NavFuse Quaternion Class Design Description

Parker Barrett

February 18th, 2023

1 Class Overview

- Header File: NavFuse/include/rotations/Quaternion.hpp
- Implementation: NavFuse/src/rotations/Quaternion.cpp

The NavFuse Quaternion class contains functions for commonly used quaternion operations. These operations include basic arithmetic, normalization, rotating vectors, converting to alternate attitude representations and more.

2 Public Class Members

The following sub-sections describe the inputs, outputs and internal algorithms used in the public interface of the Quaternion class.

2.1 Quaternion::q0_, Quaternion::q1_, Quaternion::q2_, Quaternion::q3_

- Data Type: Double
- Description: Quaternion class variables containing the 4 quaternion elements for the current object. The scalar first convention is employed, with q0_ being the scalar component and q1_, q2_ and q3_ comprising the elements of the vector component.

2.2 Quaternion::Quaternion()

- Inputs
  - double q_0: Scalar quaternion element
  - double q_1: First vector component of quaternion
  - double q_2: Second vector component of quaternion
  - double q_3: Third vector component of quaternion
- Outputs
  - No Outputs
- Algorithm
  - The Quaternion class constructor which takes as an input each of the four quaternion elements and initializes the public class member variables, q0_, q1_, q2_ and q3_, to the values provided.

2.3 Quaternion::operator*()

- Inputs
  - Quaternion q_B: Quaternion class type object
- Outputs
  - Quaternion q_C: Quaternion class type object
- Algorithm
  - Multiplication override operator which takes in a Quaternion class type object and performs the right quaternion multiplication $q_C = q_A * q_B$, where q_A is the current Quaternion class values.

2.4 Quaternion::getQuaternion()

- Inputs
  - No Inputs
- Outputs
  - Eigen::Vector4d q: Vector containing quaternion elements
- Algorithm
  - Getter function which fills an Eigen::Vector4d data type with the elements of the quaternion class, $q = [q_0, q_1, q_2, q_3]$, and returns the vector.

2.5 Quaternion::isNormalized()

- Inputs
  - No Inputs
- Outputs
  - Bool normalized: Boolean indicating whether the current Quaternion class value is normalized
- Algorithm
  - Function which returns true if the magnitude of the current Quaternion class quaternion is equal to 1.0 within a tolerance of $1.0e^{-12}$.
  - The magnitude is computed using the Quaternion::magnitude() class member function.

2.6 Quaternion::normalize()

- Inputs
  - No Inputs
- Outputs
  - No Outputs
- Algorithm
  - Function which normalizes the current Quaternion class quaternion values.
  - The magnitude, mag, is first computed using the Quaternion::magnitude() class member function..
  - Each element of the quaternion is divided by the magnitude: $q_0 = \frac{q_0}{\text{mag}}$, $q_1 = \frac{q_1}{\text{mag}}$, $q_2 = \frac{q_2}{\text{mag}}$, $q_3 = \frac{q_3}{\text{mag}}$.

2.7 Quaternion::passiveRotateVector()

- Inputs
  - Eigen::Vector3d vecIn: 3D Input Vector
- Outputs
  - Eigen::Vector3d vecOut: 3D Output Vector
- Algorithm
  - Function which takes in a vector and performs the passive quaternion rotation.
  - The passive rotation is the rotation in which the coordinate system is rotated with respect to the point.
  - The rotation is performed by performing the quaternion multiplication, $p' = q^{-1}pq$, where p is a 4D quaternion type object with the scalar element set to zero and the vector component set equal to the input vector, vecIn.
  - The output vector, vecOut, is set equal to the vector component of the output quaternion value, p'.
  - The passive quaternion rotation is equivalent to converting the quaternion to a direction cosines matrix, R, and performing the rotation $v_{\text{out}} = Rv_{\text{in}}$

2.8 Quaternion::activeRotateVector()

- Inputs
  - Eigen::Vector3d vecIn: 3D Input Vector
- Outputs
  - Eigen::Vector3d vecOut: 3D Output Vector
- Algorithm
  - Function which takes in a vector and performs the active quaternion rotation.
  - The active rotation is the rotation in which the point is rotated with respect to the coordinate system.
  - The rotation is performed by performing the quaternion multiplication, $p' = qpq^{-1}$, where p is a 4D quaternion type object with the scalar element set to zero and the vector component set equal to the input vector, vecIn.
  - The output vector, vecOut, is set equal to the vector component of the output quaternion value, p'.
  - The active quaternion rotation is equivalent to converting the quaternion to a direction cosines matrix, R, and performing the rotation $v_{\text{out}} = R'v_{\text{in}}$, where R' is the transpose of the direction cosines matrix.

2.9 Quaternion::toDcm()

- Inputs
  - No Inputs
- Outputs
  - DirectionCosinesMatrix dcm: 3x3 Direction cosines matrix

- Algorithm
  - Function which converts the quaternion to a DirectionCosinesMatrix type object.
  - $R = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & q_0^2-q_1^2+q_2^2-q_3^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix}$

2.10 Quaternion::toEuler()

- Inputs
  - No Inputs
- Outputs
  - EulerAngles eul: 3-2-1 (Z-Y-X) Euler angles
- Algorithm
  - Function which converts the quaternion to an EulerAngles type object.
  - The pitch angle is calculated by $\theta = \arcsin(2(q_0q_2 - q_1q_3))$.
  - The roll angle is calculated by $\phi = \text{arctan2}(2(q_0q_1 + q_2q_3), q_0^2 - q_1^2 - q_2^2 + q_3^2)$
  - The yaw angle is calculated by $\psi = \text{arctan2}(2(q_0q_3 + q_1q_2), q_0^2 + q_1^2 - q_2^2 - q_3^2)$
  - Gimbal lock occurs when the pitch angle is equal to 90 or -90 degrees and the roll/yaw angles cannot be uniquely determined.
  - Special care is given to avoid gimbal lock in the software by setting the pitch angle to 90 or -90 and the roll angle to 0 in the cases where the pitch equation evaluates to greater than 89 or less than -89 respectively.
  - In the case where pitch is set to 90, the yaw angle is set to $\psi = -2\text{arctan2}(q_1, q_0)$.
  - In the case where pitch is set to -90, the yaw angle is set to $\psi = 2\text{arctan2}(q_1, q_0)$.

2.11 Quaternion::toRotationVector()

- Inputs
  - No Inputs
- Outputs
  - RotationVector rv: 3x1 rotation vector
- Algorithm
  - The rotation angle is computed by $\theta = 2\arccos(q_0)$.
  - The rotation vector is computed by $rv = \frac{\theta[q_1, q_2, q_3]}{\sin(\frac{\theta}{2})}$
  - In the case where $\theta = 0$, the rotation vector is set to the zero vector.

2.12 Quaternion::conjugate()

- Inputs
  - No Inputs
- Outputs
  - Quaternion q: Quaternion type object output
- Algorithm
  - The quaternion conjugate is computed by $q_{\text{conj}} = [q_0, -q_1, -q_2, -q_3]$.

2.13 Quaternion::inverse()

- Inputs
  - No Inputs
- Outputs
  - Quaternion q: Quaternion type object output
- Algorithm
  - The quaternion inverse is computed by dividing each element of the quaternion conjugate by the squared magnitude of the quaternion.
  - $q^{-1} = \frac{\text{conjugate(q)}}{\text{mag(q)}^2}$

2.14 Quaternion::multiply()

- Inputs
  - Quaternion qB: Quaternion type object input
- Outputs
  - Quaternion qC: Quaternion type object output
- Algorithm
  - The current quaternion class object, q_A is multiplied with the input quaternion class object, q_B to get $q_C = q_A q_B$.
  - $q_C = \begin{bmatrix} q_{A,0}q_{B,0} - q_{A,1}q_{B,1} - q_{A,2}q_{B,2} - q_{A,3}q_{B,3} \\ q_{A,0}q_{B,1} + q_{A,1}q_{B,0} + q_{A,2}q_{B,3} - q_{A,3}q_{B,2} \\ q_{A,0}q_{B,2} - q_{A,1}q_{B,3} + q_{A,2}q_{B,0} + q_{A,3}q_{B,1} \\ q_{A,0}q_{B,3} + q_{A,1}q_{B,2} - q_{A,2}q_{B,1} + q_{A,3}q_{B,0} \end{bmatrix}$