$$
\begin{aligned}
{}^{A}\tilde{\boldsymbol{p}} &= \begin{pmatrix} {}^{A}\boldsymbol{R}_B & \boldsymbol{t} \\ \boldsymbol{0}_{1\times3} & 1 \end{pmatrix} {}^{B}\tilde{\boldsymbol{p}} \\
&= {}^{A}\boldsymbol{T}_B \, {}^{B}\tilde{\boldsymbol{p}}
\end{aligned}
\tag{2.19}
$$

and ${}^{A}\boldsymbol{T}_B$ is a 4×4 homogeneous transformation. The matrix has a very specific structure and belongs to the special Euclidean group of dimension 3 or $\boldsymbol{T} \in SE(3) \subset \mathbb{R}^{4\times4}$.

A concrete representation of relative pose ξ is $\xi \sim \boldsymbol{T} \in SE(3)$ and $\boldsymbol{T}_1 \oplus \boldsymbol{T}_2 \mapsto \boldsymbol{T}_1\boldsymbol{T}_2$ which is standard matrix multiplication.

$$
\boldsymbol{T}_1\boldsymbol{T}_2 = \begin{pmatrix} \boldsymbol{R}_1 & \boldsymbol{t}_1 \\ \boldsymbol{0}_{1\times3} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{R}_2 & \boldsymbol{t}_2 \\ \boldsymbol{0}_{1\times3} & 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{R}_1\boldsymbol{R}_2 & \boldsymbol{t}_1 + \boldsymbol{R}_1\boldsymbol{t}_2 \\ \boldsymbol{0}_{1\times3} & 1 \end{pmatrix}
\tag{2.20}
$$

One of the rules of pose algebra from page 18 is $\xi \oplus 0 = \xi$. For matrices we know that $\boldsymbol{TI} = \boldsymbol{T}$, where $\boldsymbol{I}$ is the identify matrix, so for pose $0 \mapsto \boldsymbol{I}$ the identity matrix. Another rule of pose algebra was that $\xi \ominus \xi = 0$. We know for matrices that $\boldsymbol{TT}^{-1} = \boldsymbol{I}$ which implies that $\ominus\boldsymbol{T} \mapsto \boldsymbol{T}^{-1}$

$$
\boldsymbol{T}^{-1} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}_{1\times3} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{R}^T & -\boldsymbol{R}^T\boldsymbol{t} \\ \boldsymbol{0}_{1\times3} & 1 \end{pmatrix}
\tag{2.21}
$$

The 4×4 homogeneous transformation is very commonly used in robotics and computer vision, is supported by the Toolbox and will be used throughout this book as a concrete representation of 3-dimensional pose.

The Toolbox has many functions to create homogeneous transformations. For example we can demonstrate composition of transforms by

```
>> T = transl(1, 0, 0) * trotx(pi/2) * transl(0, 1, 0)
T =
    1.0000         0         0    1.0000
         0    0.0000   -1.0000    0.0000
         0    1.0000    0.0000    1.0000
         0         0         0    1.0000
```

The function `transl` create a relative pose with a finite translation but no rotation, and `trotx` returns a 4×4 homogeneous transform matrix corresponding to a rotation of $\frac{\pi}{2}$ about the x-axis: the rotation part is the same as `rotx(pi/2)` and the translational component is zero.▸ We can think of this expression as representing a walk along the x-axis for 1 unit, then a rotation by 90° about the x-axis and then a walk of 1 unit along the new y-axis which was the previous z-axis. The result, as shown in the last column of the resulting matrix is a translation of 1 unit along the original x-axis and 1 unit along the original z-axis. The orientation of the final pose shows the effect of the rotation about the x-axis. We can plot the corresponding coordinate frame by

Many Toolbox functions have variants that return orthonormal rotation matrices or homogeneous transformations, for example, `rotx` and `trotx`, `rpy2r` and `rpy2tr` etc. Some Toolbox functions accept an orthonormal rotation matrix or a homogeneous transformation and ignore the translational component, for example, `tr2rpy` or `Quaternion`.

```
>> trplot(T)
```

The rotation matrix component of `T` is

```
>> t2r(T)
ans =
    1.0000         0         0
         0    0.0000   -1.0000
         0    1.0000    0.0000
```

and the translation component is a vector

```
>> transl(T)'
ans =
    1.0000    0.0000    1.0000
```

Table 2.1.
Summary of the various concrete representations of pose ξ introduced in this chapter

Representation	$\oplus$	$\ominus$	transl.	rotn.	dim	MATLAB
$(x, y, \theta) \in \mathbb{R}^2 \times \mathbb{S}$			✓	✓	2D	
$T \in SE(2)$	$T_1 T_2$	T^{-1}	✓	×	2D	`se2(x, y)`
$R \in SO(2)$	$R_1 R_2$	R^T	×	✓	2D	`se2(0, 0, th)`
$T \in SE(2)$	$T_1 T_2$	T^{-1}	✓	✓	2D	`se2(x, y, th)`
$(x, y, z, \Gamma) \in \mathbb{R}^3 \times \mathbb{S}^3$			✓	✓	3D	
$R \in SO(3)$	$R_1 R_2$	R^T	×	✓	3D	`rotx,roty,…`
$\Gamma \in \mathbb{S}^3$	×			✓	3D	`tr2eul,eul2tr`
$\Gamma \in \mathbb{S}^3$	×			✓	3D	`tr2rpyl,rpy2tr`
$T \in SE(3)$	$T_1 T_2$	T^{-1}	✓	✓	3D	`transl(x,y,z)`
$\mathring{q} \in \mathbb{Q}$	$\mathring{q}_1 \mathring{q}_2$	$\mathring{q}^{-1}$	×	✓	3D	`quaternion`

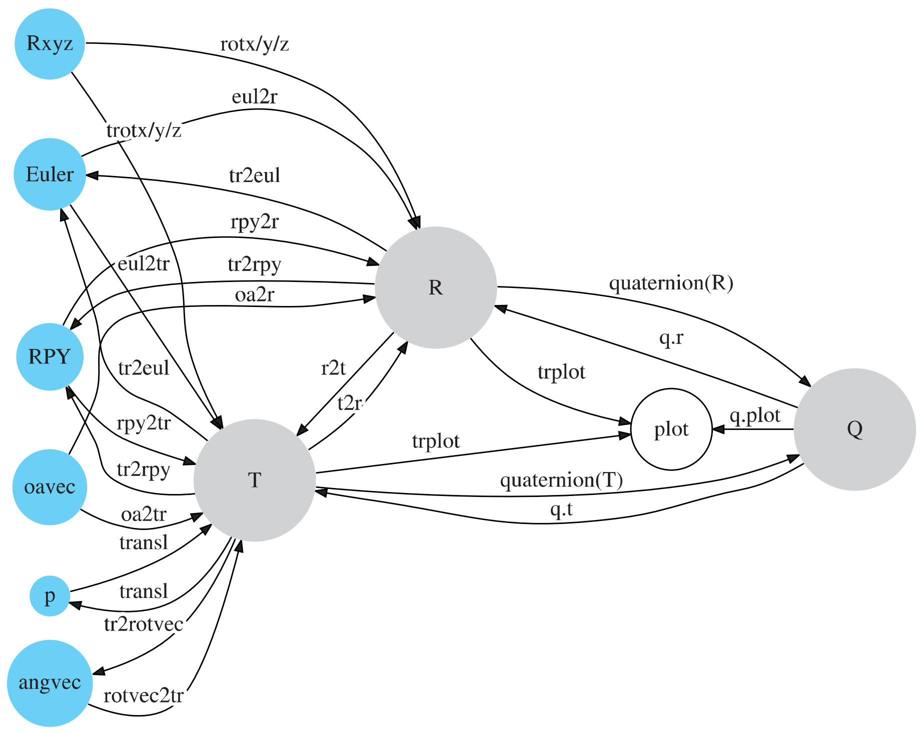

Fig. 2.15.
Conversion between rotational representations

2.3 Wrapping Up

In this chapter we learned how to represent points and poses in 2- and 3-dimensional worlds. Points are represented by coordinate vectors relative to a coordinate frame. A set of points that belong to a rigid object can be described by a coordinate frame, and its constituent points are described by displacements from the object's coordinate frame. The position and orientation of any coordinate frame can be described relative to another coordinate frame by its relative pose ξ. Relative poses can be applied sequentially (composed or compounded), and we have shown how relative poses can be manipulated algebraically. An important algebraic rule is that composition is non-commutative – the order in which relative poses are applied is important.

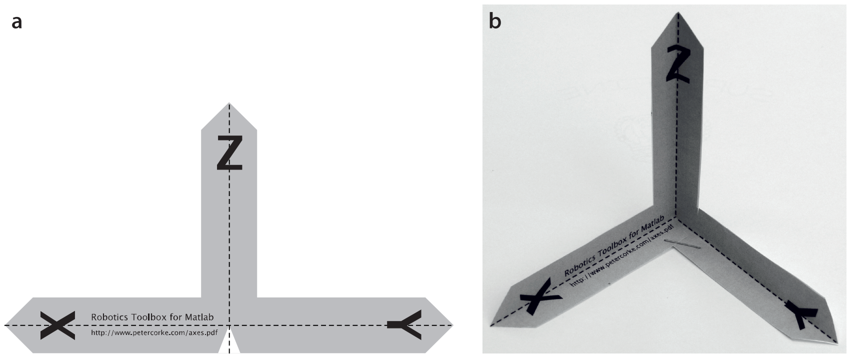
a
b
Z
X
Y
Robotics Toolbox for Matlab
http://www.petercorke.com/axes.pdf

Fig. 2.16.
Build your own coordinate frame. **a** Get the PDF file from http://www.petercorke.com/axes.pdf; **b** cut it out, fold along the dotted lines and add a staple. Voila!

We have explored orthonormal rotation matrices for the 2- and 3-dimensional case to represent orientation and its extension, the homogeneous transformation matrix, to represent orientation and translation. Rotation in 3-dimensions has subtlety and complexity and we have looked at other representations such as Euler angles, roll-pitch-yaw angles and quaternions. Some of these mathematical objects are supported natively by MATLAB® while others are supported by functions or classes within the Toolbox.

There are two important lessons from this chapter. The first is that there are *many* mathematical objects that can be used to represent pose and these are summarized in Table 2.1. There is no right or wrong – each has strengths and weaknesses and we typically choose the representation to suit the problem at hand. Sometimes we wish for a vectorial representation in which case (x, y, θ) or $(x, y, z, \boldsymbol{\Gamma})$ might be appropriate, but this representation cannot be easily compounded. Sometime we may only need to describe 3D rotation in which case $\boldsymbol{\Gamma}$ or $\mathring{q}$ is appropriate. The Toolbox supports conversions between many different representations as shown in Fig. 2.15. In general though, we will use homogeneous transformations throughout the rest of this book.

The second lesson is that coordinate frames are your friend. The essential first step in many vision and robotics problems is to assign coordinate frames to all objects of interest, indicate the relative poses as a directed graph, and write down equations for the loops. Figure 2.16 shows you how to build a coordinate frame out of paper that you can pick up and rotate.

We now have solid foundations for moving forward. The notation has been defined and illustrated, and we have started our hands-on work with MATLAB®. The next chapter discusses coordinate frames that change with time, and after that we are ready to move on and discuss robots.

Further Reading

The treatment in this chapter is a hybrid mathematical and graphical approach that covers the 2D and 3D cases by means of abstract representations and operators which are later made tangible. The standard robotics textbooks such as Spong et al. (2006), Craig (2004), Siciliano et al. (2008) and Paul (1981) all introduce homogeneous transformation matrices for the 3-dimensional case but differ in their approach. These books also provide good discussion of the other representations such as angle-vector and 3-angle representations. Spong et al. (2006, Sec 2.5.1) have a good discussion of singularities. Siegwart et al. (2011) explicitly cover the 2D case in the context of mobile robot navigation.

Hamilton and his supporters, including Peter Tait, were vigourous in defending Hamilton's precedence in inventing quaternions, and for muddying the water with respect to vectors which were then beginning to be understood and used. Rodrigues developed the key idea in 1840 and Gauss discovered it in 1819 but, as usual, did not