Representation of kinematic quantities in ROS

ME 416 - Prof. Tron

Thursday 28th February, 2019

1 Representation of rotations

In robotics, there are at least four main ways to represent rotations: rotation matrices, angle-axis, Euler angles, quaternions.

Note: In addition to the specific representation choice, it is also always necessary to also specify or keep track of the direction of the transformation (e.g., world-to-body or body-to-world).

1.1 Rotation matrices

Rotation matrices are the most unambiguous, direct way to represent rotations. They can be directly used to transform points and vectors using linear algebra (matrix-vector multiplication); the space of rotations is defined in terms of matrices.

\+ Easy to use to transform vectors (simple linear algebra).

\+ Representation most close to the definition.

\+ Unambiguous.

\- Very overparametrized: 9 elements in the matrix for 3 degrees of freedom.

-/+ Computationally not the cheapest, but the easiest to implement.

1.2 Angle-axis

By Euler's rotation theorem, any rotation can be expressed by defining a rotation axis w, and a rotation angle θ. E.g., a rotation in the x-y plane can be defined with an axis in the z direction. Since the magnitude of w is not important (only its direction), it is customary to assume $\|w\| = 1$, or combine the two by choosing $\|w\| = \theta$. For computations, it is necessary to transform this representation in a rotation matrix (using Rodrigues' rotation formula) or a quaternion (see below).

+/- Minimal or almost-minimal parametrization (three or four parameters for three degrees of freedom).

\+ Interpretation is somewhat intuitive (especially for particular cases).

\- Ambiguity in the representation for some cases (choice of w when $\theta = 0$, or sign of θ and w when $\theta = \pm\pi$).

\+ Computationally is the slowest.

1.3 Euler angles

Any rotation can be represented by composing three *elemental* rotations along two or three pre-determined axes, i.e., $R = R_{w_x}(\theta_x)R_{w_y}(\theta_y)R_{w_z}(\theta_z)$ (where $R_w(\theta)$ represents a rotation around axis v by θ radians). Hence, once the axes are determined (e.g., the canonical x, y, and z axes), it is sufficient to give the *Euler angles* (the θ's) corresponding to this rotation.

+ Minimal parametrization (three parameters for three degrees of freedom).
+ In some situation (e.g., aerospace and nautical applications), they have an intuitive meaning (e.g., roll, pitch, and yaw).
- Although any rotation can be represented through Euler angles, different Euler angles might correspond to the same rotation (this is sometimes referred to as *gimbal lock*).
- There are *many* versions of Euler angles, given by different choices of the rotation axes (especially whether one axis is repeated or not), and in what reference frame they are expressed (original or rotated frame). If you work with Euler angles, you need to make sure that these choices are well defined. It is best to use any computational library that you might have.
- Computationally, they are not as fast as quaternions.

1.4 Quaternions

Instead of handling separately the axis w and angle θ from the angle/axis representation, these can be combined into a *quaternion* vector, which has the form

$$q = \begin{bmatrix} \sin(\theta)\omega \\ \cos(\theta) \end{bmatrix}. \tag{1}$$

For instance, for $R = I$ (no rotation), the corresponding quaternion is $q = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$. From the definition above, quaternions have also unit lengths, $\|q\| = 1$.

★★★: The last component of a quaternion is sometimes referred to as a the *real* part of the quaternion, and the others as the *imaginary* part.

+ Computationally the fastest (quaternion-vector and quaternion-quaternion multiplications can be done without any intermediate conversion).

+/- Almost minimal parametrization (four parameters for three degrees of freedom).

-/+ Only one ambiguity: q and $-q$ represent the same rotation.

- Harder to interpret.
- Sometimes the real and imaginary parts are reversed in the vector.

2 Representation in ROS (message definitions)

Points, rigid body transformations, and velocities are quantities that commonly appear as types for ROS topics. As such, they have been standardized in the system package `geometry_msgs`.

General convention To avoid redundant definitions of messages for 2-D and 3-D cases, the almost-universal convention in ROS is to embed 2-D quantities in their 3-D counterparts by simply ignoring the non-relevant field (the main exception to this is the existence of the `Pose2D` message type). For instance, there is no message definition for a 2-D point; instead, one is expected to use the 3-D definition and then leave the field for the z direction empty.

Note Each subsection below contains reproduces the raw definition of the message type (i.e., what is written in the respective `.msg` file in the `msg` directory of the `geomety_msgs` package).

2.1 Points (geometry_msgs/Point)

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

The three fields contain the three coordinates of the point.

Point32 There is also a variation of `Point` called `Point32`, which uses floating point numbers with less precision and less memory. These should be used only when bandwidth or memory storage is an issue (e.g., when transmitting point in a point cloud over a wireless connection).

2.2 Vectors (geometry_msgs/Vector3)

```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

The three fields contain the three coordinates of the vector. See the comment in the definition for the difference between `Point` and `Vector3` (it is the same as the distinction done in the theory).

2.3 Rotations (expressed as quaternions, geometry_msgs/Quaternion

```
float64 x
float64 y
float64 z
float64 w
```

The `x`, `y`, `z` fields contain the imaginary part of the quaternion, while `w` the real part.

2.4 Poses (geometry_msgs/Pose)

```
Point position
Quaternion orientation
```

This message contain two fields, one for translation (of type `Point`), the other for rotation (of type `Quaternion`). Each one of these fields has subfields as specified by that type (see above).

2.5 Twists (i.e., velocities for poses, geometry_msgs/Twist)

```
Vector3  linear
Vector3  angular
```

Represents the velocity of a pose (R, T), where the field `linear` corresponds to $\dot{T}$, and `angular` correspond to the angular velocity ω in $\dot{R} = \hat{\omega} R$.

★★The convention is to express these velocities in body-frame coordinates (i.e., to use $({}^{\mathcal{B}}R_{\mathcal{W}}, {}^{\mathcal{B}}T_{c\mathcal{W}})$).

Twists for 2-D poses For 2-D poses (e.g., for ROSBot), only the fields `linear.x`, `linear.y`, and `angular.z` are used.