Robotics Prototyping Package Math and Algorithms

Ben Kolligs

Contents

1 Core Classes **2**
1.1 Transform 2
1.2 Transforming between frames 3

2 Pointcloud **4**

3 Rigid Collection **5**

4 Kinematic Tree **5**
4.1 Representation 6

1 Core Classes

1.1 Transform

The main class of the core of the package is the Transform class. This class performs the functionality of a homogeneous rigid transform, which shows up in many places within robotics and robotics adjacent fields, like computer vision and graphics.

The homogeneous rigid transform is an extremely useful tool used to represent various things in robotics. First, the rigid transform is defined as a transformation T that when acting on any vector v, produces a transformed vector $T(v)$ of the form:

$$T(v) = Rv + t \tag{1}$$

where $R^T R^{-1}$, and t is a vector giving the translation of the origin. This concept can be represented in a form called a *homogeneous transformation matrix*. A homogeneous transformation matrix $T \in \mathbb{R}^{4\times4}$ is a member of special euclidean group $SE(3)$, and can be written in the form:

$$T = \begin{bmatrix} R & t \\ \mathbf{0} & 1 \end{bmatrix} \tag{2}$$

where $R \in \mathbb{R}^{3\times3}$ is a rotation matrix, and $t \in \mathbb{R}^3$ is a translation vector. The inverse of a homogeneous transformation matrix is:

$$T^{-1} = \begin{bmatrix} R^T & -R^T t \\ \mathbf{0} & 1 \end{bmatrix} \tag{3}$$

$$TT^{-1} = \mathbf{I} \tag{4}$$

When using super and subscripts to specify the frames we are referring to, then $T_b^a = (T_a^b)^{-1}$. We can use a homogeneous transformation to operate on homogeneous points $p, q \in \mathbb{R}^4$,

$$q = Tp \tag{5}$$

$$\begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}. \tag{6}$$

In general, there are three interpretations for homogeneous transformations:

1. A description of relative orientation and translation between frames.

2. A coordinate transform between frames. Specifically, T^i_j is a transform from frame j to frame i:
$$P^i = T^i_j P^j \tag{7}$$

3. A motion of a point (or a collection of points) within a single frame. For example, as we saw above point p can be moved to point q:
$$q = Tp \tag{8}$$
$$P^i_2 = T^i_j P^i_1 \tag{9}$$
Similarly, a homogeneous transform can represent a motion from one frame to another. Specifically, T^i_j moves frame i to frame j.

It is important to specify which of these interpretations you are using within your program, as it can get confusing what these transformations represent if people aren't on the same page.

Additionally, one must pay attention to the frame that the operand is in, as this determines the function of the transform to some degree. $P^i_2 = T^i_j P^i_1$ is a motion that operates on point P^i_1 within frame i whereas $P^i = T^i_j P^j$ is a coordinate transformation from j to i, and therefore if we feed a point in frame j to this transformation, the point will not move, but the coordinates will be transformed into frame i, which is often desired.

1.2 Transforming between frames

We can chain together homogeneous transforms to represent a traversal through frames.

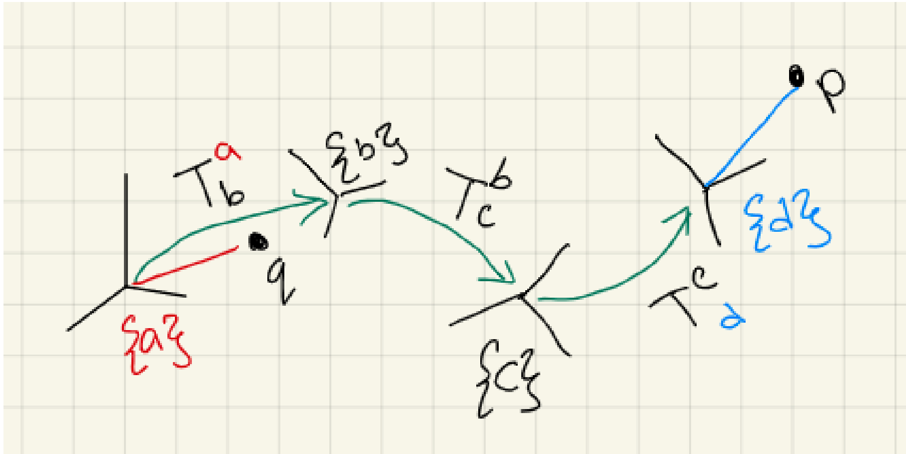

Figure 1: Traversing through different coordinate frames represented by rigid transformation matrices.

Say we want the transform from a to d:

$$T_d^a = T_b^a T_c^b T_d^c. \tag{10}$$

Notice that the bottom subscript and the following top subscript "cancel" out when multiplying frames together. Additionally when composing transforms like this, the "source" frame (in this case d) is included in the right most transform, and the "destination" frame a, is included in the left most.

Using this knowledge we can represent frames that are related in this way by a pose graph, or pose tree consisting of parent and child relationships between each frame.

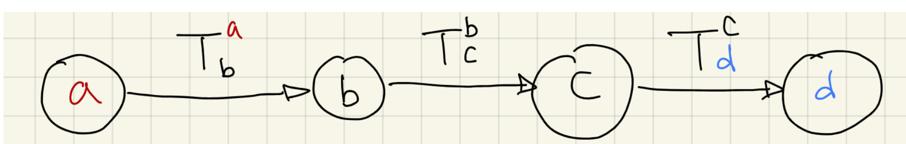

Figure 2: Frames and their transforms can be represented in graph form.

2 Pointcloud

The pointcloud class contains a list of 3D points in space. When transforming the entire pointcloud, the class simply creates homogeneous points out of the point list, and then multiplies a given homogeneous transform by this array. So if we want to transform the coordinates of pointcloud C^i from frame i into frame j:

$$C^i = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ y_1 & y_2 & y_3 & \dots & y_n \\ z_1 & z_2 & z_3 & \dots & z_n \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \tag{11}$$

Then we need to multiply by the transform T_i^j:

$$C^j = T_i^j C^i. \tag{12}$$

3 Rigid Collection

The rigid collection class $\mathcal{T}$ is simply a set of transforms, $\mathcal{T} = \{T_1, T_2, T_3, \dots T_n\}$. It is required that this list of transforms be expressed in the same datum frame, that is:

$$\mathcal{T}^i = \{T_1^i, T_2^i, T_3^i, \dots, T_n^i\} \tag{13}$$

The rigid collection is able to be graphed in its "parent" frame i, which allows all contained transforms to be visualized. The collection is able to add transforms to itself by simply appending new transforms to the set.

4 Kinematic Tree

The kinematic tree is an abstract representation of a collection of transforms. Unlike the rigid collection (section 3), the kinematic tree is able to represent transforms in multiple datum frames. The tree can then use the network of frames to "lookup" a transform between any two connected nodes, as hinted at in figure 2. This concept is widely used in robotics and therefore has a place in the prototyping package. There are several algorithms this class implements:

1. Representation: Construct a tree representation given only edges
2. Lookup: apply breadth first graph search to find a path between any two nodes on the tree
3. Root: express all frames in the base link frame or another specified frame on the tree

4.1 Representation

Frames in a kinematic tree can have a child frame, a parent frame, or both. Frame T is described as:

$$T_i^{p(i)} = \begin{bmatrix} R_i^{p(i)} & t_i^{p(i)} \\ \mathbf{0} & 1 \end{bmatrix} \tag{14}$$

where i is the child frame, and $p(i)$ is the parent frame. The kinematic tree has a root, which we can also call the ”base link”.