Mauricio Bethoven Tigauw
1103204099

Technical Report: ROS Geometry and Visualization Conversions for Drake

SC :

```
"""
ROS geometry and visualization conversions for Drake.
"""

import copy
from io import BytesIO

import numpy as np

# ROS1 Messages.
from geometry_msgs.msg import Pose, Transform, TransformStamped
from tf2_msgs.msg import TFMessage
from visualization_msgs.msg import Marker, MarkerArray
# ROS1 API.
from rospy import Duration

from pydrake.common.eigen_geometry import Quaternion
from pydrake.geometry import (
    QueryObject,
    Role,
    Rgba,
    SceneGraphInspector,
    FrameId,
    Shape,
    Box, Sphere, Cylinder, Mesh, Convex,
)
from pydrake.math import RigidTransform
```

```
def _write_pose_msg(X_AB, p, q):
    # p - position message
    # q - quaternion message
    X_AB = RigidTransform(X_AB)
    p.x, p.y, p.z = X_AB.translation()
    q.w, q.x, q.y, q.z = X_AB.rotation().ToQuaternion().wxyz()


def to_ros_pose(X_AB):
    """Converts Drake transform to ROS pose."""
    msg = Pose()
    _write_pose_msg(X_AB, p=msg.position, q=msg.orientation)
    return msg


def to_ros_transform(X_AB):
    """Converts Drake transform to ROS transform."""
    msg = Transform()
    _write_pose_msg(X_AB, p=msg.translation, q=msg.rotation)
    return msg


def _read_pose_msg(p, q):
    # p - position message
    # q - quaternion message
    return RigidTransform(
        Quaternion(wxyz=[q.w, q.x, q.y, q.z]), [p.x, p.y, p.z])


def from_ros_pose(pose):
```

```
"""Converts ROS pose to Drake transform."""
    return _read_pose_msg(p=pose.position, q=pose.orientation)



def from_ros_transform(tr):
    """Converts ROS transform to Drake transform."""
    return _read_pose_msg(p=tr.translation, q=tr.rotation)



DEFAULT_RGBA = Rgba(0.9, 0.9, 0.9, 1.0)



def to_ros_markers(shape, stamp, frame_name, X_FG, color_rgba):
    assert isinstance(shape, Shape), shape
    assert isinstance(frame_name, str), frame_name
    assert isinstance(X_FG, RigidTransform), X_FG
    marker = Marker()
    marker.header.stamp = stamp
    marker.header.frame_id = frame_name
    marker.pose = to_ros_pose(X_FG)
    marker.action = Marker.ADD
    marker.lifetime = Duration(0.)
    marker.frame_locked = True

    def use_color():
        nonlocal color_rgba
        if color_rgba is None:
            color_rgba = DEFAULT_RGBA
        (marker.color.r, marker.color.g, marker.color.b, marker.color.a) = (
            color_rgba.r(), color_rgba.g(), color_rgba.b(), color_rgba.a())
```

```
if type(shape) == Box:
        marker.type = Marker.CUBE
        marker.scale.x, marker.scale.y, marker.scale.z = shape.size()
        use_color()
        return [marker]
    if type(shape) == Sphere:
        marker.type = Marker.SPHERE
        marker.scale.x = shape.radius()
        marker.scale.y = shape.radius()
        marker.scale.z = shape.radius()
        use_color()
        return [marker]
    elif type(shape) == Cylinder:
        marker.type = Marker.CYLINDER
        marker.scale.x = shape.radius()
        marker.scale.y = shape.radius()
        marker.scale.z = shape.length()
        use_color()
        return [marker]
    elif type(shape) in (Mesh, Convex):
        marker.type = Marker.MESH_RESOURCE
        marker.mesh_resource = f"file://{shape.filename()}"
        # TODO(eric.cousineau): Is this the correct way to handle color vs.
        # texture?
        use_color()
        marker.mesh_use_embedded_materials = True
        marker.scale.x, marker.scale.y, marker.scale.z = 3 * [shape.scale()]
        return [marker]
    else:
        # TODO(eric): Support Capsule, Ellipse, all dat jazz.
        assert False, f"Unsupported type: {shape}"
```

```
def get_role_properties(inspector, role, geometry_id):
    """Permits dynamic role retrival."""
    assert isinstance(inspector, SceneGraphInspector), inspector
    if role == Role.kProximity:
        return inspector.GetProximityProperties(geometry_id)
    elif role == Role.kIllustration:
        return inspector.GetIllustrationProperties(geometry_id)
    elif role == Role.kPerception:
        return inspector.GetPerceptionProperties(geometry_id)
    assert False, role



def sanity_check_query_object(query_object, debug=False):
    # Ensures that we have a "sane" query_object for the purpose of ROS1
    # visualization.
    # The primary goal is to ensure that each geometry and frame ultimately has
    # a unique name.
    assert isinstance(query_object, QueryObject), query_object
    frame_ids = set()
    frame_names = set()
    geometry_ids = set()
    geometry_names = set()

    inspector = query_object.inspector()
    for geometry_id in inspector.GetAllGeometryIds():
        geometry_name = inspector.GetName(geometry_id)
        # TODO(eric.cousineau): Expose inspector.get_all_frames().
        frame_id = inspector.GetFrameId(geometry_id)
        frame_name = inspector.GetName(frame_id)
        geometry_ids.add(geometry_id)
        geometry_names.add(geometry_name)
```

```
frame_ids.add(frame_id)
        frame_names.add(frame_name)
        if debug:
            print(f"Geometry: {geometry_name} -> Frame: {frame_name}")
    assert len(frame_ids) == len(frame_names)
    assert len(geometry_ids) == len(geometry_names)



def to_ros_marker_array(query_object, role, stamp):
    assert isinstance(query_object, QueryObject), query_object
    assert isinstance(role, Role), role
    inspector = query_object.inspector()
    marker_array = MarkerArray()
    for geometry_id in inspector.GetAllGeometryIds():
        shape = inspector.GetShape(geometry_id)
        frame_id = inspector.GetFrameId(geometry_id)
        X_FG = inspector.GetPoseInFrame(geometry_id)
        frame_name = inspector.GetName(frame_id)
        properties = get_role_properties(inspector, role, geometry_id)
        if properties is None:
            # This role is not assigned for this geometry. Skip.
            continue
        # TODO(eric): Fix this :(
        color_rgba = None
        if properties.HasProperty("phong", "diffuse"):
            color_rgba = properties.GetProperty("phong", "diffuse")
        marker_array.markers += to_ros_markers(
            shape, stamp, frame_name, X_FG, color_rgba)
    # Ensure unique IDs.
    for i, marker in enumerate(marker_array.markers):
        # TODO(eric.cousineau): Reflect the namespace like Calder does for
```

```
# collision, visual, etc.
        marker.id = i
    return marker_array



def to_ros_tf_message(query_object, stamp):
    assert isinstance(query_object, QueryObject), query_object
    inspector = query_object.inspector()
    frame_ids = set()
    for geometry_id in inspector.GetAllGeometryIds():
        frame_ids.add(inspector.GetFrameId(geometry_id))
    frame_ids = sorted(list(frame_ids))
    tf_message = TFMessage()
    for frame_id in frame_ids:
        frame_name = inspector.GetName(frame_id)
        X_WF = query_object.X_WF(frame_id)
        transform = TransformStamped()
        transform.header.frame_id = "world"
        transform.header.stamp = stamp
        transform.child_frame_id = frame_name
        transform.transform = to_ros_transform(X_WF)
        tf_message.transforms.append(transform)
    return tf_message



def serialize_message(msg):
    # https://answers.ros.org/question/303115/serialize-ros-message-and-pass-it
    buff = BytesIO()
    msg.serialize(buff)
    return buff.getvalue()
```

```
def compare_message(a, b):
    # Naive compare.
    assert type(a) == type(b), f"{type(a)} != {type(b)}"
    data_a = serialize_message(a)
    data_b = serialize_message(b)
    return data_a == data_b
```

1. Introduction: The purpose of this technical report is to present the implementation details of ROS (Robot Operating System) geometry and visualization conversions for the Drake software framework. The conversions discussed in this report enable seamless interoperability between Drake and ROS by providing functions to convert Drake objects to ROS messages and vice versa.
2. Conversion Functions: The following conversion functions are implemented to facilitate the conversion between Drake and ROS representations:
    - **to_ros_pose(X_AB)**: Converts a Drake transform (**X_AB**) to a ROS pose message.
    - **to_ros_transform(X_AB)**: Converts a Drake transform (**X_AB**) to a ROS transform message.
    - **from_ros_pose(pose)**: Converts a ROS pose message to a Drake transform.
    - **from_ros_transform(tr)**: Converts a ROS transform message to a Drake transform.
3. ROS Marker Conversion: The conversion functions also support converting Drake geometric shapes to ROS marker messages. The following functions are used for this purpose:
    - **to_ros_markers(shape, stamp, frame_name, X_FG, color_rgba)**: Converts a Drake shape to ROS marker messages. It supports shapes such as Box, Sphere, Cylinder, Mesh, and Convex.
    - **to_ros_marker_array(query_object, role, stamp)**: Converts a Drake query object to a ROS marker array. It retrieves the geometry and frame information from the query object and assigns appropriate markers to each geometry.
4. ROS TF Message Conversion: Drake's query object is converted to a ROS TF message, which represents the transforms between different frames. The following function is used for this conversion:
    - **to_ros_tf_message(query_object, stamp)**: Converts a Drake query object to a ROS TF message. It retrieves the frame information from the query object and creates appropriate transform messages for each frame.

5. Utility Functions: The technical report also includes utility functions for sanity checks and comparison:
    - **sanity_check_query_object(query_object, debug=False)**: Performs a sanity check on the query object to ensure unique names for frames and geometries.
    - **compare_message(a, b)**: Compares two ROS messages for equality.
6. Conclusion: The implementation of these conversion functions facilitates the seamless integration of Drake and ROS. They enable the conversion of Drake transforms, shapes, and query objects to their corresponding ROS representations, allowing users to leverage both frameworks effectively.

Please note that the code snippets provided in this report are excerpts and may require additional context or dependencies to work properly. They serve as an illustration of the concepts discussed in the report.