2 Homogeneous Transformation

In order to compute robot kinematics, we need to describe mathematically the relation of some position and angle of an object, say, the end-effector of a robot arm, with respect to a reference coordinate. For a complicated manipulator, that relationship is normally difficult to find in a single step. Instead, intermediate coordinate axes, called *frames* (or *poses*) in robotic textbooks, are placed properly between adjacent joints. Then the overall kinematics are computed as a product of matrices representing relationship between frames, from base upwards. Homogeneous transformation refers to the mathematics used to form a matrix that describes the location and orientation of a frame with respect to another reference frame.

In general, six degree of freedoms are required to describe a relation between two frames in 3 dimensional space: 3 for orientation and 3 for location. In this chapter we start by discussing frame rotation and translation separately, then combine them to a homogeneous transformation matrix.

2.1 Rotation in 3-D

Rotational relation between two frames can be described as having an original coordinate axes, called fixed or world frame, then rotate that frame either with some basic operation or arbitrarily. So the origins of the two frames remain at same position. Figure 2.1 shows a world frame $x_0y_0z_0$, or using shorthand notation {0}, and a rotated frame {1} ($x_1y_1z_1$).

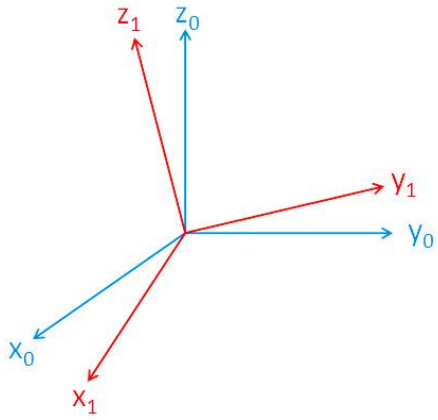


Figure 2.1. Rotation between two frames

Assume that x_i, y_i, z_i are unit vectors, we want to find a math expression that describes the rotation; i.e., the orientation of {1} w.r.t {0}. It can be shown that the relationship is

$$ {}^{0}R_{1} = \begin{bmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 & z_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot y_0 & z_1 \cdot y_0 \\ x_1 \cdot z_0 & y_1 \cdot z_0 & z_1 \cdot z_0 \end{bmatrix} \qquad (2.1) $$

called a *rotation matrix*. This is derived by projecting each of {1} axis onto each of {0}'s . Suppose, however, that we want to describe {0} w.r.t {1}, the corresponding rotation matrix is then

$$ {}^{1}R_{0} = \begin{bmatrix} x_0 \cdot x_1 & y_0 \cdot x_1 & z_0 \cdot x_1 \\ x_0 \cdot y_1 & y_0 \cdot y_1 & z_0 \cdot y_1 \\ x_0 \cdot z_1 & y_0 \cdot z_1 & z_0 \cdot z_1 \end{bmatrix} \tag{2.2} $$

By the commutative property of dot product, we have

$$ {}^{1}R_{0} = {}^{0}R_{1}^{T} \tag{2.3} $$

Meanwhile, the operations (2.1) and (2.2) are inverses of each other; i.e.,

$$ {}^{1}R_{0} = {}^{0}R_{1}^{-1} \tag{2.4} $$

Hence,

$$ {}^{0}R_{1}^{T} = {}^{0}R_{1}^{-1} \tag{2.5} $$

Any matrix that has its transpose equal its inverse like in (2.5) is called an *orthogonal matrix*.

For the special case of rotation around main axis X_0, Y_0, Z_0 of {0}, it can be shown by using this dot product identity for vectors in 3-D

$$ v_0 \cdot v_1 = \|v_0\| \|v_1\| \cos\theta \tag{2.6} $$

that the resulting rotation matrices are

$$ R_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \; R_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \; R_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7} $$

Each of these is called a *basic rotation matrix*.

The basic rotation matrices in (2.7) are implemented in RTSX with function names `rotx`, `roty`, `rotz`. The argument is rotation angle, and option to specify value in degree if you wish.

```
-->R=rotx(pi/2)
 R  =
    1.    0.    0.
    0.    0.  - 1.
    0.    1.    0.
```

```
-->R=rotz(30,'deg')
R  =
   0.8660254  - 0.5          0.
   0.5          0.8660254    0.
   0.           0.           1.
```

To visualize frame rotations, use trpot for still image and tranimate for motion. For example, suppose we want to see the rotation of 45 degrees about X. These commands

```
-->R=rotx(pi/4);
-->trplot(R,'world');
-->tranimate(R, 'world');

Generating trajectory (this
process takes some time)
Generating animation data
.............................
.................
Enter [Y] to replay,[s] to
change speed,  any other key to
quit:
```

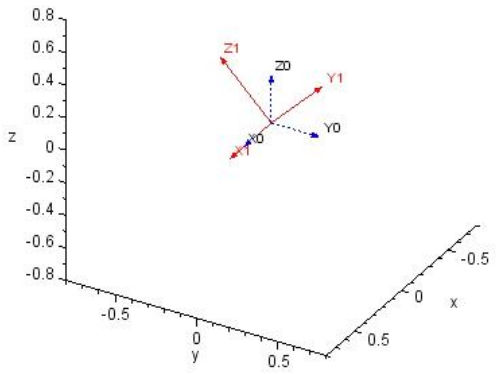


Figure 2.2 basic rotation in RTSX

give a plot like shown in Figure 2.2, where the option is used to plot world coordinate as reference frame. For `tranimate`, you can click-right on the graphical window and drag the frames to a better view angle, then replay the animation by pressing [y], or even change the animation speed by pressing [s].

Tips: to close the current Scilab graphic window, type `close`. Sometimes you have so many windows open and want to close them all at once, use `xdel(winsid());`

2.2 Composition of Rotations

In case a few rotations are performed in series, the whole rotation can be described by a product of rotation matrices. The order of multiplication is essential and dictated by whether a frame is rotated w.r.t the current frame, or the world frame.

2.2.1 Rotation about the Current Frame

The keyword here is "current frame," so we first explain what that is. Suppose 3 consecutive rotations are required. Starting from {0}, the world frame, we rotate it to create {1}. At this point, {1} is referred to as the current frame. So if we rotate {1} to create {2}, this rotation is done w.r.t the current frame. Now the current frame changes to {2}. If we rotate {2} to create {3}, this last rotation is w.r.t the current frame as well. Got it? In the next section we will discuss the case when some of the succeeding rotation might be done w.r.t {0}, or the world frame, which will give different result.

It can be easily shown that, when all rotations are performed w.r.t the current frame, the overall rotation can be computed from successive *post*-multiplication of each rotation matrix. For the 3 rotations described above, for example

$$ {}^{0}R_{3} = {}^{0}R_{1}\,{}^{1}R_{2}\,{}^{2}R_{3} \tag{2.8} $$

Ex. 2.1: rotation about current frame

Figure 2.3 shows 2 consecutive rotations, each about a main axis of the current frame. The process starts from rotating ϕ about X of {0}, and then rotating about Z of {1}. The resulting combination of rotation can be describes as

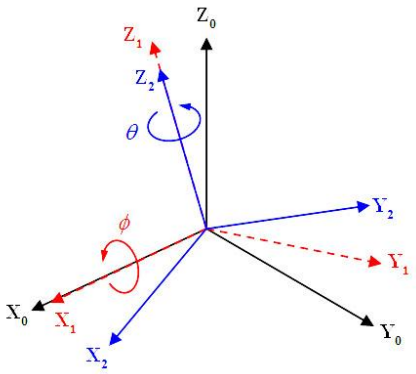


Figure 2.3 rotation about current frame

$$ R = R_{x,\phi}R_{z,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c\theta & -s\theta & 0 \\ c\phi s\theta & c\phi c\theta & -s\phi \\ s\phi s\theta & s\phi c\theta & c\phi \end{bmatrix} \tag{2.9} $$

Note: for brevity, we use only letter *c* and *s* for sin and cos, respectively.

To demonstrate the plot and animation of the rotation in Ex 2.1, type the following RTSX commands in Scilab console.

```
-->phi=pi/4; th=pi/3;
-->R1=rotx(phi); R2=rotz(th);
// plot {1} w.r.t {0}
-->trplot(R1,'world','figure',1);
// plot {2} w.r.t {1}
-->trplot(R1*R2,'hold','frame',2,
...
-->'color','c');
```

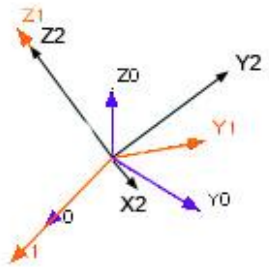


Figure 2.4 plot result from RTSX for rotation of current frame case.

This gives the plot shown in Figure 2.4. Comparison with Figure 2.3 confirms it is the desired operation. Note in the last command we use option `'hold'` to freeze the current frame {1} on the plot, `'frame',2` to set next frame number to 2, and `'color','c'` to set color to cyan.

To show animation, uses tranimate command. For first rotation,

```
-->tranimate(eye(3,3),R1,'figure',2,'holdstart');
```

Type [y] to replay the animation. The option `'holdstart'` is used to retain the start frame {0}. Then, for second rotation

```
-->tranimate(R1,R1*R2,'hold','color','b','frame',2);
```

Right-click and drag for better view angle and type [y] to replay.

2.2.2 Rotation about the World Frame

For a given series of rotation, the reference frame for the first operation is always the same. For any successive operation, however, it is possible that the rotation is done w.r.t the world frame, which in this case is the frame we start with, say, {0}. The multiplication sequence of rotation matrices now changes from the previous case. The formal analysis could be done using math tools like similarity transformation, though [SHV06] suggests a simple trick to handle such situation. This can be best shown by an example.

Ex 2.2: rotation about world frame

Figure 2.5 shows a series of 2 rotations. The first one is exactly the same as in Ex 2.1; i.e., rotating ϕ about X of {0}. It is the second rotation that is done at an angle θ about Z axis of the world frame {0}.

Since the first rotation is the same, we only need to find the right multiplication order of the second rotation. The trick is rotating {1} to coincide with {0} first, rotating about Z of both, and rotating {1} back. The result is

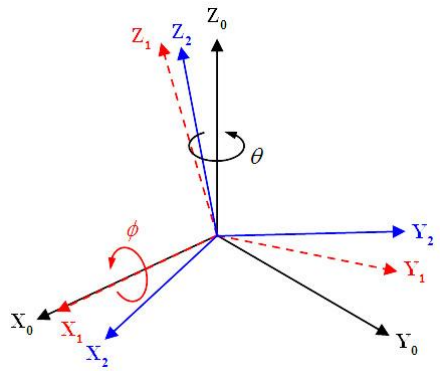


Figure 2.5 rotation about world frame

$$R = R_{x,\phi} R_{x,-\phi} R_{z,\theta} R_{x,\phi} = R_{z,\theta} R_{x,\phi} \qquad (2.10)$$

It can be concluded that that rotation about world frame at any point in the sequence results in pre-multiplication by the corresponding rotation matrix of that operation, in this case, $R_{z,\theta}$

An RTSX command designed specifically for rotation about world axis is `trotw`. This series of commands can be used to illustrate Ex 2.2, which gives the plot as in Figure 2.6

```
-->phi=pi/4; th=pi/6;
-->R1=rotx(phi);
-->R=trotw(R1,th,'z')
 R  =
    0.8660254  - 0.3535534    0.3535534
    0.5          0.6123724  - 0.6123724
    0.           0.7071068    0.7071068

// check that this equals (2.10)
-->R2 = rotz(th);
-->Rc = R2*R1
 Rc  =
    0.8660254  - 0.3535534    0.3535534
    0.5          0.6123724  - 0.6123724
    0.           0.7071068    0.7071068

// plot the first rotation
-->trplot(R1,'world','figure',1);

// plot the second rotation
-->trplot(R,'hold','frame',2,'color','k');

// animate the first rotation
-->tranimate(eye(3,3),R1,'figure',2,'holdstart');

// animate the second rotation
-->tranimate(R1,R,'hold','color','k','frame',2);
```

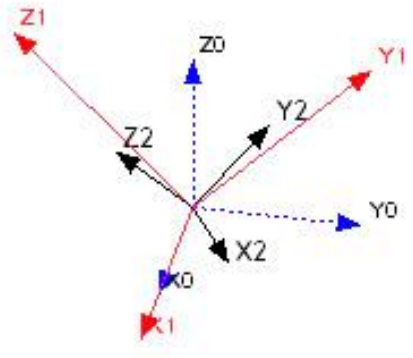


Figure 2.6 plot result from RTSX for rotation of world frame case

Observe how the second rotation differs from Ex. 2.1.

Tips: for a mixture of rotation w.r.t current and world frames, it can be shown that the overall rotation equals post and pre-multiplication by individual rotation matrix in the same order as the rotation operation takes place.

2.3 Representations for General Rotation

While basic rotation matrices defined above can be used to describe rotations about principle axes of a coordinate frame, they cannot be directly applied to a rotation in general case. One might guess that since a rotation matrix has size 3 x 3, nine parameters are needed to represent an arbitrary rotation. In fact, the rotational degree-of-freedom for an object in space is at most 3, so only three independent parameters are required.

For the general case, the common representations used are *Euler-angle*, *Roll-Pitch-Yaw* (RPY), and *axis/angle*. The derivations for each of these are too mathematically-involved for the main objective of this document. We will give only the key equations and focus on how to use RTSX commands for these parameteriztions.

2.3.1 Euler Angles

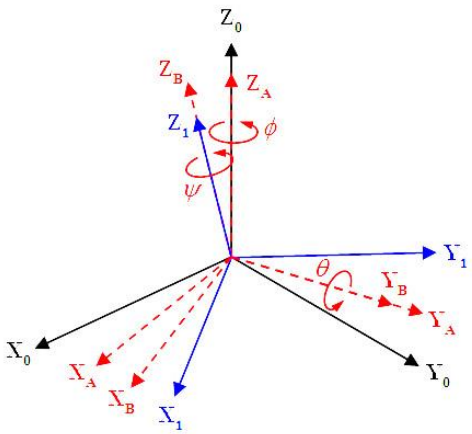


Figure 2.7 ZYZ Euler angle representation.

The first method to represent a general rotation in 3D is called *Euler angles*, which are simply derived from three successive basic rotations w.r.t current frames of particular order. There are variations of such order (total of twelve to choose from). Here we give an example of ZYZ sequence shown in Figure 2.7, commonly used in mechanical and aeronautical applications. The process starts by rotating ϕ about Z_0 to get {A}, then θ about Z_A to get {B}, and finishing with ψ about Z_B to get {1}. The resulting rotation matrix is then

$$
\begin{aligned}
R &= R_{z,\phi} R_{y,\theta} R_{z,\psi} \\
&= \begin{bmatrix} c_\phi c_\theta c_\psi - s_\phi s_\psi & -c_\phi c_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta \\ s_\phi c_\theta c_\psi + c_\phi s_\psi & -s_\phi c_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta \\ -s_\theta c_\psi & s_\theta s_\psi & c_\theta \end{bmatrix}
\end{aligned} \tag{2.11}
$$

The inverse problem is to find angles that satisfy (2.11), which is generally harder since there can be more than one solution.

ZYZ representation is the method used in RTSX. Define the Euler angles by $E = [\phi \quad \theta \quad \psi]$. To compute a rotation matrix for $E = [0.3 \quad 0.4 \quad 0.5]$, say, we can do it straight from the definition

```
-->R = rotz(0.3)*roty(0.4)*rotz(0.5)
 R  =
    0.6305253  - 0.6812010    0.3720256
    0.6968838    0.7078908    0.1150810
  - 0.3417467    0.1866971    0.9210610
```

or use the `eul2r` command

```
-->R = eul2r([0.3, 0.4, 0.5])
 R  =
    0.6305253  - 0.6812010    0.3720256
    0.6968838    0.7078908    0.1150810
  - 0.3417467    0.1866971    0.9210610
```

Note the syntax differnce from rvctools of MATLAB. Typing

```
-->R = eul2r(0.3, 0.4, 0.5);
```

in RSTX will cause an error.

which gives identical result with less typing. For the inverse problem,

```
-->E=tr2eul(R)
 E  =
    0.3    0.4    0.5
```

returns the original Euler angles. This is not the case when θ is negative

```
-->R=eul2r([0.3, -0.4, 0.5])
 R  =
    0.6305253  - 0.6812010  - 0.3720256
    0.6968838    0.7078908  - 0.1150810
    0.3417467  - 0.1866971    0.9210610
-->tr2eul(R)
 ans  =
  - 2.8415927    0.4  - 2.6415927
```

The inverse function returns a different set of Euler angles. In fact, it always returns a positive angle for θ. Of course, this set is also valid Euler angles for the given R

```
-->eul2r(ans)
 ans  =
    0.6305253  - 0.6812010  - 0.3720256
    0.6968838    0.7078908  - 0.1150810
    0.3417467  - 0.1866971    0.9210610
```

More interestingly, for the case $\theta = 0$

```
-->R=eul2r([0.3, 0, 0.5]);
-->tr2eul(R)
 ans  =
    0.    0.    0.8
```

Can you explain this situation?

Hint: let $\theta = 0$ in (2.11) and use a property of basic rotation matrix $R_{z,\phi}R_{z,\psi} = R_{z,\phi+\psi}$

2.3.2 RPY Angles

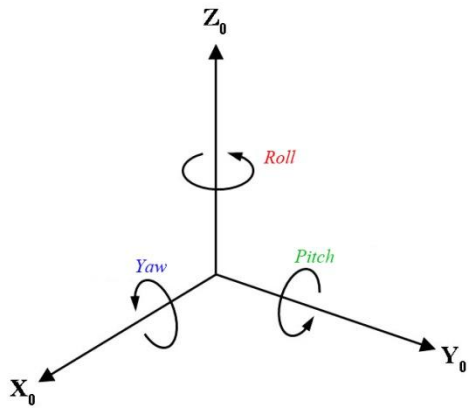


Figure 2.8 roll, pitch, and yaw angles

A rotation matrix can be described by a series of rotations about the principle coordinate axes X_0, Y_0, and Z_0 taken in some specific order. This is called a *Roll-Pitch-Yaw representation*, or RPY in short. There are some variations in the literature. The most common are XYZ order.

$$R = R_{x,\psi}R_{y,\theta}R_{z,\phi} \qquad (2.12)$$

shown in Figure 2.8, and the ZYX order

$$R = R_{z,\phi}R_{y,\theta}R_{x,\psi} \qquad (2.13)$$

which give different results. For the ZYX case, for example, the rotation matrix R can be computed as

$$R = \begin{bmatrix} c_\phi c_\theta & -s_\phi c_\psi + c_\phi s_\theta s_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ s_\phi c_\theta & c_\phi c_\psi + s_\phi s_\theta s_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix} \qquad (214)$$

An RTSX command to compute a rotation matrix is `rpy2r`, which is defaulted to XYZ order. An argument 'zyx' may be passed as an option

```
-->R=rpy2r([0.3, 0.4, 0.5],'zyx')
 R  =
    0.8799232  - 0.0809848    0.4681631
    0.2721921    0.8935594  - 0.3570196
  - 0.3894183    0.4415802    0.8083071
```

```
-->tr2rpy(R,'zyx')
 ans  =
    0.3     0.4     0.5
```

The RPY representation has singularity at pitch angle of $\pm$ 90 degrees.

2.3.3 Angle/Vector Representation

Instead of rotating about a principal axis, we may be interested in the case where rotation is performed about an arbitrary axis in space. Let $k = \begin{bmatrix} k_x & k_y & k_z \end{bmatrix}^T$ be a unit vector defining that axis, and θ the rotation angle. This situation is depicted in Figure 2.9. It can be shown that the resulting representation is [SHV06]

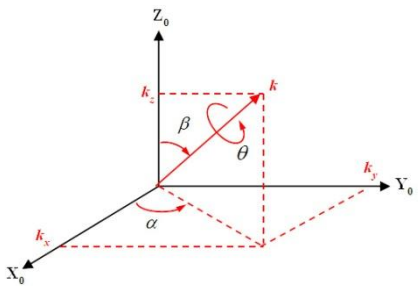


Figure 2.9 rotation about an arbitrary axis

$$R_{k,\theta} = \begin{bmatrix} k_x^2 v_\theta + c_\theta & k_x k_y v_\theta - k_z s_\theta & k_x k_z v_\theta + k_y s_\theta \\ k_x k_y v_\theta + k_z s_\theta & k_y^2 v_\theta + c_\theta & k_y k_z v_\theta - k_x s_\theta \\ k_x k_z v_\theta - k_y s_\theta & k_y k_z v_\theta + k_x s_\theta & k_z^2 v_\theta + c_\theta \end{bmatrix} \tag{2.15}$$

where $v_\theta = 1 - c_\theta$. The pair (k, θ) is called an *angle/vector representation* of a rotation. Let r_{ij} represent the $(i, j)^{\text{th}}$ element of $R_{k,\theta}$ in (2.15). The inverse problem can then be computed by directly solving for k and θ, which gives

$$\theta = \cos^{-1}\left(\frac{r_{11} + r_{22} + r_{33} - 1}{2}\right) \tag{2.16}$$

$$k = \frac{1}{2\sin\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \tag{2.17}$$

Use commands `angvec2r` and its inverse `tr2angvec` for angle/vector representation.

```
-->R = angvec2r(pi/4,[1 0 -1])
 R  =
    0.8535534    0.5          - 0.1464466
  - 0.5          0.7071068    - 0.5
  - 0.1464466    0.5            0.8535534
```

```
-->[th,k] = tr2angvec(R)
k  =
   0.7071068
   0.
 - 0.7071068
th  =
   0.7853982
```

Note that the return value for *k* is normalized.

2.4 Quaternions

A quaternion can be thought of as an extension of complex number. This math tool was discovered by W.R. Hamilton in 1843 and has found its use in 3D mechanics, computer graphics, and robotics. A quaternion is defined as follows

$$q = s + v_1 i + v_2 j + v_3 k \tag{2.18}$$

where $s \in \mathbb{R}$, $v \in \mathbb{R}^3$ and the orthogonal imaginery parts *i, j, k* such that

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.19}$$

In this book we write a quaternion in the form

$$q = s, < v_1, v_2, v_3 > \tag{2.20}$$

or, more briefly, $q = s, < v >$. To represent rotation in 3D we use unit quaternions, which are defined as quaternions with $|q| = 1$ or $s^2 + v_1^2 + v_2^2 + v_3^2 = 1$.

Function files for quaternion math operations in RTSX are included in subdirectory `/quaternion` , which are loaded with all other functions by `startup_rtsx.sci`. To create a quaternion that corresponds to an RPY rotation of 30, 45, and 60 degrees, type

```
-->q = Quaternion(rpy2r([pi/6, pi/4, pi/3]));
```

We can display it in string format by q2str command

```
-->q2str(q)
 ans  =
 0.723317, < 0.391904, 0.200562, 0.531976>
```

When a quaternion is multiplied by its inverse, the result is a quaternion identity 1,<0>

```
-->q2str(qmult(q, qinv(q)))
 ans  =
 1.000000, < 0.000000, 0.000000, 0.000000>
```

Plot and animation functions can accept a quaternion as argument. For instance, to show the rotation from this quaternion,

```
-->tranimate(q,'world');
```

Useful quaternion functions are listed below.

- `Quaternion` – create a quaternion
- `isquaternion, isqequal` – check if a variable is quaternion, or if two are equal
- `qadd, qsubtract` – add/subtract two quaternions
- `qmult, qdivide` – multiply/divide
- `qinv` – compute an inverse
- `qnorm` – compute quaternion norm
- `q2vec, q2str` – convert a quaternion to vector or string
- `q2tr, tr2q` – conversion between a quaternion and homogenous transformation/rotation matrix
- `qinterp` – quaternion interpolation

2.5 Homogeneous Transformation

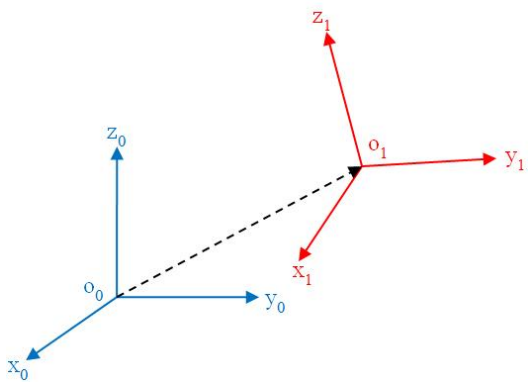


Figure 2.10 rotation and translation between two frames

So far we have discussed only rotation between two coordinate frames. In robot analysis, frames are attached to each joint of a robot so their origins do not coincide. Figure 2.10 shows a general relationship between two frames consisting of both rotation and translation. A convenient math tool called *homogeneous transformation* is used to describe such relationship. Rotational and translational data are packed into a 4x4 homogeneous matrix having the following form.

$$T = \left[\begin{array}{ccc|c} & R & & d \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.21}$$

where R is a 3 x 3 rotation matrix discussed earlier, and d a 3 x 1 vector representing the translation between two frames. As usual, we use pre-superscript and post-subscript to indicate the frames. For the frame {0} and {1}shown in Figure 2.10, for instance

$${}^0T_1 = \left[\begin{array}{ccc|c} & {}^0R_1 & & {}^0d_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.22}$$

describes the homogeneous transformation of {1} w.r.t {0}. It can be shown that its inverse can be computed as

$${}^1T_0 = {}^0T_1^{-1} = \left[\begin{array}{ccc|c} & {}^0R_1^T & & -{}^0R_1^T\,{}^0d_1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.23}$$

Homogeneous matrices are convenient in describing a chain of transformation, which can be computed as a product of successive frames in proper order.

$${}^0T_n = {}^0T_1\,{}^1T_2\,{}^2T_3 \ldots {}^{n-1}T_n \tag{2.24}$$

Basic homogeneous transformations consist of the following

$$Transl_{x,a} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Transl_{y,b} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Transl_{z,c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trot_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_\theta & -s_\theta & 0 \\ 0 & s_\theta & c_\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Trot_{y,\phi} = \begin{bmatrix} c_\phi & 0 & s_\phi & 0 \\ 0 & 1 & 0 & 0 \\ -s_\phi & 0 & c_\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Trot_{z,\psi} = \begin{bmatrix} c_\psi & -s_\psi & 0 & 0 \\ s_\psi & c_\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.25}$$

Here we use *Transl* to represent a translation along a principal axis, and *Trot* a rotation about a principal axis.

Suppose we want to construct {1} by shifting the origin of {0} to position $[-1, 1, 0.5]^T$, then rotating 45 degrees about X. The homogenous maritx can be computed as

$$T = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\pi/4} & -s_{\pi/4} & 0 \\ 0 & s_{\pi/4} & c_{\pi/4} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 0.707 & -0.707 & 1 \\ 0 & 0.707 & 0.707 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To verify this using RTSX, issue the following commands

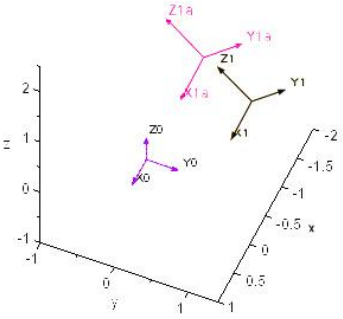
```
-->T=transl([-1,1,0.5])*trotx(45,'deg')
 T  =
    1.    0.          0.         - 1.
    0.    0.7071068   - 0.7071068    1.
    0.    0.7071068    0.7071068    0.5
    0.    0.           0.           1.
```

Note that the multiplication order is important.

```
-->Ta = trotx(45,'deg')*transl([-1,1,0.5])
 Ta  =
    1.    0.          0.         - 1.
    0.    0.7071068   - 0.7071068    0.3535534
    0.    0.7071068    0.7071068    1.0606602
    0.    0.           0.           1.
```

Use trplot to show that the resulting frames are different

```
-->trplot(T,'world','figure',1)
-->trplot(Ta,'figure',1,'hold',..
'frame','1a','color','m')
```



We can use tranimate command with a homogeneous maritx

```
-->tranimate(T,'world')
```

RTSX also includes a group of command to convert data to/from a homogeneous matrix, a rotation matrix, and vector *d* representing translation, such as `t2r`, `t2d`, `r2t`.

2.6 Transform Equations

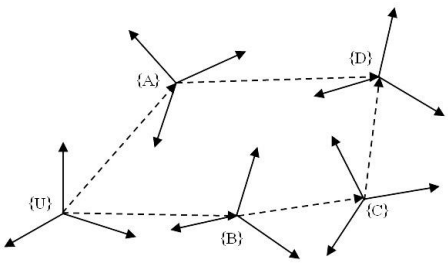


Figure 2.11 a closed-loop transformation chain

Kinematic study of a robot typically involves a series of coordinate frame transformation from its base up to the end-effector, which results in a product of each transformation as expressed by (2.24). There are situations when two transformation chains are formed into closed-loop like shown in Figure 2.1

In this case an unknown transformation in the loop can be computed as a function of other known homogeneous matrices.

As an illustration, suppose in Figure 2.11 we know every transformation except ${}^{B}T_{C}$, which we want to compute. From the upper link in Figure 2.11,

$$ {}^{U}T_{D} = {}^{U}T_{A}\,{}^{A}T_{D}\,. \tag{2.26} $$

And from the lower link

$$ {}^{U}T_{D} = {}^{U}T_{B}\,{}^{B}T_{C}\,{}^{C}T_{D} \tag{2.27} $$

After equating the two equations and some algebra (details are left as an exercise), we eventually have

$$ {}^{B}T_{C} = {}^{B}T_{U}\,{}^{U}T_{A}\,{}^{A}T_{D}\,{}^{D}T_{C} \tag{2.28} $$

The next example shows how to solve this kind of problem using software.

Ex. 2.3 (SHV06, problem 2-39) In Figure 2.12, coordinate frames are attached to the base of robot arm, table corner, workpiece, and camera with positions and orientations as shown. The workpiece is a cube object with size 20x20x20 cm., and {2} is attached at the cube center. The camera is mounted directly on top ot the cube at height 2 meters. We want to find homogeneous transformations of each frame w.r.t robot base {0}, and camera {3}.

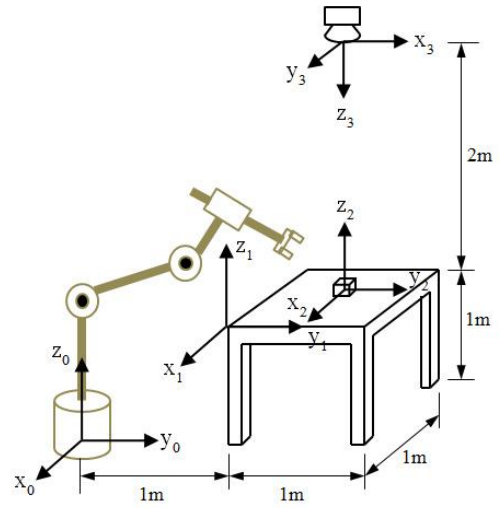


Figure 2.12 robot diagram for Ex 2.3

From the given frame positions and orientations, we see that ${}^{0}T_{1}$ and ${}^{0}T_{2}$ can be found easily as

$$ {}^{0}T_{1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {}^{0}T_{2} = \begin{bmatrix} 1 & 0 & 0 & -0.5 \\ 0 & 1 & 0 & 1.5 \\ 0 & 0 & 1 & 1.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

Since {0}, {1}, {2} have the same orientation, only the translation parts need to be determined.

To compute ${}^0T_3$ is more involved. First we must find the orientation of {3} w.r.t {0}. A direct way is to find a combination of rotation from {0} to {3}. There are more than one way to do so. A solution is ${}^0R_3 = R_{x,\pi}R_{z,-\pi/2}$. Then, the translation part is measured directly from the figure as ${}^0d_3 = \begin{bmatrix} -0.5 & 1.5 & 3 \end{bmatrix}^T$. Hence,

```
-->T30=transl([-0.5,1.5,3])*trotx(pi)*trotz(pi/2)
 T30  =
    0.    - 1.      0.    - 0.5
  - 1.      0.      0.      1.5
    0.      0.    - 1.      3.
    0.      0.      0.      1.
```

We can use similar method to compute other non-trivial transformations such as ${}^3T_2$. Alternatively, there is a set of RTSX commands designed to solve for an unknown frame in closed-loop transformation chain. In this case, we already compute ${}^0T_1, {}^0T_2, {}^0T_3$ and now want to find ${}^2T_3$ which is an inverse of ${}^3T_2$. The first step in the process is to create a definition of transformation chain with `Frame` and `SerialFrame` commands

```
-->T10 = [1 0 0 0; 0 1 0 1; 0 0 1 1;0 0 0 1];
-->T20 = [1 0 0 -0.5;0 1 0 1.5;0 0 1 1.1; 0 0 0 1];
-->T30 = [0 1 0 -0.5;1 0 0 1.5;0 0 -1 3;0 0 0 1];
-->F(1) = Frame(eye(4,4),'name','0');
-->F(2) = Frame(T10,'name','1');
-->F(3) = Frame(T20,'abs','name','2');
-->F(4) = Frame([],'name','3');  //unknown frame
```

The chain construction in data structure `F` always starts from the base frame upwards. Arguments to the command Frame are the homogeneous matrix, frame name, and an optional flag `'rel'` (default) indicating that the homogeneous matrix is described w.r.t the frame below; i.e., ${}^1T_2, {}^2T_3, {}^3T_4, \cdots, {}^{n-1}T_n$, or `'abs'` for the case the homogenous matrix is w.r.t base; i.e., ${}^0T_2, {}^0T_3, \cdots, {}^0T_n$. For an unknown frame, put a blank square bracket `[]` in place of the homogeneous matrix. After finishing this frame creation step, we pass the data structure `F` to `SerialFrame` and name it `Chain 1`.

```
-->fc1 = SerialFrame(F,'name','Chain 1');
Reading frame data and computing missing information
Processing Upwards ...
```

```
1 -- {0} : Found T_rel. Fill in T_abs with T_rel
2 -- {1}: Found T_rel. Computing T_abs :{1} w.r.t {0}-- Finished
3 -- {2}: Found T_abs. Computing T_rel :{2} w.r.t {1}-- Finished
4 -- {3}: *** Missing both T_abs: {3} w.r.t {0} and T_rel: {3} w.r.t {2}
***
List of missing frames in Chain 1
================================
4 -- {3}: T_abs : {3} w.r.t {0} , T_rel : {3} w.r.t {2}
```

Observe from the output messages how `SerialFrame` attempts to compute and fill in the missing transformation in the chain using the given data, until it reaches {3}, the unknown frame. At that point the algorithm does not have enough information to continue. From the discussion relating to Figure 2.11 above, we know this problem can be solved only when another transformation chain is available to from a closed-loop. For this problem, this another chain is simply ${}^0T_3$.

The command `ResolveFrame` computes the unknown transformation in a closed-loop chain. `PlotResolveFrame` does the same thing but also plots the frame chain diagram, so we will use the latter. Simply input `fc1` and ${}^0T_3$ to the function

```
-->T32 = PlotResolveFrame(fc1,T30);
Reading frame data and computing missing information
Processing Upwards ...
1 -- {1} : Found T_rel. Fill in T_abs with T_rel
2 -- {2}: Found T_rel. Computing T_abs :{2} w.r.t {1}-- Finished
Chain 2: All missing information are computed and filled in
Running ResolveFrame ...
4 -- {3} (Missing Frame): fill in T_abs : {3} w.r.t {0} -- Finished
4 -- {3} (Missing Frame): Computing T_rel : {3} w.r.t {2} -- Finished
Rechecking if there are still missing frame data in Chain 1-- none found.
--- Chain 1 is now completed. ---
 Missing data T: {3} w.r.t {2} in Chain 1 is computed as
    0.    1.    0.    0.
    1.    0.    0.    0.
    0.    0.  - 1.    1.9
    0.    0.    0.    1.
```

Then we can find as an inverse of

```
-->T23 = inv(T32)
```

```
T23 =
   0.    1.    0.    0.
   1.    0.    0.    0.
   0.    0.  - 1.    1.9
   0.    0.    0.    1.
```

which, for this problem, they are equal. Figure 2.13 shows the frame diagram from `PlotResolveFrame` command. The missing transformation is displayed as dotted line.

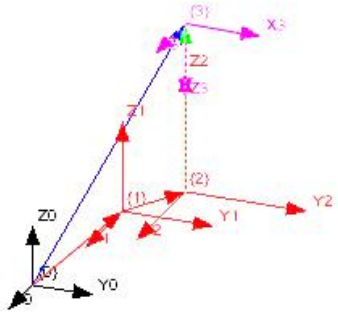


Figure 2.13 diagram from `PlotResolveFrame`

Tips: There are other commands in this group that facilitates frame creation and management such as `InsertFrame, DeleteFrame, ReplaceFrame`, that do what the command name says, while `PlotFrame` can be used to show any frame diagram.

Problems

2-1 Given coordinate frame {1},{2},{3} and rotation matrices

$$ {}^1R_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & -0.866 \\ 0 & 0.866 & 0.5 \end{bmatrix}, \quad {}^1R_3 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, $$

compute matrix $ {}^2R_3 $. Verify using RTSX.

2-2 Given $k = \frac{1}{\sqrt{3}}[1 \quad 1 \quad 1]^T, \theta = \frac{\pi}{2}$, compute $R_{k,\theta}$ by hand and by RTSX.

2-3 Find the rotation matrix corresponding to the Euler angles $\left(\frac{\pi}{2}, 0, \frac{\pi}{4}\right)$. What is the direction of the axis z_1 w.r.t the base frame?

2-4 From Ex. 2.3, suppose the block on the table is rotated 90° about z_2 and moved so that its center has coordinate $[0 \quad 0.8 \quad 0.1]^T$ relative to the frame {1}. Compute the homogeneous transformation relating the block frame to the camera frame; the block frame to the base frame.

2-5 Refering to Ex. 2.3, attach a new frame {4} to the robot gripper. Compute the homogeneous transformation of {4} w.r.t the base and camera frame, when the gripper moves to position $[-1 \quad 1.5 \quad 1.2]^T$ and has orientation $R_{x,\pi/2}$ w.r.t {1}.