Lecture 06.04 Coordinate frame transformations

position
orientation

Robotics requires we keep track of the positions and orientations of many objects in three-space. A rigid body's state can be expressed as a three-dimensional *position* and *orientation* (angular position). In a coordinate system, this takes a minimum of $3 + 3 = 6$ coordinates.

body-fixed

Different objects have different convenient coordinate systems. For instance, a mobile robot might have a *body-fixed coordinate system* with origin at its geometric centroid, x-axis pointing forward, y-axis pointing leftward, and z-axis pointing upward. Locating an object in this coordinate system would be different than that of, say, a base station. Consider for a mobile robot a two-dimensional body-fixed coordinate system o, world coordinate system w, and a pseudo body-fixed coordinate system p that is merely a translation of the world coordinate system to the p origin—see Figure 06.5. Let a point in space in $w/p/o$-coordinates is represented by the position vector $\mathbf{r}_w/\mathbf{r}_p/\mathbf{r}_o$. Let $\mathbf{t}$, be a vector from the w-origin to the p-origin.

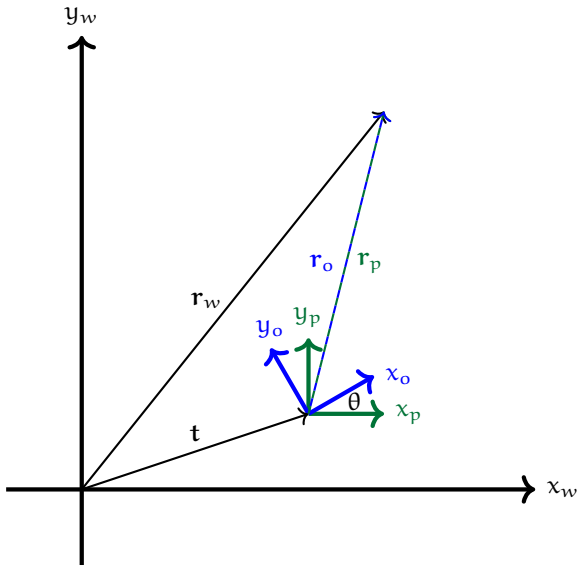

Figure 06.5: a two-dimensional body-fixed coordinate system o, world coordinate system w, and a pseudo body-fixed coordinate system p.

06.04.1 Translation

Suppose the robot can only translate and not rotate. The w and p coordinate transformations are sufficient to describe its motion. The transformation is

$$\mathbf{r}_w = \mathbf{r}_\mathrm{p} + \mathbf{t} \tag{06.1a}$$
$$\mathbf{r}_\mathrm{p} = \mathbf{r}_w - \mathbf{t}. \tag{06.1b}$$

As we will see in a moment, rotation of a vector is described by a matrix operation on a vector. It is therefore convenient to write translation as a matrix operation in one extra dimension:

$$\mathbf{r}_w = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{r}_\mathrm{p} \tag{06.2a}$$
$$= \underbrace{\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathsf{T}} \begin{bmatrix} r_x^\mathrm{p} \\ r_y^\mathrm{p} \\ 1 \end{bmatrix} \tag{06.2b}$$
$$= \begin{bmatrix} r_x^\mathrm{p} + t_x \\ r_y^\mathrm{p} + t_y \\ 1 \end{bmatrix}. \tag{06.2c}$$

The last component, then, becomes an accounting tool for writing the translation operation in this form—called a *homogeneous representation* (Bullo and Lewis, 2005). The transformation matrix T translates but does not rotate.

homogeneous representation

Exercise 06.1

Show that $\mathbf{r}_\mathrm{p} = \mathsf{T}^{-1}\mathbf{r}_w$ by showing it to be equivalent to Equation 06.1b.

06.04.2 Rigid body transformation

Transformation to and from a body-fixed coordinate system is usually a *rigid body transformation*: one that changes coordinate frame origin position and orientation, but preserves the Euclidean distance between any two points. Transformations between the w and o coordinate systems, above, are rigid body transformations. These could be represented as a *rotation matrix* R transformation followed by a translation by $\mathbf{t}$:

rigid body transformation

rotation matrix

$$\mathbf{r}_w = \mathsf{R}\mathbf{r}_\mathrm{o} + \mathbf{t}. \tag{06.3}$$

Here, R rotates counter-clockwise by θ with matrix

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{06.4}$$

However, we frequently like to write this in a homogeneous representation, as well, again adding a component to the vectors such that R becomes

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{06.5}$$

and the rigid body transformation becomes

$$\mathbf{r}_w = TR\mathbf{r}_o. \tag{06.6}$$

06.04.3 Rotation transformations

Rotation transformations, such as R above, come in a variety of flavors.

Euler angles These rotations are described by the sequential rotation about a (typically) body-fixed coordinate system. The order matters because rotating about one axis changes the direction of the others! Not one, but several conventions exist for Euler angle rotation.

Fixed angles Similarly, rotations can be described about axes the origin of which remains fixed to the body, but the orientation of which remains *fixed to the world frame*.

Axis-angles Axis-angle representations describe a rotation as a unit vector and an angle of rotation about that vector.

Quaternions Quaterions are complex numbers with a real part and *three* (instead of the usual one) imaginary parts. They can describe rotations in a manner that avoids certain problems (e.g. gimbal lock and ill-conditioned quatities) of other representations and is more computationally efficient.

The non-quaternion rotation transformations use matrix multiplication and can therefore have homogeneous forms that include translation. Quaternions cannot represent translations, so vector-addition must supplement (multiplicative) quaternion transformations.

06.04.4 The ROS package `tf2`

At this point, some things should be clear:

1. for a three-dimensional robot with six degrees of freedom, keeping track of even two coordinate systems (e.g. world and body-fixed) can be complicated;
2. adding more coorinate systems for arms, sensors, moving objects in the environment, etc.—as most real robots require—vastly complicates coordinate transformations;
3. coordinate transformations change with time as body-fixed coordinate systems move; and finally
4. keeping track of all this in an *ad hoc* way would be disastrous, so a systematic approach is required.

For these reason, ROS provides just such a systematic approach via its *`tf2`* package.[5,6] **tf2**

The `tf2` package has conventions for coordinate transformation data, organized into a tree structure and *buffered in time*. Time-buffering is important: frequently, we need not just the latest data, but recent data as well. As with all ROS dataflow, `tf2` communicates via publishing and subscribing to topics. **time-buffering**

ROS `tf2` uses quaternions to apply and store rotation information. However, it is usually easier for us to think in terms of Euler angles. The older `tf` package provides a nice conversion from Euler angles to quaternions:

```
from tf.transformations import quaternion_from_euler
q = quaternion_from_euler(ax,ay,az) # usage
```

In the usage example, above, rotation angles ("`a`") are applied sequentially to body-fixed `x`, `y`, and `z` axes.

06.04.5 Try out `tf2`

In a Terminal window, enter the following to get and compile a turtle `tf2` demo.

[5]The `tf2` package documentation can be found here:
`wiki.ros.org/tf2`.
The `tf2_ros` package provides Python bindings:
`wiki.ros.org/tf2_ros`.

[6]The `tf2` package replaces the older `tf` package. For information about migrating, see
`wiki.ros.org/tf2/Migration`.

```
sudo apt-get install \
  ros-$ROS_DISTRO-turtle-tf2 \
  ros-$ROS_DISTRO-tf2-tools \
  ros-$ROS_DISTRO-tf
```

Now launch the demo with the following command.

```
roslaunch turtle_tf2 turtle_tf2_demo.launch
```

A separate screen should load with two turtles. Select the Terminal window and use the arrow keys to direct one of the turtles about. The other turtle will follow, as shown in Figure 06.6.

For the full demo, see

`wiki.ros.org/tf2/Tutorials/Introduction to tf2`.

Figure 06.6: a turtle-follow-turtle graphic using `tf2`.