
[ERROR] [1613730481.601846614]: Hit enter key to continue!
```

Expected Output

In Rviz, the the robot should trace a trajectory in its environment on top of the cube object:



The Entire Code

The entire example code is written from one executable node that links against a *BasicCartesianExample(…)* class (installed as a library).

Executable Node:

- basic_cartesian_example_node.cpp

BasicCartesianExample(…) Class:

- basic_cartesian_example.h
- basic_cartesian_example.cpp

Next we step through the code piece by piece. The basic_cartesian_example_node.cpp initializes ROS, instantiates the *BasicCartesianExample(…)* class, and calls the *run()* method from the class.

```cpp
/**
 * @file basic_cartesian_example_node.cpp
 * @brief Basic cartesian example node
 *
 * @author Levi Armstrong
 * @date July 22, 2019
 * @version TODO
 * @bug No known bugs
 *
 * @copyright Copyright (c) 2017, Southwest Research Institute
 *
 * @par License
 * Software License Agreement (Apache License)
 * @par
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * @par
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include <tesseract_common/macros.h>
TESSERACT_COMMON_IGNORE_WARNINGS_PUSH
#include <ros/ros.h>
TESSERACT_COMMON_IGNORE_WARNINGS_POP

#include <tesseract_examples/basic_cartesian_example.h>
#include <tesseract_monitoring/environment_monitor.h>
#include <tesseract_rosutils/plotting.h>

using namespace tesseract_examples;
using namespace tesseract_rosutils;

/** @brief Default ROS parameter for robot description */
const std::string ROBOT_DESCRIPTION_PARAM = "robot_description";

/** @brief Default ROS parameter for robot description */
const std::string ROBOT_SEMANTIC_PARAM = "robot_description_semantic";

/** @brief RViz Example Namespace */
const std::string EXAMPLE_MONITOR_NAMESPACE = "tesseract_ros_examples";

int main(int argc, char** argv)
{
  ros::init(argc, argv, "basic_cartesian_example_node");
  ros::NodeHandle pnh("~");
  ros::NodeHandle nh;
```

(continues on next page)

(continued from previous page)

```
  bool plotting = true;
  bool rviz = true;
  bool ifopt = false;
  bool debug = false;

  // Get ROS Parameters
  pnh.param("plotting", plotting, plotting);
  pnh.param("rviz", rviz, rviz);
  pnh.param("ifopt", ifopt, ifopt);
  pnh.param("debug", debug, debug);

  // Initial setup
  std::string urdf_xml_string, srdf_xml_string;
  nh.getParam(ROBOT_DESCRIPTION_PARAM, urdf_xml_string);
  nh.getParam(ROBOT_SEMANTIC_PARAM, srdf_xml_string);

  auto env = std::make_shared<tesseract_environment::Environment>();
  auto locator = std::make_shared<tesseract_rosutils::ROSResourceLocator>();
  if (!env->init(urdf_xml_string, srdf_xml_string, locator))
    exit(1);

  // Create monitor
  auto monitor = std::make_shared<tesseract_monitoring::ROSEnvironmentMonitor>(env,␣
↪EXAMPLE_MONITOR_NAMESPACE);
  if (rviz)
    monitor->startPublishingEnvironment();

  ROSPlottingPtr plotter;
  if (plotting)
    plotter = std::make_shared<ROSPlotting>(env->getSceneGraph()->getRoot());

  BasicCartesianExample example(env, plotter, ifopt, debug);
  example.run();
}
```

For the *BasicCartesianExample(…)* class within basic_cartesian_example.cpp, the translation unit is setup with using-declarations and required strings.

```
using namespace tesseract_environment;
using namespace tesseract_scene_graph;
using namespace tesseract_collision;
using namespace tesseract_rosutils;
using namespace tesseract_visualization;

/** @brief Default ROS parameter for robot description */
const std::string ROBOT_DESCRIPTION_PARAM = "robot_description";

/** @brief Default ROS parameter for robot description */
const std::string ROBOT_SEMANTIC_PARAM = "robot_description_semantic";

/** @brief RViz Example Namespace */
const std::string EXAMPLE_MONITOR_NAMESPACE = "tesseract_ros_examples";
```

The primary functionality is in the *run()* method of the class.

- Get parameters from the ROS Parameter Server and pass them into the Tesseract environment (*env_*).
- Create the environment monitor that understands contact points within the environment.
- Create an octomap of a cube that is given to the environment monitor.

```
bool BasicCartesianExample::run()
{
  using tesseract_planning::CartesianWaypoint;
  using tesseract_planning::CompositeInstruction;
  using tesseract_planning::CompositeInstructionOrder;
  using tesseract_planning::Instruction;
  using tesseract_planning::ManipulatorInfo;
  using tesseract_planning::PlanInstruction;
  using tesseract_planning::PlanInstructionType;
  using tesseract_planning::ProcessPlanningFuture;
  using tesseract_planning::ProcessPlanningRequest;
  using tesseract_planning::ProcessPlanningServer;
  using tesseract_planning::StateWaypoint;
  using tesseract_planning::Waypoint;
  using tesseract_planning_server::ROSProcessEnvironmentCache;

  // Initial setup
  std::string urdf_xml_string, srdf_xml_string;
  nh_.getParam(ROBOT_DESCRIPTION_PARAM, urdf_xml_string);
  nh_.getParam(ROBOT_SEMANTIC_PARAM, srdf_xml_string);

  ResourceLocator::Ptr locator = std::make_shared<tesseract_rosutils::ROSResourceLocator>
↪();
  if (!env_->init<OFKTStateSolver>(urdf_xml_string, srdf_xml_string, locator))
    return false;

  // Create monitor
  monitor_ = std::make_shared<tesseract_monitoring::EnvironmentMonitor>(env_, EXAMPLE_
↪MONITOR_NAMESPACE);
  if (rviz_)
    monitor_->startPublishingEnvironment(tesseract_
↪monitoring::EnvironmentMonitor::UPDATE_ENVIRONMENT);

  // Create octomap and add it to the local environment
  Command::Ptr cmd = addPointCloud();
  if (!monitor_->applyCommand(cmd))
    return false;
```

In the *addPointCloud()* method, we use the Point Cloud Library to create a cube. Then, the cube is transformed into an Octomap representation, and this is passed to the Tesseract environment.

```
Command::Ptr BasicCartesianExample::addPointCloud()
{
  // Create octomap and add it to the local environment
  pcl::PointCloud<pcl::PointXYZ> full_cloud;
  double delta = 0.05;
  auto length = static_cast<int>(1 / delta);
```

(continues on next page)

(continued from previous page)

```
  for (int x = 0; x < length; ++x)
    for (int y = 0; y < length; ++y)
      for (int z = 0; z < length; ++z)
        full_cloud.push_back(pcl::PointXYZ(-0.5f + static_cast<float>(x * delta),
                                           -0.5f + static_cast<float>(y * delta),
                                           -0.5f + static_cast<float>(z * delta)));

  sensor_msgs::PointCloud2 pointcloud_msg;
  pcl::toROSMsg(full_cloud, pointcloud_msg);

  octomap::Pointcloud octomap_data;
  octomap::pointCloud2ToOctomap(pointcloud_msg, octomap_data);
  std::shared_ptr<octomap::OcTree> octree = std::make_shared<octomap::OcTree>(2 * delta);
  octree->insertPointCloud(octomap_data, octomap::point3d(0, 0, 0));

  // Add octomap to environment
  Link link_octomap("octomap_attached");

  Visual::Ptr visual = std::make_shared<Visual>();
  visual->origin = Eigen::Isometry3d::Identity();
  visual->origin.translation() = Eigen::Vector3d(1, 0, 0);
  visual->geometry = std::make_shared<tesseract_geometry::Octree>(octree, tesseract_
↪geometry::Octree::BOX);
  link_octomap.visual.push_back(visual);

  Collision::Ptr collision = std::make_shared<Collision>();
  collision->origin = visual->origin;
  collision->geometry = visual->geometry;
  link_octomap.collision.push_back(collision);

  Joint joint_octomap("joint_octomap_attached");
  joint_octomap.parent_link_name = "base_link";
  joint_octomap.child_link_name = link_octomap.getName();
  joint_octomap.type = JointType::FIXED;

  return std::make_shared<tesseract_environment::AddLinkCommand>(link_octomap, joint_
↪octomap);
}
```

Continuing in the *run()* method, we define motion parameters by:

- Setting cartesian waypoints and asking the planner to perform freespace and linear moves for different programs.
- Create a planning server that will solve each program for the robot's joint kinematics.
- Plot the trajectory in Rviz for animation.

```
  // Create Program
  CompositeInstruction program("cartesian_program", CompositeInstructionOrder::ORDERED,␣
↪ManipulatorInfo("manipulator"));

  // Start Joint Position for the program
  Waypoint wp0 = StateWaypoint(joint_names, joint_pos);
```

(continues on next page)

(continued from previous page)

```
  PlanInstruction start_instruction(wp0, PlanInstructionType::START);
  program.setStartInstruction(start_instruction);

  // Create cartesian waypoint
  Waypoint wp1 = CartesianWaypoint(Eigen::Isometry3d::Identity() *␣
↪Eigen::Translation3d(0.5, -0.2, 0.62) *
                                   Eigen::Quaterniond(0, 0, 1.0, 0));

  Waypoint wp2 = CartesianWaypoint(Eigen::Isometry3d::Identity() *␣
↪Eigen::Translation3d(0.5, 0.3, 0.62) *
                                   Eigen::Quaterniond(0, 0, 1.0, 0));

  // Plan freespace from start
  PlanInstruction plan_f0(wp1, PlanInstructionType::FREESPACE, "freespace_profile");
  plan_f0.setDescription("from_start_plan");

  // Plan linear move
  PlanInstruction plan_c0(wp2, PlanInstructionType::LINEAR, "RASTER");

  // Plan freespace to end
  PlanInstruction plan_f1(wp0, PlanInstructionType::FREESPACE, "freespace_profile");
  plan_f1.setDescription("to_end_plan");

  // Add Instructions to program
  program.push_back(plan_f0);
  program.push_back(plan_c0);
  program.push_back(plan_f1);

  ROS_INFO("basic cartesian plan example");

  // Create Process Planning Server
  ProcessPlanningServer planning_server(std::make_shared<ROSProcessEnvironmentCache>
↪(monitor_), 5);
  planning_server.loadDefaultProcessPlanners();

  // Create Process Planning Request
  ProcessPlanningRequest request;
  request.name = tesseract_planning::process_planner_names::TRAJOPT_PLANNER_NAME;
  request.instructions = Instruction(program);

  // Print Diagnostics
  request.instructions.print("Program: ");

  // Solve process plan
  ProcessPlanningFuture response = planning_server.run(request);
  planning_server.waitForAll();

  // Plot Process Trajectory
  if (rviz_ && plotter != nullptr && plotter->isConnected())
  {
    plotter->waitForInput();
    const auto* ci = response.results->cast_const<tesseract_
```

(continues on next page)

(continued from previous page)

```
↪planning::CompositeInstruction>();
    tesseract_common::Toolpath toolpath = tesseract_planning::toToolpath(*ci, env_);
    tesseract_common::JointTrajectory trajectory = tesseract_
↪planning::toJointTrajectory(*ci);
    plotter->plotMarker(ToolpathMarker(toolpath));
    plotter->plotTrajectory(trajectory, env_->getStateSolver());
  }

  ROS_INFO("Final trajectory is collision free");
  return true;
}

}  // namespace tesseract_ros_examples
```

1.14 Quickstart

1.15 Examples

1.16 Tesseract ROS Packages

Warning: These packages are under heavy development and are subject to change.

1.16.1 Tesseract Examples Package

This package contains examples using tesseract and tesseract_ros for motion planning and collision checking.

1.16.2 Tesseract Monitoring Package

This package contains different types of environment monitors. It currently contains a contact monitor and environment monitor. The contact monitor will monitor the active environment state and publish contact information. This is useful if the robot is being controlled outside of ROS, but you want to make sure it does not collide with objects in the environment. The second is the environment monitor, which is the main environment which facilitates requests to add, remove, disable and enable collision objects, while publishing its current state to keep other ROS nodes updated with the latest environment.

1.16.3 Tesseract Msgs Package

This package contains the ROS message types used by Tesseract ROS.

1.16.4 Tesseract Rosutils Package

This package contains the utilities like converting from ROS message types to native Tesseract types and the reverse.

1.16.5 Tesseract Rviz Package

This package contains the ROS visualization plugins for Rviz to visualize Tesseract. All of the features have been composed in libraries to enable to the ability to create custom displays quickly.

1.17 Overview

1.18 Quickstart

1.19 Examples

1.20 ROS2 Packages