ECE 495/595 Lecture Slides

Winter 2017

Instructor: Micho Radovnikovich

Summary and Quick Links

These slides contain the following concepts:

Transforming Between Coordinate Frames

- Transforming points between different coordinate frames is an important concept applied in robotics.
  - Relating the coordinates of the end effector of a robotic arm to the individual joint angles.
  - Sensor data processing.
  - Building a map and navigating in an obstacle-rich environment.

Transforming Between Coordinate Frames

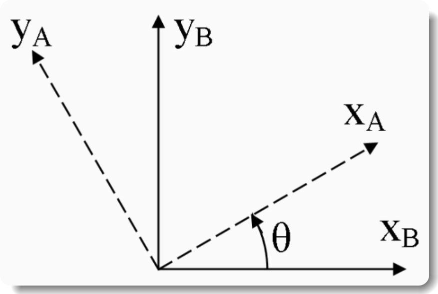
- A complete coordinate transformation involves both a translation and an orientation component.

- Translations are represented by vectors that are added to produce an offset.

- Orientations are much more complicated, and involve the use of *rotation matrices* or *quaternions* to represent them.

Transforming Between Coordinate Frames

- Transform a point $(x_A,\ y_A)$ into $(x_B,\ y_B)$ by projecting the x_A and y_A axes onto the x_B and y_B axes.



x_B Projection

$$x_B = x_A \cos\theta - y_A \sin\theta$$

y_B Projection

$$y_B = x_A \sin\theta + y_A \cos\theta$$

- z_B axis doesn't change:

z_B Projection

$$z_B = z_A$$

Transforming Between Coordinate Frames

- These three projection equations can be combined into a single matrix equation:

Rotation Matrix Equation

$$\underbrace{\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix}}_{P_B} = \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_B^A} \underbrace{\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}}_{P_A}$$

- A rotation matrix describes the orientation of one reference frame relative to another.

Rotation Matrix Notation

$$R_B^A$$

- The subscript $\underline{B}$ indicates that B is the *parent frame*, and the superscript $\underline{A}$ indicates that A is the *child frame*.

- In English, this notation is interpreted as the *rotation from frame B to frame A*, since it represents how the B frame can be manipulated to align with the A frame.

- However, muliplying a vector in A frame by this rotation matrix will transform it into B frame coordinates, which seems opposite to the English interpretation.

- **It is important to remember this convention, otherwise things can be very confusing!**

Rotation Matrix Notation

- One benefit of this method of notating rotation matrices is that the subscripts and superscripts behave vaguely algebraically:

Matrix Notation

$$P_B = R_B^{\cancel{A}} P_{\cancel{A}}$$

- The subscripts A 'cancel' out, leaving the result in the B frame.

Properties of Rotation Matrices

- Rotation matrices are *orthogonal*, meaning all rows and columns are orthogonal unit vectors.

- Because of the orthogonality, the inverse of a rotation matrix is simply its transpose:

$$\boxed{R^{-1} = R^T}$$

Properties of Rotation Matrices

- To reverse the process and transform a point $(x_B,\ y_B)$ into $(x_A,\ y_A)$, simply multiply both sides by the inverse of the rotation matrix:

$$\underbrace{\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1}}_{(R_B^A)^{-1}} \underbrace{\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix}}_{P_B} = \underbrace{\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}}_{P_A}$$

- Since the inverse of a rotation matrix is its transpose:

Reverse Transformation

$$\underbrace{\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_A^B} \underbrace{\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix}}_{P_B} = \underbrace{\begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}}_{P_A}$$

Properties of Rotation Matrices

- The determinant of a rotation matrix must be equal to $+1$
- Otherwise, the multiplication would end up changing the vector's magnitude.

$$\begin{aligned}
&\det\left(\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\right)\\
&= [(\cos\theta\cdot\cos\theta\cdot 1) + 0 + 0]\\
&\qquad - [0 + (-\sin\theta\cdot\sin\theta\cdot 1) + 0]\\
&= \cos^2\theta + \sin^2\theta = 1
\end{aligned}$$

Roll-Pitch-Yaw

- Any arbitrary orientation can be represented by a combination of three single-axis rotations.
- The three independent rotation angles are known as *Euler Angles*.
- In the Roll-Pitch-Yaw (RPY) convention, the three sub-rotation matrices are:

Roll (x)

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

Pitch (y)

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Yaw (z)

$$R_z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Roll-Pitch-Yaw

- ▷ In terms of the RPY matrices, transforming a point from frame A to frame B is given by

RPY Transformation

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = \underbrace{R_z R_y R_x}_{R_B^A} \begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix}$$

- ▷ The rotation matrix from B to A is constructed by multiplying the individual roll, pitch, and yaw matrices.

- ▷ Keep in mind that the order of multiplication matters. Changing the order will change the result!

Translation and Rotation

- With a nonzero translation, transforming a point in A frame into B frame is performed by:

$$P_B = R_B^A P_A + T_B^A$$

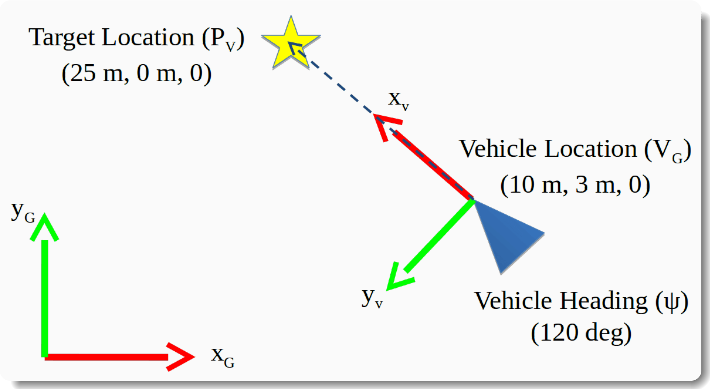
- The inverse transform can be derived by rearranging the above equation:

$$P_A = (R_B^A)^{-1} \left(P_B - T_B^A\right) = R_A^B P_B - R_A^B T_B^A$$

- Notice that the translation vector isn't simply negated in the inverse transform, but is also rotated by the inverse rotation matrix!

Example #1

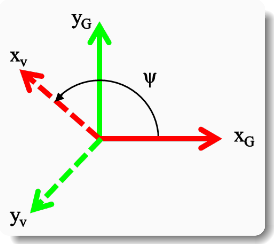
- Consider a vehicle traveling in a GPS reference frame at a heading angle of 120 degrees (north-northwest).
- Compute the global coordinates of a point 25 meters in front of the vehicle.



Example #1

- First, compute the description of the transform from global to vehicle frame:

$$T_G^V = \begin{bmatrix} 10 \\ 3 \\ 0 \end{bmatrix}$$



$$R_G^V = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} -0.5 & -0.866 & 0 \\ 0.866 & -0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\psi=120 \text{ deg}}$$

Example #1

- Since the target point is in vehicle frame, the corresponding point in global frame is obtained by applying the forward transformation:

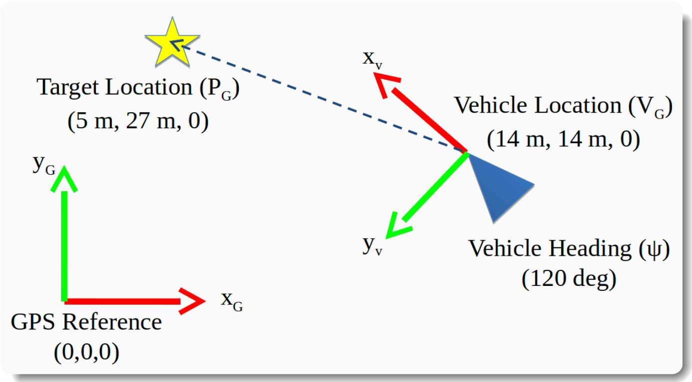
$$P_G = R_G^V P_V + T_G^V$$

$$P_G = \underbrace{\begin{bmatrix} -0.5 & -0.866 & 0 \\ 0.866 & -0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_G^V} \underbrace{\begin{bmatrix} 25 \\ 0 \\ 0 \end{bmatrix}}_{P_V} + \underbrace{\begin{bmatrix} 10 \\ 3 \\ 0 \end{bmatrix}}_{T_G^V} = \begin{bmatrix} -2.5 \\ 24.65 \\ 0 \end{bmatrix}$$

Example #2

- Consider a vehicle traveling in a GPS reference frame at a heading angle of 120 degrees (north-northwest).

- Compute the target location's coordinates in the vehicle's reference frame.



Example #2

- In this case, the rotation from global to vehicle is the same, but the translation vector is:

$$T_G^V = \begin{bmatrix} 14 \\ 14 \\ 0 \end{bmatrix}$$

$$R_G^V = \begin{bmatrix} -0.5 & -0.866 & 0 \\ 0.866 & -0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example #2

- This time, the target position is in global frame, so the corresponding point in vehicle frame is obtained using the inverse transformation instead:

$$P_V = R_V^G P_G - R_V^G T_G^V$$

$$P_G = \underbrace{\begin{bmatrix} -0.5 & 0.866 & 0 \\ -0.866 & -0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R_V^G = (R_G^V)^T} \underbrace{\begin{bmatrix} 5 \\ 27 \\ 0 \end{bmatrix}}_{P_G} - \underbrace{\begin{bmatrix} -2.4 \\ -10.16 \\ 0 \end{bmatrix}}_{R_V^G T_G^V} = \begin{bmatrix} 15.75 \\ 1.29 \\ 0 \end{bmatrix}$$

Eigen C++ Library

- The *Eigen* matrix library is the closest thing to MATLAB that exists in C++.
- Eigen is a highly versatile and optimized, and is used widely in demanding applications.
- Syntax is relatively clean and understandable, but things can go badly if not set up correctly.
- Full documentation of Eigen can be found at `http://eigen.tuxfamily.org/index.php?title=Main_Page`

Eigen C++ Library

- The base matrix object is declared like this:

```
//Eigen::Matrix<type, rows, cols> name;
Eigen::Matrix<double, 3, 3> example_mat;
```

- A matrix of dynamic size is declared with *Eigen::Dynamic*:

```
Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic> example_mat;
```

- This dynamic matrix declaration is identical to using the *Eigen::MatrixXd* "typedef"ed object:

```
Eigen::MatrixXd example_mat;
```

Eigen C++ Library

- There are many other objects "typedef"ed from the base *Matrix* object:
  - *Eigen::VectorXd* – A dynamic matrix with a fixed width of 1 column.
  - *Eigen::Matrix4d* – A fixed-sized matrix of size 4×4.
- The letter at the end of the "typedef"ed objects indicates the data type of the matrix elements:
  - *Matrix3d* – 'd' means double (64-bit floating point)
  - *Matrix3f* – 'f' means float (32-bit floating point)
  - *Matrix3i* – 'i' means int (32-bit integer)

Eigen C++ Library

- Matrices can be filled with arbitrary data like this:

```
Eigen::Matrix3f example_mat;
example_mat << 1, 2, 3, 4, 5, 6, 7, 8, 9;
```

- They can be initialized to special types of matrices:

```
example_mat.setZero(3, 3);
example_mat.setIdentity(3, 3);
```

- MATLAB syntax is used to access individual elements (remember zero-based indexing though!):

```
example_mat(0, 1) = 4;
```

Eigen C++ Library

- MATLAB syntax for matrix arithmetic:

```
Eigen::Matrix3d A;
Eigen::Vector3d b;
Eigen::Vector3d c;
Eigen::Vector3d d;

d = A * b + c;
```

- Computed properties can be used on the fly:

```
Eigen::Matrix3d A;
Eigen::Vector3d b;
Eigen::Vector3d c;
Eigen::Vector3d d;

d = A.transpose() * b + c;
```