Industrial Training

Nov 06, 2019

Contents

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 1

Tesseract Core Packages

- **tesseract** – This is the main class that manages the major component Environment, Forward Kinematics, Inverse Kinematics and loading from various data.
- **tesseract_collision** – This package contains privides a common interface for collision checking prividing several implementation of a Bullet collision library and FCL collision library. It includes both continuous and discrete collision checking for convex-convex, convex-concave and concave-concave shapes.
- **tesseract_common** – This package contains common functionality needed by the majority of the packages.
- **tesseract_environment** – 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.
- **tesseract_geometry** – This package contains geometry types used by Tesseract including primitive shapes, mesh, convex hull mesh, octomap and signed distance field.
- **tesseract_kinematics** – This package contains a common interface for Forward and Inverse kinematics for Chain, Tree's and Graphs including implementation using KDL and OPW Kinematics.
- **tesseract_planners** – This package contains a common interface for Planners and includes implementation for OMPL, TrajOpt and Descartes.
- **tesseract_scene_graph** – This package contains the scene graph which is the data structure used to manage the connectivity of objects in the environment. It inherits from boost graph and provides addition functionality for adding,removing and modifying Links and Joints along with search implementation.
- **tesseract_support** – This package contains support data used for unit tests and examples throughout Tesseract.
- **tesseract_visualization** – This package contains visualization utilities and libraries.
- **tesseract_urdf** - This package contains a custom urdf parser supporting addition shapes and features currently not supported by urdfdom.

CHAPTER 2

Tesseract ROS Packages

- **tesseract_examples** – This package contains examples using tesseract and tesseract_ros for motion planning and collision checking.
- **tesseract_plugins** – This contains plugins for collision and kinematics which are automatically loaded by the monitors.
- **tesseract_rosutils** – This package contains the utilities like converting from ROS message types to native Tesseract types and the reverse.
- **tesseract_msgs** – This package contains the ROS message types used by Tesseract ROS.
- **tesseract_rviz** – 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.
- **tesseract_monitoring** – 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.

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

CHAPTER 3

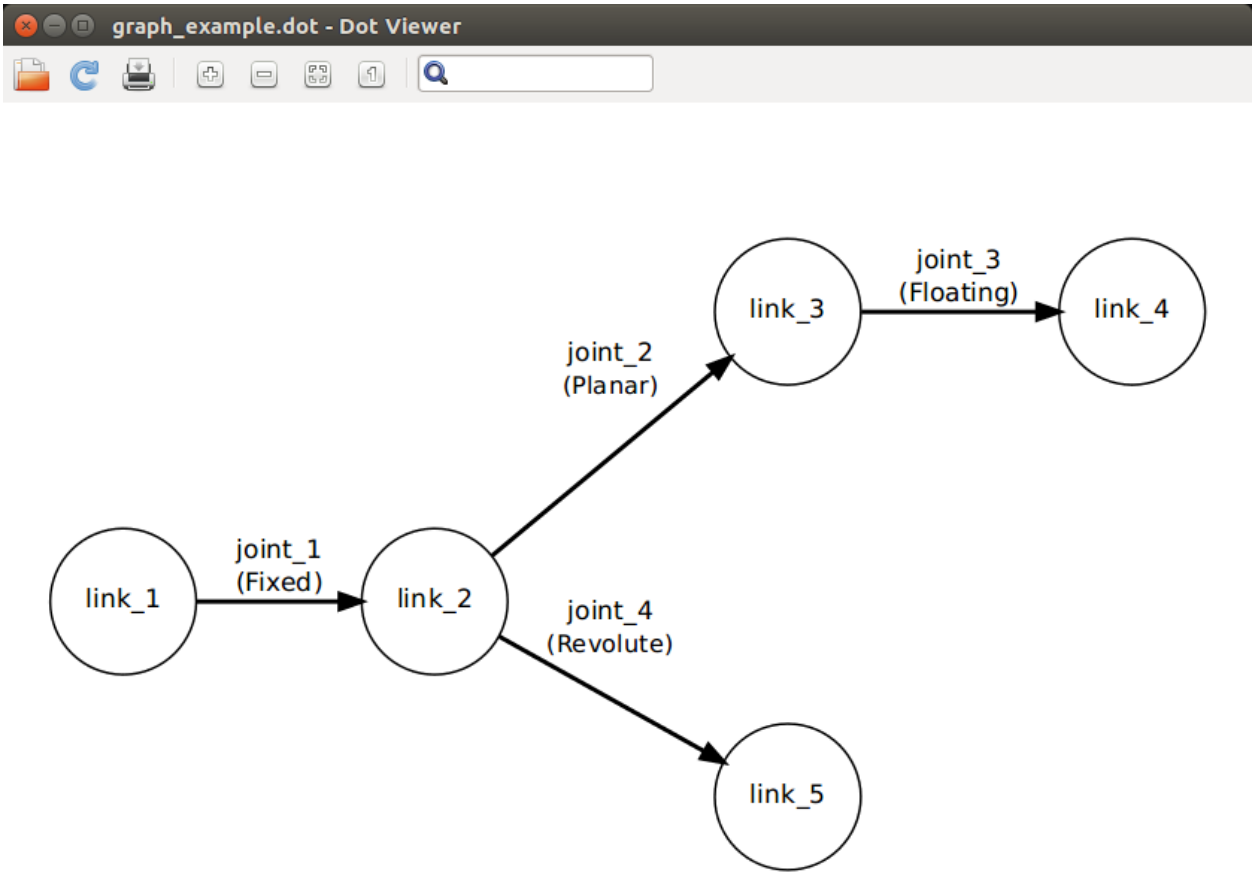
Packages

3.1 Tesseract Scene Graph Package

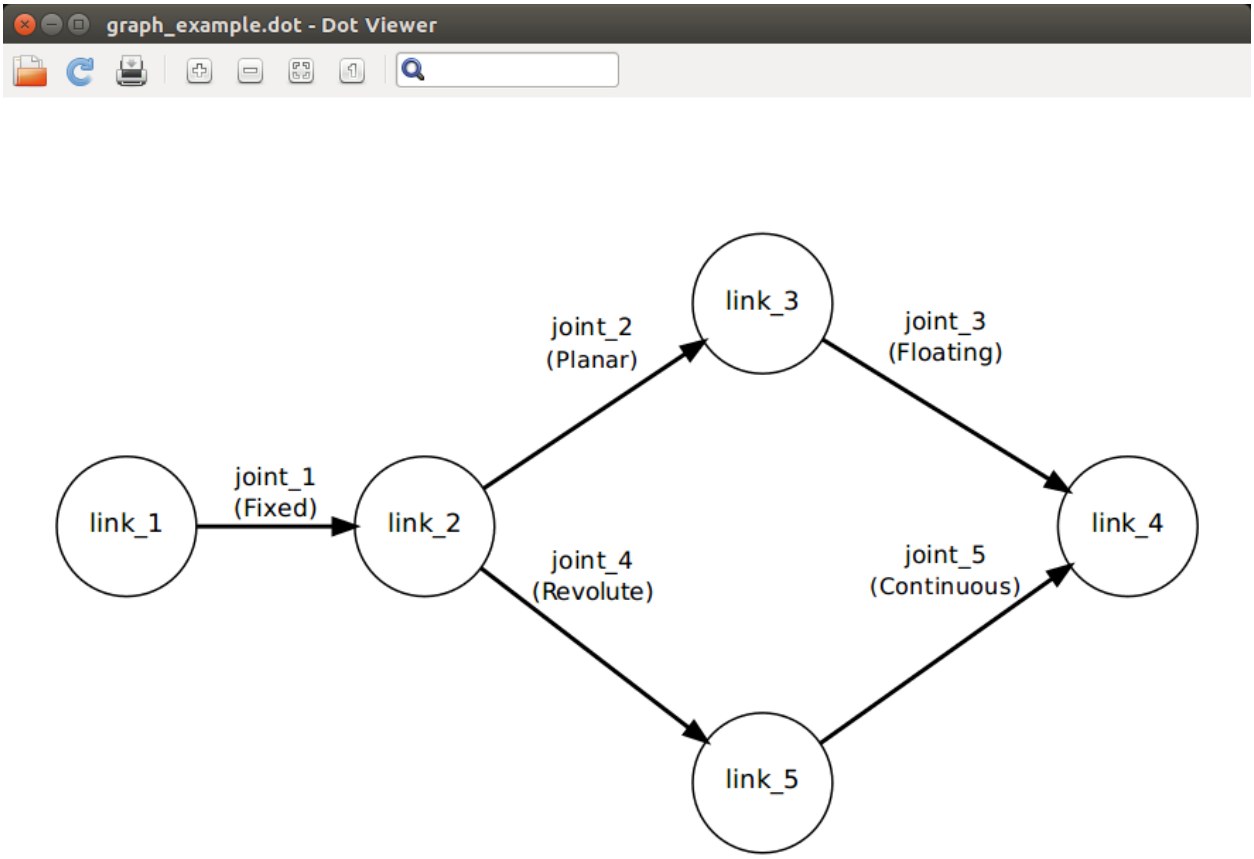
3.1.1 Background

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)



3.1.2 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

3.1.3 Examples

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

3.1.4 Building A Scene Graph

```
#include <console_bridge/console.h>
#include <tesseract_scene_graph/graph.h>
#include <iostream>

using namespace tesseract_scene_graph;

std::string toString(const SceneGraph::Path& 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);

  // Create scene graph
  SceneGraph g;

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

(continues on next page)

(continued from previous page)

```
// 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;

// Add joints
g.addJoint(joint_1);
g.addJoint(joint_2);
g.addJoint(joint_3);
g.addJoint(joint_4);

// 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());

// Check getInvAdjacentLinkNames Method
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());

// 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());

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

// Save Graph
g.saveDOT("/tmp/graph_acyclic_tree_example.dot");

// Test if the graph is Acyclic
bool is_acyclic = g.isAcyclic();
CONSOLE_BRIDGE_logInform(toString(is_acyclic).c_str());
```

(continues on next page)

(continued from previous page)

```
  // Test if the graph is Tree
  bool is_tree = g.isTree();
  CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());

  // 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());

  // Remove unused link
  g.removeLink(link_6.getName());
  is_tree = g.isTree();
  CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());

  // 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);

  // Save new graph
  g.saveDOT("/tmp/graph_acyclic_not_tree_example.dot");

  // Test again if the graph is Acyclic
  is_acyclic = g.isAcyclic();
  std::cout << toString(is_acyclic).c_str();

  // Test again if the graph is Tree
  is_tree = g.isTree();
  CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());

  // Get Shortest Path
  SceneGraph::Path path = g.getShortestPath("link_1", "link_4");
  CONSOLE_BRIDGE_logInform(toString(path).c_str());
}
```

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");
```

(continues on next page)

(continued from previous page)

```
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("/tmp/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.

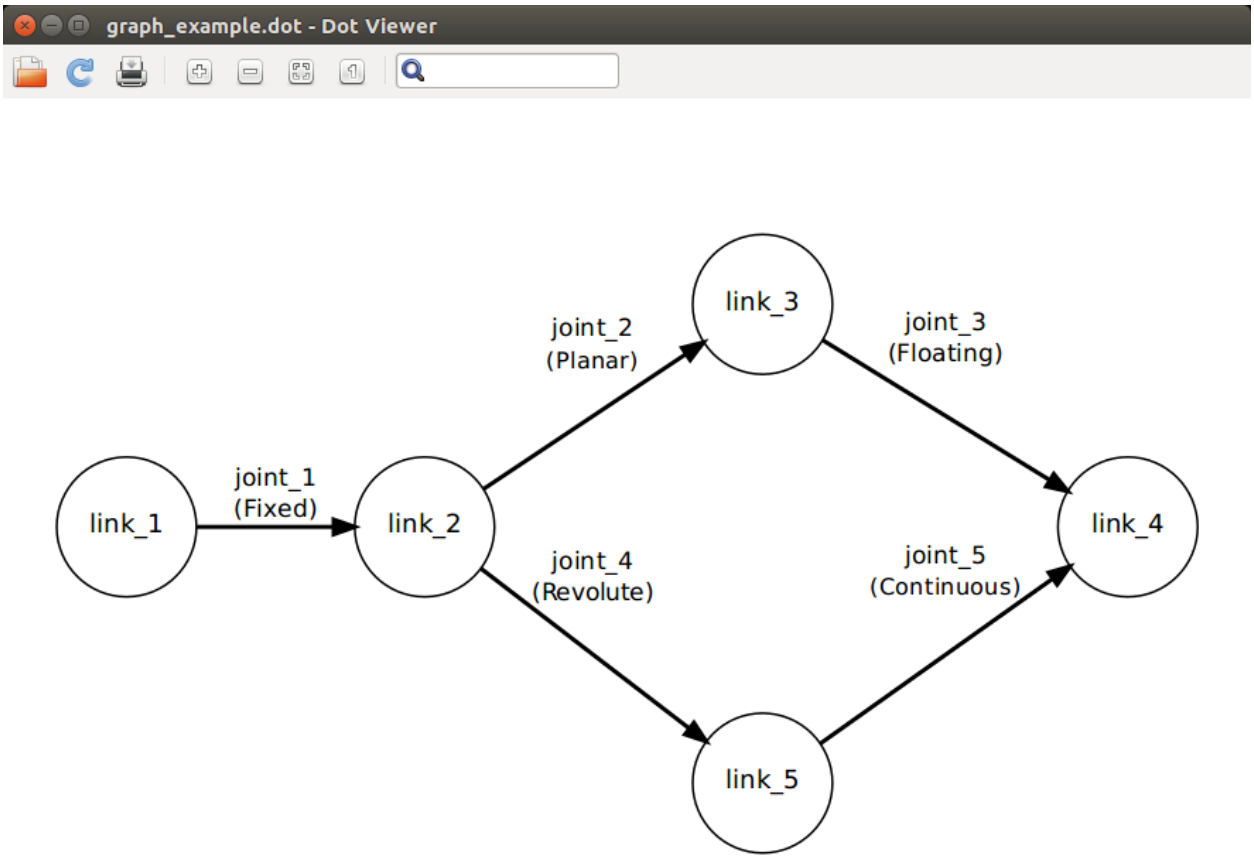
```
Link link_6("link_6");
g.addLink(link_6);
is_tree = g.isTree();
CONSOLE_BRIDGE_logInform(toString(is_tree).c_str());
```

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

```
g.removeLink(link_6.getName());
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("/tmp/graph_acyclic_not_tree_example.dot");
```

Test to confirm it is acyclic, should return true.

```
is_acyclic = g.isAcyclic();
std::cout << 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

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

3.1.5 Create Scene Graph from URDF

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

Create scene graph from urdf

Print information about the scene graph to the terminal

Save the graph to a file.

3.1.6 Parse SRDF adding Allowed Collision Matrix to Graph

```
#include <console_bridge/console.h>
#include <tesseract_scene_graph/graph.h>
#include <tesseract_scene_graph/parser/srdf_parser.h>
#include <tesseract_scene_graph/utils.h>
#include <iostream>

using namespace tesseract_scene_graph;

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

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

// Define a resource locator function
std::string locateResource(const std::string& url)
{
  std::string mod_url = url;
  if (url.find("package://tesseract_support") == 0)
  {
    mod_url.erase(0, strlen("package://tesseract_support"));
    size_t pos = mod_url.find("/");
    if (pos == std::string::npos)
    {
```

(continues on next page)

(continued from previous page)

```
      return std::string();
    }

    std::string package = mod_url.substr(0, pos);
    mod_url.erase(0, pos);
    std::string package_path = std::string(TESSERACT_SUPPORT_DIR);

    if (package_path.empty())
    {
      return std::string();
    }

    mod_url = package_path + mod_url;
  }

  return mod_url;
}

int main(int /*argc*/, char** /*argv*/)
{
  std::string srdf_file = std::string(TESSERACT_SUPPORT_DIR) + "/urdf/lbr_iiwa_14_
↪r820.srdf";

  // Create scene graph
  ResourceLocatorFn locator = locateResource;
  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";
```

(continues on next page)

(continued from previous page)

```
  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);

  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);

  // Parse the srdf
  SRDFModel srdf;
  bool success = srdf.initFile(g, srdf_file);
  CONSOLE_BRIDGE_logInform("SRDF loaded: %s", toString(success).c_str());

  processSRDFAllowedCollisions(g, srdf);

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

Example Explanation

Create Resource Locator

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

```
std::string locateResource(const std::string& url)
{
  std::string mod_url = url;
  if (url.find("package://tesseract_support") == 0)
  {
    mod_url.erase(0, strlen("package://tesseract_support"));
    size_t pos = mod_url.find("/");
    if (pos == std::string::npos)
    {
      return std::string();
    }

    std::string package = mod_url.substr(0, pos);
    mod_url.erase(0, pos);
    std::string package_path = std::string(TESSERACT_SUPPORT_DIR);

    if (package_path.empty())
    {
      return std::string();
    }

    mod_url = package_path + mod_url;
  }

  return mod_url;
}
```

Load URDF and SRDF

Get the file path to the URDF and SRDF file

Create Scene Graph from URDF

```
  ResourceLocatorFn locator = locateResource;
  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);
```

(continues on next page)

(continued from previous page)

```
  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);

  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";
```

(continues on next page)

(continued from previous page)

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

Parse SRDF

Add Allowed Collision Matrix to Scene Graph

Methods for getting Allowed Collision Matrix from Scene Graph

3.1.7 Parse Mesh from file

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.

Print Mesh Information to Terminal

3.2 Tesseract Collision Package

3.2.1 Background

This package is a 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 objects and perform collision checks.

3.2.2 Features

1. Add/Remove collision objects consisting of multiple collision shapes.
2. Enable/Disable collision objects
3. Set collision objects transformation
4. Set contact distance threshold. If two objects are further than this distance they are ignored.
5. Perform Contact Test with various exit conditions
    - Exit on first **tesseract::ContactTestType::FIRST**
    - Store only closets for each collision object **tesseract::ContactTestType::CLOSEST**
    - Store all contacts for each collision object **tesseract::ContactTestType::ALL**

3.2.3 Discrete Collision Checker Example

```
#include <console_bridge/console.h>
#include "tesseract_collision/bullet/bullet_discrete_bvh_manager.h"

using namespace tesseract_collision;
using namespace tesseract_geometry;

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*/)
{
  // Create Collision Manager
  tesseract_collision_bullet::BulletDiscreteBVHManager checker;

  // Add box to checker
  CollisionShapePtr box(new 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);

  checker.addCollisionObject("box_link", 0, obj1_shapes, obj1_poses);

  // Add thin box to checker which is disabled
  CollisionShapePtr thin_box(new 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);

  checker.addCollisionObject("thin_box_link", 0, obj2_shapes, obj2_poses, false);

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

  tesseract_common::VectorVector3d mesh_vertices;
  Eigen::VectorXi mesh_faces;
  loadSimplePlyFile(std::string(DATA_DIR) + "/box_2m.ply", mesh_vertices, mesh_faces);

  // This is required because convex hull cannot have multiple faces on the same␣
↪plane.
  std::shared_ptr<tesseract_common::VectorVector3d> ch_verticies(new tesseract_
↪common::VectorVector3d());
```

(continues on next page)

(continued from previous page)

```
  std::shared_ptr<Eigen::VectorXi> ch_faces(new Eigen::VectorXi());
  int ch_num_faces = createConvexHull(*ch_verticies, *ch_faces, mesh_vertices);
  second_box.reset(new ConvexMesh(ch_verticies, ch_faces, ch_num_faces));

  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);

  CONSOLE_BRIDGE_logInform("Test when object is inside another");
  checker.setActiveCollisionObjects({ "box_link", "second_box_link" });
  checker.setContactDistanceThreshold(0.1);

  // 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
  ContactResultMap result;
  checker.contactTest(result, ContactTestType::CLOSEST);

  ContactResultVector result_vector;
  flattenResults(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());

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

  checker.setCollisionObjectsTransform(location);

  // Check for collision after moving object
```

(continues on next page)

(continued from previous page)

```
  checker.contactTest(result, ContactTestType::CLOSEST);
  flattenResults(std::move(result), result_vector);

  CONSOLE_BRIDGE_logInform("Has collision: %s", toString(result_vector.empty()).c_
↪str());

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

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

  // Check for contact with new threshold
  checker.contactTest(result, ContactTestType::CLOSEST);
  flattenResults(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());
}
```

Example Explanation

Create Contact Checker

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

There are several available contact checkers.

- Recommended
  - BulletDiscreteBVHManager
  - BulletCastBVHManager
- Alternative
  - BulletDiscreteSimpleManager
  - BulletCastSimpleManager
- Beta
  - FCLDiscreteBVHManager

Add Collision Objects to Contact Checker

Add collision object in a enabled state

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

```
  CollisionShapePtr box(new 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);

  checker.addCollisionObject("box_link", 0, obj1_shapes, obj1_poses);
```

Add collision object in a disabled state

```
  CollisionShapePtr thin_box(new 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);

  checker.addCollisionObject("thin_box_link", 0, obj2_shapes, obj2_poses, false);
```

Add another collision object

```
  std::shared_ptr<tesseract_common::VectorVector3d> ch_verticies(new tesseract_
↪common::VectorVector3d());
  std::shared_ptr<Eigen::VectorXi> ch_faces(new Eigen::VectorXi());
  int ch_num_faces = createConvexHull(*ch_verticies, *ch_faces, mesh_vertices);
  second_box.reset(new ConvexMesh(ch_verticies, ch_faces, ch_num_faces));

  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 object's

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

Set the contact distance threshold

```
checker.setContactDistanceThreshold(0.1);
```

Set the collision object's transform

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

ContactResultVector result_vector;
flattenResults(std::move(result), result_vector);
```

Set the collision object's transform

```
location["box_link"].translation() = Eigen::Vector3d(1.60, 0, 0);
```

```
checker.setCollisionObjectsTransform(location);
```

Perform collision check

Note: The objects are outside the contact threshold

```
checker.contactTest(result, ContactTestType::CLOSEST);
flattenResults(std::move(result), result_vector);
```

Change contact distance threshold

```
  checker.setContactDistanceThreshold(0.25);
```

Perform collision check

Note: The objects are inside the contact threshold

```
  checker.contactTest(result, ContactTestType::CLOSEST);
  flattenResults(std::move(result), result_vector);
```

3.3 Tesseract Geometry Package

3.3.1 Background

This package contains geometries used by Tesseract

3.3.2 Features

1. Primitive Shapes
    - Box
    - Cone
    - Cylinder
    - Plane
    - Sphere
2. Mesh
3. Convex Mesh
4. SDF Mesh
5. Octree

3.3.3 Creating Geometry Shapes

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

using namespace tesseract_geometry;

int main(int /*argc*/, char** /*argv*/)
{
  // Primitive Shapes
```

(continues on next page)

(continued from previous page)

```
  auto box = std::make_shared<tesseract_geometry::Box>(1, 1, 1);
  auto cone = std::make_shared<tesseract_geometry::Cone>(1, 1);
  auto cylinder = std::make_shared<tesseract_geometry::Cylinder>(1, 1);
  auto plane = std::make_shared<tesseract_geometry::Plane>(1, 1, 1, 1);
  auto sphere = std::make_shared<tesseract_geometry::Sphere>(1);

  // 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 cylinder.

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

4. Create a plane.

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

5. Create a sphere.

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

6. 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.

7. Create a signed distance field mesh.

Note: This should be the same as a mesh, but when interperated 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.

8. 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.

9. 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 benificial to prune the octree prior to creating the tesseract octree shap 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**

3.4 Tesseract ROS Package

3.5 Tesseract Msgs Package

3.6 Tesseract Rviz Package

3.7 Tesseract Monitoring Package

3.8 Tesseract Planning Package

3.9 Tesseract Geometry Package

3.9.1 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.

3.9.2 Features

1. New Shapes
   - Capsule
   - Mesh
   - Convex Mesh
   - SDF Mesh
   - Octomap
2. Origin
   - Quaternion

3. 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.

3.9.3 Change URDF Version

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

3.9.4 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 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, shere 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 larget 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 clound 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 compatability 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.

CHAPTER 4

FAQ

4.1 Frequently Asked Questions

This wiki highlights the frequently asked questions on the issue tracker.

1. *Place Holder 1?*
2. *Place Holder 2?*

4.1.1 Place Holder 1?

TBD

4.1.2 Place Holder 2?

TBD