ECE 495/595 Lecture Slides

Winter 2017

Instructor: Micho Radovnikovich

Summary and Quick Links

These slides contain the following concepts:

Quaternion Fundamentals

- A quaternion is a 4-dimensional number:

Quaternion

$$q = w + x\hat{\imath} + y\hat{\jmath} + z\hat{k}$$

- Contains a real component (w), and three independent imaginary components (x, y and z).
- The imaginary values $\hat{\imath}$, $\hat{\jmath}$ and $\hat{k}$ are all equal to $\sqrt{-1}$, but on different axes!
- Many more details about quaternions can be found on the Internet. Wikipedia has a good article about them: http://en.wikipedia.org/wiki/Quaternion

Quaternion Fundamentals

- The imaginary components of a quaternion multiply together in a vector cross product fashion to yield the following identities:

Multiplication Identities

$$
\begin{array}{ccc}
\hat{\imath}\hat{\jmath} = \hat{k} & \hat{\jmath}\hat{k} = \hat{\imath} & \hat{k}\hat{\imath} = \hat{\jmath} \\
\hat{\jmath}\hat{\imath} = -\hat{k} & \hat{k}\hat{\jmath} = -\hat{\imath} & \hat{\imath}\hat{k} = -\hat{\jmath}
\end{array}
$$

Quaternion Fundamentals

- Multiplying two quaternions yields another quaternion.

Multiplying Quaternions

$$q_3 = \underbrace{(w_1 + x_1\hat{\imath})}_{q_1}\underbrace{(w_2 + y_2\hat{\jmath})}_{q_2}$$

$$q_3 = w_1 w_2 + x_1 w_2 \hat{\imath} + w_1 y_2 \hat{\jmath} + x_1 y_2 \underbrace{(\hat{\imath}\hat{\jmath})}$$

$$q_3 = \underbrace{w_1 w_2}_{w_3} + \underbrace{x_1 w_2}_{x_3}\,\hat{\imath} + \underbrace{w_1 y_2}_{y_3}\,\hat{\jmath} + \underbrace{x_1 y_2}_{z_3}\,\hat{k}$$

- The multiplication identities are used to resolve the multiplication of different imaginary components.

Quaternion Fundamentals

- It is helpful to represent a quaternion multiplication as a matrix equation:

Quaternion Matrix Multiplication

$$\begin{bmatrix} w_3 \\ x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} w_1 & -x_1 & -y_1 & -z_1 \\ x_1 & w_1 & -z_1 & y_1 \\ y_1 & z_1 & w_1 & -x_1 \\ z_1 & -y_1 & x_1 & w_1 \end{bmatrix} \begin{bmatrix} w_2 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

- Using this notation, the previous example would look like this:

$$\begin{bmatrix} w_3 \\ x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} w_1 & -x_1 & 0 & 0 \\ x_1 & w_1 & 0 & 0 \\ 0 & 0 & w_1 & -x_1 \\ 0 & 0 & x_1 & w_1 \end{bmatrix} \begin{bmatrix} w_2 \\ 0 \\ y_2 \\ 0 \end{bmatrix}$$

Quaternion Fundamentals

- The complex conjugate of a quaternion is defined by:

Complex Conjugate

$$q^* = w - x\hat{\imath} - y\hat{\jmath} - z\hat{k}$$

- Like any other vector, a quaternion's magnitude is:

Magnitude

$$||q|| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

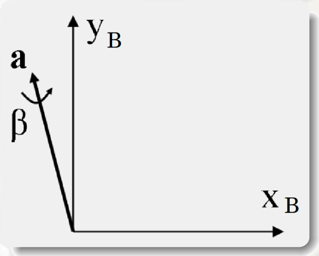
- The **inverse** of a quaternion is computed using the complex conjugate and the magnitude:

Quaternion Inverse

$$q^{-1} = \frac{q^*}{||q||}$$

Quaternions Representing Rotation

- A rotation from frame B to frame A can be described by rotating B abount a unit-length vector $\mathbf{a}$ by an angle β.



- Mathematically, this action is represented as a quaternion by:

Rotation Quaternion

$$q_B^A = \cos\left(\frac{\beta}{2}\right) + \sin\left(\frac{\beta}{2}\right)\mathbf{a}$$

Quaternions Representing Rotation

- The x, y and z components of the vector $\mathbf{a}$ are the $\hat{\imath}$, $\hat{\jmath}$ and $\hat{k}$ components of the quaternion:

$$q_B^A = \underbrace{\cos\left(\frac{\beta}{2}\right)}_{w} + \underbrace{a_x \sin\left(\frac{\beta}{2}\right)}_{x} \hat{\imath} + \underbrace{a_y \sin\left(\frac{\beta}{2}\right)}_{y} \hat{\jmath} + \underbrace{a_z \sin\left(\frac{\beta}{2}\right)}_{z} \hat{k}$$

- Rotation quaternions must always have a magnitude of 1, just like rotation matrices must have a determinant of +1.

Quaternions Representing Rotation

- To transform a point using a rotation quaternion, the point is represented as a quaternion with the real part equal to zero:

Point in Quaternion Form

$$P_A = 0 + x_A\hat{\imath} + y_A\hat{\jmath} + z_A\hat{k}$$

- The actual transformation is then performed by:

Quaternion Transformation

$$P_B = q_B^A P_A \left(q_B^A\right)^{-1}$$

- Because the magnitude of a rotation quaternion is 1, the inverse is just the complex conjugate:

$$P_B = q_B^A P_A \left(q_B^A\right)^{*}$$

Equivalent Rotation Matrix

- Expressing the transformation as a matrix equation:

$$P_B = \underbrace{\begin{bmatrix} w & -x & -y & -z \\ x & w & -z & y \\ y & z & w & -x \\ z & -y & x & w \end{bmatrix}}_{q_B^A} \underbrace{\begin{bmatrix} 0 & -x_A & -y_A & -z_A \\ x_A & 0 & -z_A & y_A \\ y_A & z_A & 0 & -x_A \\ z_A & -y_A & x_A & 0 \end{bmatrix}}_{P_A} \underbrace{\begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix}}_{\left(q_B^A\right)^*}$$

- This can be re-organized into a rotation matrix equation:

Rotation Matrix Equation

$$P_B = R_{q_B^A} P_A$$

Equivalent Rotation Matrix

- This yields the equivalent rotation matrix, which is defined in terms of the individual quaternion components:

Equivalent Rotation Matrix

$$R_{q_B^A} = \begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & w^2 - x^2 + y^2 - z^2 & 2(-wx + yz) \\ 2(xz - wy) & 2(wx + yz) & w^2 - x^2 - y^2 + z^2 \end{bmatrix}$$

Converting RPY Angles to Quaternion

- In addition to specifying the **a** vector and the β angle, a quaternion can also be computed in terms of RPY angles:

RPY to Quaternion

$$\begin{cases} w = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ x = \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ y = \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ z = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \end{cases}$$

TF Library

- The **tf** library contains classes and functions designed to help perform and manage transforms between different reference frames.
- Translation vectors are represented using the **tf::Vector3** class.
- Rotations are represented as Quaternions using the **tf::Quaternion** class.
- Complete transformations are represented using the **tf::Transform** class.

TF Library

- **tf::Transform** classes are typically populated with the **setOrigin** and **setRotation** methods.
- These methods take **tf::Vector3** and **tf::Quaternion** instances as arguments, respectively.

```
// Declare a tf::Transform class instance
tf::Transform transform;

// Set the translation to 1 meter in x
transform.setOrigin(tf::Vector3(1, 0, 0));

// Set rotation to identity
transform.setRotation(tf::Quaternion(0, 0, 0, 1));
```

TF Library

- To set the rotation to a yaw angle of 90°:

$$w = \cos\frac{\psi}{2} = 0.707,\ x = 0,\ y = 0,\ z = \sin\frac{\psi}{2} = 0.707$$

- Set the rotation using the quaternion directly:

```
transform.setRotation(tf::Quaternion(0, 0, 0.707, 0.707));
```

- ... or by using a handy function:

```
transform.setRotation(tf::createQuaternionFromYaw(M_PI / 2));
```

TF Library

- To set the rotation with roll-pitch-yaw angles of 0.2, 0.5, and 1.0 radians, either:
  - Compute the corresponding quaternion parameter values using Slide 13 and apply directly
  - Use another handy function:

```
transform.setRotation(tf::createQuaternionFromRPY(0.2, 0.5, 1.0));
```

Quaternions in ROS Messages

- ROS messages containing rotation information use a quaternion.
- The fundamental quaternion message type is the **geometry_msgs/Quaternion** message.
- A **geometry_msgs/Quaternion** message is included in other messages that have a rotation component.
- The most commonly used example of this is the **geometry_msgs/Pose** message, which contains both position and orientation components.

Quaternions in ROS Messages

```
micho@ubuntuvm: ~
micho@ubuntuvm:~$ rosmsg show geometry_msgs/Pose
geometry_msgs/Point position
  float64 x
  float64 y
  float64 z
geometry_msgs/Quaternion orientation
  float64 x
  float64 y
  float64 z
  float64 w

micho@ubuntuvm:~$ 
```

Quaternions in ROS Messages

- The **tf** library also has functions to generate quaternion ROS messages from RPY angle inputs, in addition to the ones that output **tf::Quaternion** class instances.

```
// Declare quaternion message structure
geometry_msgs::Quaternion q;

// Set a yaw angle
q = tf::createQuaternionMsgFromYaw(M_PI / 2);

// Set arbitrary RPY angles
q = tf::createQuaternionMsgFromRPY(0.2, 0.5, 1.0);
```