Industrial Training

Jan 15, 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 very few external ROS packages. It currently contains seven packaged described below:

- **tesseract_core** – This package contains only the interface header files and data type structures to be used. This is the only package that does not depend on ROS for the sole purpose to facilitate integration outside of ROS, with little changes, which can sometimes be required.
- **tesseract_ros** – This package is the ROS implementation of the interface identified in the tesseract_core package. It currently leverages orocos libraries for data storage of both the environment and kinematics.
- **tesseract_collision** – This package contains the ROS implementation of a Bullet collision library. It includes both continuous and discrete collision checking for convex-convex and convex-concave shapes.
- **tesseract_msgs** – This package contains the ROS message types used by Tesseract.
- **tesseract_rviz** – This package contains the ROS visualization plugins for Rviz to visualize both the environment state and trajectories.
- **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.
- **tesseract_planning** – This package contains the planners available to be used with the tesseract environment. It currently contains OMPL and TrajOpt.

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

CHAPTER 1

Packages

1.1 Tesseract Core Package

1.2 Tesseract ROS Package

1.3 Tesseract Collison Package

1.3.1 Background

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

1.3.2 Features

1. Add/Remove collison 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 colliison object **tesseract::ContactTestType::ALL**

1.3.3 Discrete Collision Checker Example

```
// Create Bullet Discrete BVH Manager
tesseract::tesseract_bullet::BulletDiscreteBVHManager checker;

// Add box to checker
shapes::ShapePtr box(new shapes::Box(1, 1, 1));
Eigen::Isometry3d box_pose;
box_pose.setIdentity();

std::vector<shapes::ShapeConstPtr> obj1_shapes;
tesseract::VectorIsometry3d obj1_poses;
tesseract::CollisionObjectTypeVector obj1_types;
obj1_shapes.push_back(box);
obj1_poses.push_back(box_pose);
obj1_types.push_back(tesseract::CollisionObjectType::UseShapeType);

checker.addCollisionObject("box_link", 0, obj1_shapes, obj1_poses, obj1_types);

// Add thin box to checker which is disabled
shapes::ShapePtr thin_box(new shapes::Box(0.1, 1, 1));
Eigen::Isometry3d thin_box_pose;
thin_box_pose.setIdentity();

std::vector<shapes::ShapeConstPtr> obj2_shapes;
tesseract::VectorIsometry3d obj2_poses;
tesseract::CollisionObjectTypeVector obj2_types;
obj2_shapes.push_back(thin_box);
obj2_poses.push_back(thin_box_pose);
obj2_types.push_back(tesseract::CollisionObjectType::UseShapeType);

checker.addCollisionObject("thin_box_link", 0, obj2_shapes, obj2_poses, obj2_types,␣
↪false);

// Add second box to checker.
shapes::ShapePtr second_box(new shapes::Box(2, 2, 2));

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

std::vector<shapes::ShapeConstPtr> obj3_shapes;
tesseract::VectorIsometry3d obj3_poses;
tesseract::CollisionObjectTypeVector obj3_types;
obj3_shapes.push_back(second_box);
obj3_poses.push_back(second_box_pose);
obj3_types.push_back(tesseract::CollisionObjectType::UseShapeType);

checker.addCollisionObject("second_box_link", 0, obj3_shapes, obj3_poses, obj3_types);

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

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

(continues on next page)

(continued from previous page)

```
location["box_link"].translation()(1) = 0.1;
location["second_box_link"] = Eigen::Isometry3d::Identity();

checker.setCollisionObjectsTransform(location);

// Perform collision check
tesseract::ContactResultMap result;
checker.contactTest(result, tesseract::ContactTestType::CLOSEST);

tesseract::ContactResultVector result_vector;
tesseract::moveContactResultsMapToContactResultsVector(result, result_vector);

// 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);
checker.contactTest(result, tesseract::ContactTestType::CLOSEST);
tesseract::moveContactResultsMapToContactResultsVector(result, result_vector);

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

checker.setContactDistanceThreshold(0.25);
checker.contactTest(result, tesseract::ContactTestType::CLOSEST);
tesseract::moveContactResultsMapToContactResultsVector(result, result_vector);
```

Example Explanation

Create Contact Checker

There are several available contact checkers.

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

```
tesseract::tesseract_bullet::BulletDiscreteBVHManager checker;
```

Add Collision Objects to Contact Checker

Add collision object in a enabled state

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

```
shapes::ShapePtr box(new shapes::Box(1, 1, 1));
Eigen::Isometry3d box_pose;
box_pose.setIdentity();

std::vector<shapes::ShapeConstPtr> obj1_shapes;
tesseract::VectorIsometry3d obj1_poses;
tesseract::CollisionObjectTypeVector obj1_types;
obj1_shapes.push_back(box);
obj1_poses.push_back(box_pose);
obj1_types.push_back(tesseract::CollisionObjectType::UseShapeType);

checker.addCollisionObject("box_link", 0, obj1_shapes, obj1_poses, obj1_types);
```

Add collision object in a disabled state

```
shapes::ShapePtr thin_box(new shapes::Box(0.1, 1, 1));
Eigen::Isometry3d thin_box_pose;
thin_box_pose.setIdentity();

std::vector<shapes::ShapeConstPtr> obj2_shapes;
tesseract::VectorIsometry3d obj2_poses;
tesseract::CollisionObjectTypeVector obj2_types;
obj2_shapes.push_back(thin_box);
obj2_poses.push_back(thin_box_pose);
obj2_types.push_back(tesseract::CollisionObjectType::UseShapeType);

checker.addCollisionObject("thin_box_link", 0, obj2_shapes, obj2_poses, obj2_types,␣
↪false);
```

Add another collision object

```
shapes::ShapePtr second_box(new shapes::Box(2, 2, 2));

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

std::vector<shapes::ShapeConstPtr> obj3_shapes;
tesseract::VectorIsometry3d obj3_poses;
tesseract::CollisionObjectTypeVector obj3_types;
obj3_shapes.push_back(second_box);
obj3_poses.push_back(second_box_pose);
obj3_types.push_back(tesseract::CollisionObjectType::UseShapeType);

checker.addCollisionObject("second_box_link", 0, obj3_shapes, obj3_poses, obj3_types);
```

Set the collision object's transform

```
tesseract::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

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

tesseract::ContactResultVector result_vector;
tesseract::moveContactResultsMapToContactResultsVector(result, result_vector);
```

1.4 Tesseract Msgs Package

1.5 Tesseract Rviz Package

1.6 Tesseract Monitoring Package

1.7 Tesseract Planning Package

CHAPTER 2

FAQ

2.1 Frequently Asked Questions

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

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

2.1.1 Place Holder 1?

TBD

2.1.2 Place Holder 2?

TBD