Survival Guide for Coordinate Transforms

Wyatt Newman
February, 2015

Defining coordinate frames: A "frame" is defined by a point in 3-D space, **p**, that is the frame's origin, and three vectors: **n**, **t** and **b** (which define the local x, y and z axes, respectively). The axis vectors are normalized (have unit length), and they form a right-hand triad, such that **n** crossed into **t** equals **b:**

$\boldsymbol{b}=\boldsymbol{n}\times\boldsymbol{t}$ These three directional axes can be stacked side-by-side as column vectors, comprising a 3x3 matrix, **R**.

$$\boldsymbol{R}=[\boldsymbol{n}\,\boldsymbol{t}\,\boldsymbol{b}]=\begin{bmatrix} n_x & t_x & b_x \\ n_y & t_y & b_y \\ n_z & t_z & b_z \end{bmatrix}$$

We can include the origin vector as well to define a 3x4 matrix as:

$$\boldsymbol{A}=[\boldsymbol{n}\,\boldsymbol{t}\,\boldsymbol{b}\,\boldsymbol{p}]=\begin{bmatrix} n_x & t_x & b_x & p_x \\ n_y & t_y & b_y & p_y \\ n_z & t_z & b_z & p_z \end{bmatrix}$$

A useful trick to simplify mathematical operations is to define an augmented matrix, converting the above 3x4 matrix in to a square 4x4 matrix by adding a fourth row consisting of [0 0 0 1]. This augmented matrix is a 4x4, which we will refer to as a "**T**" matrix:

$$\boldsymbol{T}=\begin{bmatrix} n_x & t_x & b_x & p_x \\ n_y & t_y & b_y & p_y \\ n_z & t_z & b_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Conveniently, matrices constructed as above (consistent with valid frame specifications) are always invertible. Further, computation of the inverse of a **T** matrix is efficient.

Abstractly, one can refer to an origin and a set of orientation axes (vectors) without having to specify numerical values. However, to perform computations, numerical values are required—and this requires further definitions. Specifically, when numerical values are given, one must define the coordinate system in which the values are measured. For example, to specify the origin of frame "B" (i.e. point "**p**") with respect to frame "A", we can measure p_x along the x axis of frame "A", measure p_y along the y axis of frame "A", and p_z along the z axis of frame A. These can be referred to explicitly as $p_{x/A}$, $p_{y/A}$ and $p_{z/A}$, respectively. If we had provided coordinates for the origin of frame B from any other viewpoint, the numerical values of the components of **p** would be different.

Similarly, we can describe the components of frame-B's **n** axis (similarly, **t** and **b** axes) with respect to frame A by measuring the x, y and z components along the respective axes of frame A. We can then state the position and orientation of frame B with respect to frame A as:

$$ {}^{A}\boldsymbol{T}_{B} = \boldsymbol{T}_{B/A} = \begin{bmatrix} n_{x/A} & t_{x/A} & b_{x/A} & p_{x/A} \\ n_{y/A} & t_{y/A} & b_{y/A} & p_{y/A} \\ n_{z/A} & t_{z/A} & b_{z/A} & p_{z/A} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

We can refer to the above matrix as “frame B with respect to frame A.”

Having labelled frames “A” and “B”, providing values for the elements of ${}^{A}\boldsymbol{T}_{B}$ fully specifies the position and orientation of frame B with respect to frame A.

Coordinate-frame transformations: In addition to providing a means to explicitly declare the position and orientation of a frame (with respect to some named frame, e.g. frame B with respect to frame A), **T** matrices can also be interpreted as operators. For example, if we know the position and orientation of frame C with respect to frame B, i.e. ${}^{B}\boldsymbol{T}_{C}$, and if we also know the position and orientation of frame B with respect to frame A, ${}^{A}\boldsymbol{T}_{B}$, then we can compute the position and orientation of frame C with respect to frame A as follows:

$$ {}^{A}\boldsymbol{T}_{c} = {}^{A}\boldsymbol{T}_{B}\,{}^{B}\boldsymbol{T}_{C} $$

That is, a simple matrix multiplication yields the desired transform. This process can be extended, e.g.:

$$ {}^{A}\boldsymbol{T}_{F} = {}^{A}\boldsymbol{T}_{B}\,{}^{B}\boldsymbol{T}_{C}\,{}^{C}\boldsymbol{T}_{D}\,{}^{D}\boldsymbol{T}_{E}\,{}^{E}\boldsymbol{T}_{F} $$

In the above, the 4x4 on the left-hand side can be interpreted column by column. For example, the fourth column (rows 1 through 3), are the coordinates of the origin of frame “F” as measured with respect to frame “A.”

By using the notation of prefix superscripts and post subscripts, there is a visual mnemonic to aid logical compatibility. The super and sub-scripts act like “Lego” blocks, such that a subscript of a leading **T** matrix must match the pre-superscript of the trailing **T** matrix. Following this convention help to keep transform operations consistent.

Coordinate transforms for navigation on a plane: To define “where” our robot is with respect to the world, we need to define a “robot” frame and a “map” frame. We define the robot frame as a reference frame on (attached to) our mobile robot, defined to have an origin between the drive wheels, projected down to floor level. The x axis of the robot frame is forward, the y axis is to the left, and the z axis is straight up.

We will also define a “map” frame. Given a map, we define an origin and x and y axes on the map (independent of the robot). We can specify where our robot is in the map frame by specifying (x,y) coordinates of the robot's origin (as measured from the map origin, along the map x and y axes) and robot heading (a rotation about z of the robot's x axis relative to the map's x axis). Although only 3 values are required (x, y and heading), we can use the more general (6-D) pose specification by filling in values of: ${}^{map}\boldsymbol{T}_{robot}$.

For **T** matrices restricted to motion on a plane, 10 of the 16 components are always the same:

$$ {}^{map}\boldsymbol{T}_{robot} = \begin{bmatrix} n_{x/map} & t_{x/map} & 0 & p_{x/map} \\ n_{y/map} & t_{y/map} & 0 & p_{y/map} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

In fact, the above representation contains redundancy. E.g., given $n_{x/map}$ and $n_{y/map}$ we can deduce the heading as $\psi = atan2(n_{x/map}, n_{y/map})$. The values of $t_{x/map}$ and $t_{y/map}$ follow from $n_{x/map}$ and $n_{y/map}$ as a rotation of 90 deg about the z axis. Although the **T**-matrix representation contains redundant information, it is nonetheless convenient to use, since it supports use of linear algebra for computing coordinate transformations.

In addition to expressing ${}^{map}\boldsymbol{T}_{robot}$, it is also useful to refer to an "odometry" frame. When the robot is started up, it arbitrarily defines its odometry coordinates to be (x,y) = (0,0) with a heading of 0. In this case, the robot's origin (a point between the left and right wheels) defines the odometry-frame origin, and the heading of the robot defines the x axis of the odometry frame. Equivalently, at start-up (by definition), the T-matrix ${}^{odom}\boldsymbol{T}_{robot}$ is simply the identity:

$$ {}^{odom}\boldsymbol{T}_{robot} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

As the robot moves, the robot frame and the odom frame will no longer be aligned, and the corresponding T matrix will have values of:

$$ {}^{odom}\boldsymbol{T}_{robot} = \begin{bmatrix} n_{x/odom} & t_{x/odom} & 0 & p_{x/odom} \\ n_{y/odom} & t_{y/odom} & 0 & p_{y/odom} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The odom frame is related to the map frame through the T-matrix ${}^{map}\boldsymbol{T}_{odom}$ (whether we know the values for this or not).

There exists a transform ${}^{map}\boldsymbol{T}_{odom}$ corresponding to the odometry frame expressed in the map frame. If we knew the values of the components of ${}^{map}\boldsymbol{T}_{odom}$, we could convert odometry coordinates to map coordinates through the following operation:
${}^{map}\boldsymbol{T}_{robot} = {}^{map}\boldsymbol{T}_{odom}\,{}^{odom}\boldsymbol{T}_{robot}$.

Odometry coordinates get updated by keeping track of incremental wheel motions. With the additional information of start-up pose, i.e. ${}^{map}\boldsymbol{T}_{odom}$, we can keep track of where the robot is in the map.

Correcting for odometry drift: A problem with depending on odometry to get map coordinates is that odometry is subject to drift, e.g. due to wheel slip. This drift effect is relatively slow (barring dramatic jolts of the robot), but it accumulates. A virtue of the odometry values is that they are updated rapidly and smoothly (avoiding jumps or discontinuities). These properties make the odometry coordinates

suitable for feedback (e.g. for steering). Alternatively, ${}^{map}\boldsymbol{T}_{robot}$ can be estimated by comparing LIDAR measurements to a pre-defined map, seeking a best fit. Such estimates do not suffer from drift. However, these computations are typically slower, resulting in discontinuous jumps. As such, these values are not suitable for feedback in steering.

To combine the best properties of odometry vs map fitting, we integrate this through a somewhat counter-intuitive trick: to allow for a time-varying transform ${}^{odom}\boldsymbol{T}_{robot}$. Instead of considering ${}^{odom}\boldsymbol{T}_{robot}$ to be flawed, we can interpret this value (the odom output, as published by the robot from wheel motions) to be perfect—but that the origin of this frame is variable. Equivalently, we can say ${}^{odom}\boldsymbol{T}_{robot}$ is perfect but ${}^{map}\boldsymbol{T}_{odom}$ drifts slowly. Interpreted visually, if we were to display the odom frame on the map, the odom frame would drift (and rotate) slowly as the robot moves and accumulates odometry error.

As new estimates of ${}^{map}\boldsymbol{T}_{robot}$ are updated (e.g. based on LIDAR values with respect to the map), the transform ${}^{map}\boldsymbol{T}_{odom}$ may be updated, computed as:

$${}^{map}\boldsymbol{T}_{odom} = {}^{map}\boldsymbol{T}_{robot}\,{}^{odom}\boldsymbol{T}_{robot}^{-1}$$

where ${}^{odom}\boldsymbol{T}_{robot}^{-1} = {}^{robot}\boldsymbol{T}_{odom}$. That is, the odom frame can be expressed with respect to the robot frame by computing the matrix inverse of the robot frame with respect to the odom frame.

Thus, given ${}^{odom}\boldsymbol{T}_{robot}$ (which is updated rapidly and smoothly) and ${}^{map}\boldsymbol{T}_{robot}$ (which is updated relatively infrequently, possibly with discrete jumps), we can compute the updates of the hypothetically mobile origin of the odom frame, ${}^{map}\boldsymbol{T}_{odom}$.

Steering control in the odom frame: Since the robot pose in odom coordinates is preferred for feedback, we must correspondingly express the desired trajectory in odom coordinates. For planning purposes, a desired path is computed with respect to a map (thus assuring trajectories that avoid collisions with objects in the map). For feedback, the desired state (a pose as a function of time along a pre-defined path) and the measured state (the current pose of the robot, as reported by odometry) are compared to derive a steering angle. For feedback control, the desired and measured states must be expressed in the same frame.

For example, we may desire to go from frame "A" to frame "B", expressed in the map frame as ${}^{map}\boldsymbol{T}_{A}$ and ${}^{map}\boldsymbol{T}_{B}$. With a relatively recent update of ${}^{map}\boldsymbol{T}_{odom}$, these can be converted to odom coordinates through the transform:

${}^{odom}\boldsymbol{T}_{A} = {}^{odom}\boldsymbol{T}_{map}\,{}^{map}\boldsymbol{T}_{A}$, and similarly: ${}^{odom}\boldsymbol{T}_{B} = {}^{odom}\boldsymbol{T}_{map}\,{}^{map}\boldsymbol{T}_{B}$.

A path-interpolator node can then compute incremental updates of desired pose, ${}^{odom}\boldsymbol{T}_{des}$, that advance smoothly from ${}^{odom}\boldsymbol{T}_{A}$ to ${}^{odom}\boldsymbol{T}_{B}$. In this manner, both the desired state and the measured state are updated frequently and smoothly.

It should be noted, however, that conversion from a plan in map coordinates to odom coordinates should be done incrementally (converting each subsequent subgoal as the prior subgoal is reached). If all coordinates are converted in advance, these conversions will become invalid as ${}^{map}\boldsymbol{T}_{odom}$ evolves en route.

Transforms in ROS: Coordinate-frame transformations are ubiquitous in robotics. Coordinate transforms for navigation, as introduced above, are more complex in higher dimensions (e.g. for aerial drones or submarines), but the transforms introduced handle these higher-dimensional cases as well. For articulated robot arms, full 6-DOF transforms are required to compute gripper poses as a function of joint angles. Multiplication of sequential transforms, link by link, performs such computations. Sensor data, e.g. from cameras, or LIDARs, is acquired in terms of the sensor's own frame, and this data must be interpreted in terms of alternative frames (e.g. world frame or robot frame).

Since coordinate transforms are so common in robotics, ROS has provided a powerful package—the "tf" package (see http://wiki.ros.org/tf)-- for handling transforms. Use of this package can be confusing, because "tf" performs considerable work.

Within ROS, the topic "/tf" carries messages of type tf2_msgs/TFMessage. There can be (and typically are) many publishers to this topic. Each publisher expresses a transform relationship, describing a named "child" frame with respect to a named "parent" frame. The ROS transform datatype is not identical to a 4x4 homogeneous transformation matrix, but it carries equivalent information (and more). The ROS transform datatype contains a 3-D vector (equivalent to the 4th column of a 4x4 transform) and a quaternion (an alternative representation of orientation). In addition, transform messages have time stamps—as well as a history of prior transforms.

User-written ROS nodes can publish transforms to the tf topic, announcing important spatial relationships. For example, a sensor can publish how it is mounted to a robot—whether this spatial relationship is fixed (bolted to the frame) or variable (e.g., a rotating or oscillating LIDAR).

Computation of localization relative to a map can post its results by publishing ${}^{map}\boldsymbol{T}_{robot}$ to the tf topic (using the corresponding tf message type). Similarly, an odometry node can publish its pose as an equivalent ${}^{odom}\boldsymbol{T}_{robot}$ message on the tf topic. A robot arm can publish successive link transforms on the tf topic. When sensor data is published, it should use a header that contains the name of the sensor's frame. This information, together with tf messages, allows one to transform the sensor data to any frame of interest (e.g., world frame or robot frame).

A user node can instantiate a "transformListener" object (from the tf library), which is very useful for computing coordinate transforms. The transformListener subscribes to the tf topic, receives messages relating child and parent frames (with random arrival times) and assembles this information into a connected chain (using the most recent available incremental transforms). By maintaining this chain, the transformListener is capable of responding to inquiries regarding coordinate-frame transforms (e.g., where is my right index fingertip with respect to my nose).

Further details of the ROS tf package are deferred for now, but its use will be prevalent in navigation, arm kinematics, and sensor-data transforms.