2. Coordinate Frames

2.1 Introduction

A key requirement in robotics programming is keeping track of the positions and velocities of objects in space. For example, consider the situation in Figure 2.1. This robot has been programmed to find the blue teapot and report its position to a remote user. The robot's vision system has detected the teapot directly in front of the camera at a distance of 1 meter.

In this situation it would not be very helpful to report to the user that the teapot has been located `ONE METER IN FRONT OF MY CAMERA`. Presumably, the user wants to know where *in the room* the teapot is located. Providing this information requires us to know each of the following:

1. The position of the teapot relative to the camera.
2. The position and orientation of the camera relative to the base of the robot.

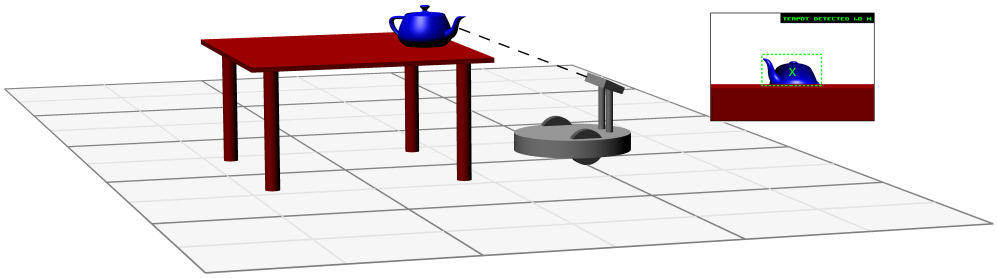


Figure 2.1: This robot has located a teapot and must report the location to a user. The inset image shows the view from the robot's camera.

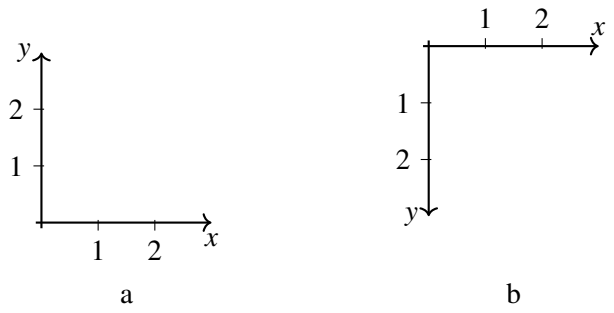


Figure 2.2: There are several ways we can draw two-dimensional coordinate systems. a. The origin is drawn in the lower-left corner. b. the origin is drawn in the upper-left.

3. The position of the robot in the room.

Determining items 1 and 3 may be challenging problems on their own, but for now, we will assume that we have access to this information. The goal in this chapter is to develop a framework for representing these relative locations and using that information to translate coordinates from one frame of reference to another. In this case, from a frame of reference defined by the position of the camera (`THE TEAPOT IS ONE METER IN FRONT OF MY CAMERA`), to a fixed frame of reference defined relative to the room (`THE TEAPOT IS NEAR THE SOUTH-WEST CORNER OF THE ROOM`).

2.2 Conventions

As a first step we need to establish some conventions for describing locations and orientations in three dimensions.

You certainly have experience working with two-dimensional coordinate systems like the one illustrated in Figure 2.2a. This figure follows the common convention of placing the origin is at the lower-left with the positive x-axis draw horizontally and the positive y-axis drawn vertically.

It is possible to draw this coordinate system differently. For example, the convention when working with coordinates on a computer screen is to place the origin at the upper-left corner as shown in Figure 2.2b.

In a sense, Figures 2.2a and 2.2b are the same coordinate system drawn in two different ways. You can imagine picking up the axes in Figure 2.2a, flipping them over and placing them back on top of the axes in Figure 2.2b so that the axes are aligned.

The situation is different in three dimensions. Depending on how the axes are arranged we can end up with one of two fundamentally incompatible coordinate systems as illustrated in Figure 2.3. The coordinate system on the left is referred to as a **left-handed coordinate system**, while the one on the right is a **right-handed coordinate system**. In a right-handed coordinate system we determine the direction of the z-axis by aiming the pointer finger of the right hand along the positive x-axis and curling our palm toward the positive y-axis. The thumb will then point in the direction of positive z. For a left-handed coordinate system we follow the same procedure using the left hand.

There is no way to rotate these two coordinate systems so that they align. They represent two incompatible ways of representing three-dimensional coordinates. This means that whenever we

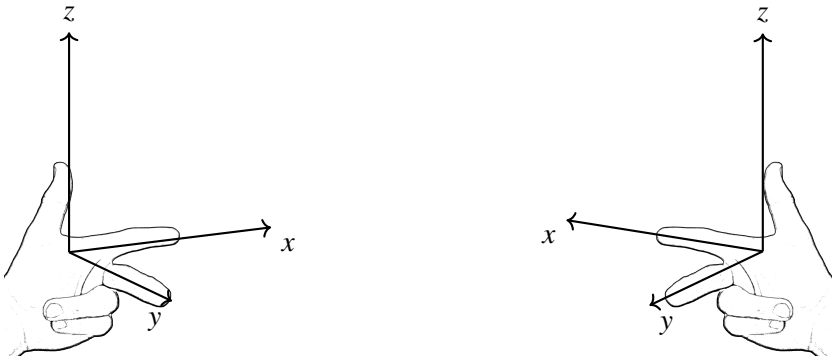


Figure 2.3: Left and right-handed coordinate systems.

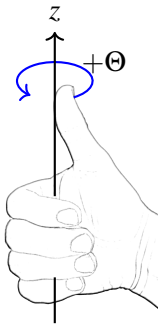


Figure 2.4: Right-hand rule for determining the direction of positive rotations around an axis.

provide coordinates in three dimensions we must specify whether we are using a left-handed or right-handed system. There is no universal convention for which should be used, but right-handed systems are more common in robotics. All of the examples in this book will assume right-handed coordinate systems.

We also need to establish a convention for describing the direction of a rotation. Again, we will follow a right-hand rule. To determine the direction of positive rotation around a particular axis we point the thumb of the right hand along the axis in question. The fingers will then curl around the axis in the direction of positive rotation. This is illustrated in Figure 2.4.

2.3 Poses and Coordinate Frames

The position and orientation of an object in space is referred to as its **pose**. Any description of an object's pose must always be made in relation to some to some **coordinate frame**. It isn't helpful to know that my teapot is at position (-2, 2.2, 1.6) unless I know where (0, 0, 0) is and which directions the three axes are pointing.

In robotics, it is often convenient to keep track of multiple coordinate frames. Figure 2.5 illustrates three potentially useful coordinate frames related to the teapot scenario described above. The "world" coordinate frame is indicated with the subscript w. This coordinate frame is fixed at a known location in space and assumed not to move over time. The "robot" coordinate frame is indicated with the subscript r. We can envision this coordinate frame as being attached to the base of the robot so that the origin of

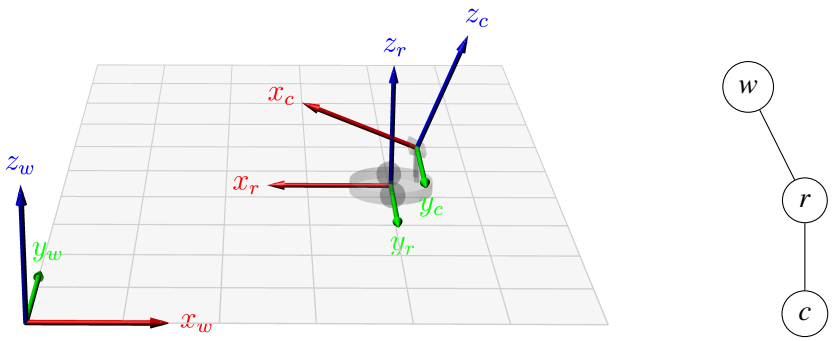


Figure 2.5: Left: three coordinate frames from the teapot scenario illustrated in Figure 2.1. Right: tree representing the relationship between the three coordinate frames.

the frame moves as the robot moves. In other words, the robot is always located at (0, 0, 0) in its own coordinate frame. Similarly, the "camera" coordinate frame is attached to the camera.

This figure follows the common convention of drawing the x-axis in red, the y-axis in green and the z-axis in blue. In the case of robots (and many other objects) we have a natural notion of directions corresponding to "forward", "left" and "up". Throughout this book these will correspond to positive x, positive y and positive z respectively.

We can organize these three coordinate frames into a tree structure with the "world" coordinate frame as the root. Each edge in the tree represents the pose of a particular coordinate frame relative to its parent. Ideally, this tree structure should mirror the physical connections between the objects involved. In this case it makes sense to describe the pose of the camera in the robot's coordinate frame, because the two are physically attached and the camera is constrained to move along with the robot. When the robot moves, it is only necessary to update the pose of the robot coordinate frame. The camera remains stationary relative to the robot.

Assuming a connected tree, it will be possible to translate a point between any two coordinate frames. For example, given the coordinates of the teapot in the camera coordinate frame, we can determine its coordinates in the robot coordinate frame, and then in the room coordinate frame. In order to accomplish this we need to specify how poses are represented. We also need a mechanism for converting from parent to child coordinate frames and vise-versa.

2.3.1 Translations

Recall that the pose of an object (or a coordinate frame) includes both its position and orientation relative to the parent coordinate frame. Representing the position is straightforward. Three numbers may be used to represent a translation of the object along each of the three axes of the parent coordinate frame.

Figure 2.6 shows two coordinate frames separated by a simple translation. In this example, the child coordinate frame is located at position (1.5, 1.0, .5) in the parent coordinate frame.

We will use subscripts to indicate the coordinate frame associated with a point or pose. For example, $(0,0,.5)_c$, refers to the point $(0,0,.5)$ in the child coordinate frame.

When coordinate frames are separated by a pure translation transforming points between frames is

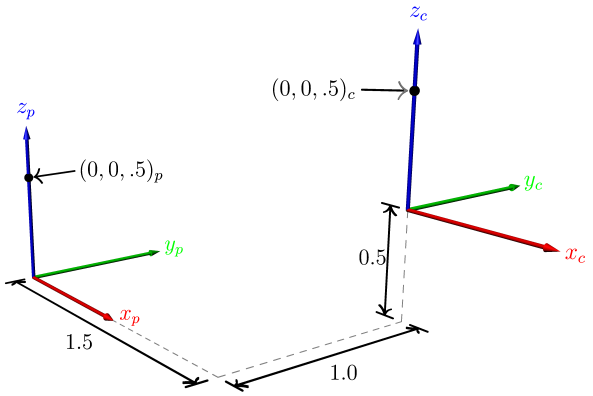


Figure 2.6: Two coordinate frames separated by a simple translation. The parent frame is labeled p, the child frame is labeled c.

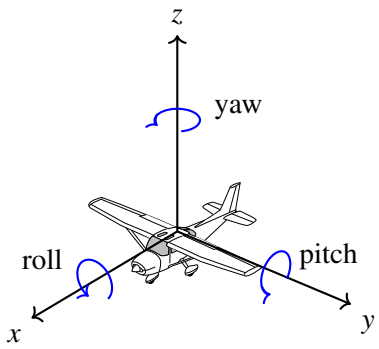


Figure 2.7: Euler angles.

straightforward: we only need to add or subtract the three coordinate offsets. For example, the point $(0,0,.5)_c$ is located at $(0+1.5,0+1.0,.5+.5) = (1.5,1.0,.5)$ in the parent coordinate frame. Moving from the parent to the child requies us to subtract instead of add: $(0,0,.5)_p$ is located at $(-1.5,-1.0,0)$ in the child frame.

2.3.2 Euler Angles

Specifying orientation in three dimensions is more complicated than specifying position. There are several approaches, each with its own strengths and weaknesses. **Euler angles** are probably the most intuitive representation. Figure 2.7 illustrates the basic idea. Orientation is expressed as a sequence of three rotations around the three coordinate axes. These three rotations are traditionally referred to as **roll**, **pitch** and **yaw**.

When working with Euler angles it is necessary to specify the order that the rotations will be applied: a 90^o rotation around the x-axis followed by a 90^o rotation around the y-axis does *not* result in the same orientaion as the same rotations applied in the opposite order. There are, in fact, *twelve* valid rotation orderings: xyz, yzx, zxy, xzy, zyx, yxz, zxz, xyx, yzy, zyz, xzx, and yxy.

It is also necessary to specify whether the rotations are relative to the parent coordinate frame, or whether each rotation is performed around the axes of a coordinate frame aligned with the earlier

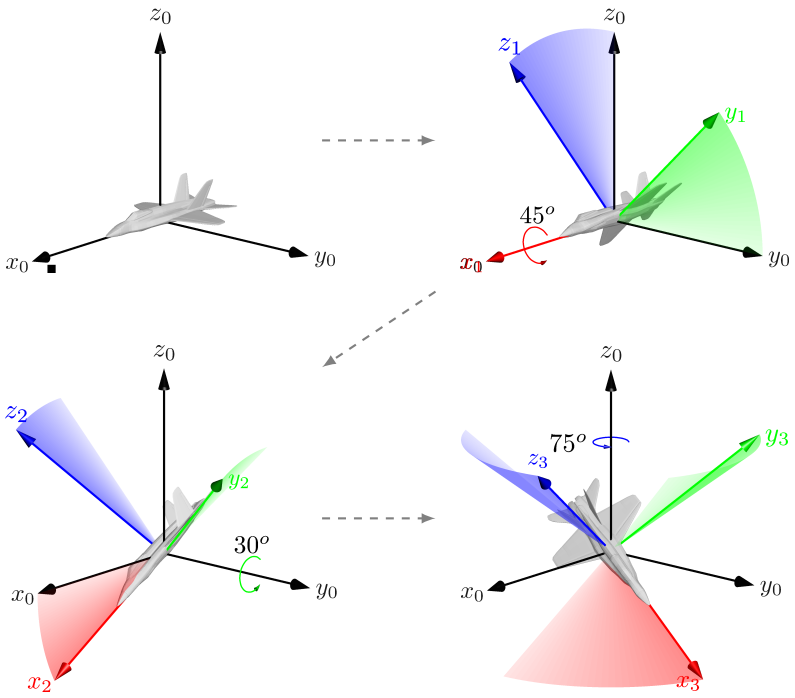


Figure 2.8: Static

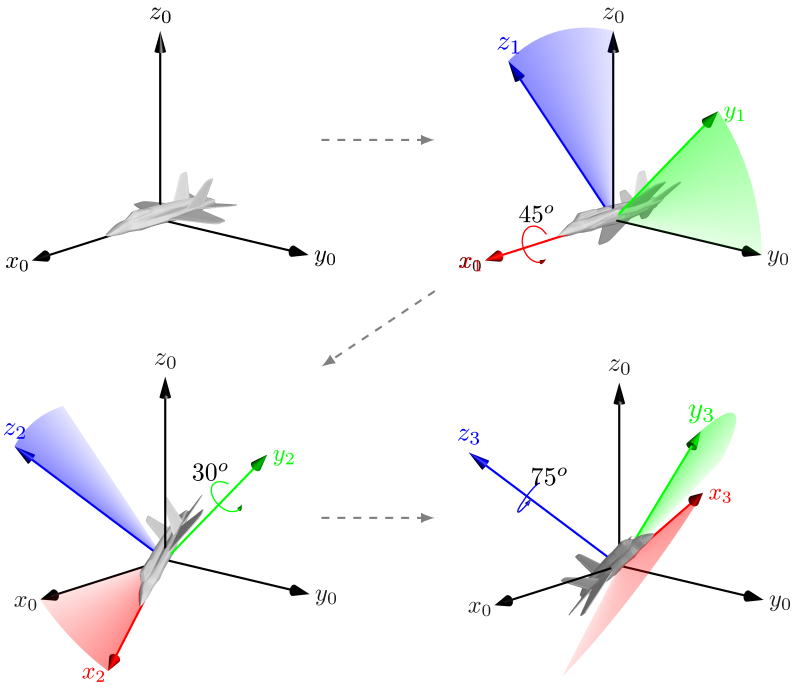


Figure 2.9: Non static

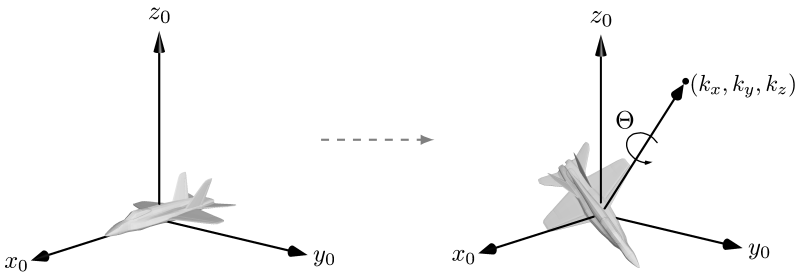


Figure 2.10: Axis-angle representation of orientation.

rotations. These two alternatives are illustrated in Figures 2.8 and 2.9. The first alternative is sometimes referred to as "static" or "extrinsic" rotations, while the second may be referred to as "relative" or "intrinsic" rotations. Combined with the choice of axis orderings, this means that there are twenty-four possible conventions for specifying Euler angles.

Euler angles are relatively easy to visualize, but they are inconvenient to work with from a mathematical point of view. The key problem is that the mapping from spatial orientations to Euler angles is discontinuous: small changes in orientation may cause big jumps in the required representation. This can cause difficulties when we need to smoothly update the orientation of a moving object over time.

2.3.3 Axis Angle

Euler angles encode an orienation using three rotations around pre-defined axes. An alternate approach is to encode an orientation as a *single* rotation Θ around an arbitrary unit vector $[k_x, k_y, k_z]^T$. This is referred to as the **axis-angle** representation. The idea is illustrated in Figure 2.10.

2.3.4 Quaternions

Probably the most widely used method for encoding orientation is the **quaternion**. There is a close relationship quaternions and the axis-angle representation described above. If Θ and $[k_x, k_y, k_z]^T$ describe the axis-angle representation of an orientation, then the quaternion represetation is a four-tuple (x, y, z, w) such that:

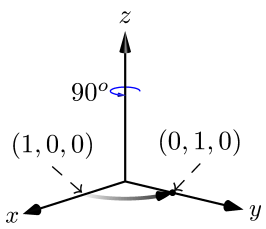
$$x = k_x \sin\frac{\Theta}{2} \tag{2.1}$$

$$y = k_y \sin\frac{\Theta}{2} \tag{2.2}$$

$$z = k_z \sin\frac{\Theta}{2} \tag{2.3}$$

$$w = \cos\frac{\Theta}{2} \tag{2.4}$$

Notice that the x, y and z components point in the same direction as the original axis of rotation. A quaternion constructed according to Equations 2.1-2.4 is also referred to as a unit-quaternion because it has the property that $\sqrt{x^2 + y^2 + z^2 + w^2} = 1$.



$$R_z(90^o) \times \begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix}\cos 90^o & -\sin 90^o & 0\\ \sin 90^o & \cos 90^o & 0\\ 0 & 0 & 1\end{bmatrix} \times \begin{bmatrix}1\\0\\0\end{bmatrix}$$
$$= \begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix} \times \begin{bmatrix}1\\0\\0\end{bmatrix}$$
$$= \begin{bmatrix}0\\1\\0\end{bmatrix}$$

Figure 2.11: Example rotation

Quaternions are *not* particularly intuitive or easy to interpret. Their popularity is based on the fact that they support efficient computation and they avoid the troublesome discontinuities associated with Euler angles.

2.3.5 Rotation Matrices

Until now we have represented spatial coordinates as ordered triples: (x,y,z). Equivalently, we may think of points in space as three-dimensional column vectors: $\begin{bmatrix}x & y & z\end{bmatrix}^T$. This view is convenient because certain spatial transformations may be accomplished through matrix operations. In particular, any rotation can be encoded as a 3×3 **rotation matrix**. Pre-multiplying the matrix by a point will have the effect of performing the desired rotation around the origin.

The following three matrices correspond to rotations around the x, y, z axes respectively:

$$R_x(\Theta) = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\Theta & -\sin\Theta\\ 0 & \sin\Theta & \cos\Theta\end{bmatrix} \tag{2.5}$$

$$R_y(\Theta) = \begin{bmatrix}\cos\Theta & 0 & \sin\Theta\\ 0 & 1 & 0\\ -\sin\Theta & 0 & \cos\Theta\end{bmatrix} \tag{2.6}$$

$$R_z(\Theta) = \begin{bmatrix}\cos\Theta & -\sin\Theta & 0\\ \sin\Theta & \cos\Theta & 0\\ 0 & 0 & 1\end{bmatrix} \tag{2.7}$$

Figure 2.11 shows the example of rotating the point $\begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$ by 90^o around the z-axis.

It is possible to represent any orienation as a product of three rotation matrices around the x, y and z axes. It is straightforward to convert from an Euler angle representation to the corresponding rotation

matrix. For static rotations, the three elementary rotation matrices must be multiplied in the order that the rotations should be applied. This means that the rotation matrix that corresponds to the Euler angle rotations illustrated in Figure 2.8 would be:

$$R_{static} = R_x(45^o) \times R_y(30^o) \times R_z(75^o)$$

For Euler angles represented using relative rotations, the order is reversed. The rotation matrix corresponding to Figure 2.9 would be:

$$R_{relative} = R_z(75^o) \times R_y(30^o) \times R_x(45^o)$$

The main advantage of rotation matrices is that they provide a convenient mechasism for composing rotations and for applying rotations to a point. The downside is that this is a highly redundant represenatation. A 3×3 rotation matrix contains 9 values, but as we saw in Section 2.3.2 three numbers are sufficient to represent any orientation in a three dimensional space.

2.4 References and Further Reading

Jennifer Kay provides an excellent and very accessible tutorial on kinematics and coordinate transforms in (Kay, 2020). That tutorial also describes **homogeneous coordinates** which provide a convenient mechanism for combining rotations and translations into a single matrix product.

A widely used open-source software library for managing coordinate frames and performing coordinate transforms is tf/tf2 (http://wiki.ros.org/tf2). The tf2 library tracks time-stamped information about the relative positions and orientations of a tree of coordinate frames and supports efficient transforms between any two frames. Introducing a time dimension is valuable because robots move over time and must interact with dynamic environments. The design and implementation of the tf library is described in (Foote, 2013).