1 Szenario

In diesem Tutorial wird auf die Möglichkeiten eingegangen, einen Vektor $\vec{v}$ (oder eine Menge von Vektoren) im eigenen System zu rotieren. Grundsätzlich stehen dafür 4 Möglichkeiten bereit, mit denen das gleiche Resultat erzielt werden kann.

2 Rotation mit Rotationsmatrix

Eine Rotationsmatrix ist eine Matrix R mit 2 speziellen Eigenschaften:

- $det(R) = +1$
- $R \cdot R^t = I$

Die erste Eigenschaft wird schnell klar, wenn man sich in Erinnerung ruft, dass jede Matrix M auch eine lineare Abbildung darstellt. Die Determinante dieser Matrix M charackterisiert die lineare Abbildung insofern, dass sie die Streckung der Abbildung repräsentiert. So wird ein Vektor der Länge 1, nach einer Mulitplikation mit einer Matrix mit der Determinante $+2$ zu einem Vektor der Länge 2. Eine Rotation stellt eine Kongruenzabbildung dar, was bedeutet, dass weder Orientierungssinn(rechtshändisch,linkshändisch) bei einer Abbildung verändert werden, noch eine Strecke oder Winkel. Als einzig verbleibende Möglichkeit für die Determinante ist $det(M) = +1$, denn $det(M) = -1$ würde den Orientierungssinn unserer Matrix ändern.
Die zweite Eigenschaft zu zeigen ist etwas schwieriger, wie erwähnt werden Winkel bei einer Rotation nicht verändert, dann gilt insbesondere für das Skalarprodukt zweier Vektoren $\vec{v_1}$ und $\vec{v_2}$:

$$\begin{aligned} (M\vec{v_1})^t \cdot (M\vec{v_2}) &= \vec{v_1}^t \vec{v_2} \\ \vec{v_1}^t M^t M \vec{v_2} &= \vec{v_1}^t \vec{v_2} \\ \vec{v_1}^t M^t M \vec{v_2} - \vec{v_1}^t \vec{v_2} &= 0 \\ \text{und damit} \quad \vec{v_1}^t (M^t M - I) \vec{v_2} &= 0 \end{aligned}$$

Da dies nun für jedes Paar von Vektoren erfüllt sein muss, muss gelten:

$$\begin{aligned} M^t M - I &= 0 \\ \text{und letztendlich} \quad M M^t &= I \end{aligned}$$

Nachdem nun bekannt ist, warum eine solche Matrix eine Rotation darstellt, folgt nun ein Implementierungsbeispiel

```
Vector3D v(1,1,1) // Ausgangsvektor
Matrix3D R        // Einheitsmatrix

Vector3D rotated = v.mRotate(R);  // w = R*v
```

3 Rotation mit einem Quaternion

Ein Rotationsquaternion q ist ein Quaternion, mit der Norm 1. In diesem Tutorial wird nicht darauf eingegangen, warum ein Quaternion generell eine Rotation darstellt, es wird lediglich die Idee präsentiert. Eine Möglichkeit eine Rotation darzustellen, besteht darin, sich eine beliebige Achse $\vec{u}$ im Raum zu suchen, und um diese Achse mit einem Winkel α zu drehen. Für diese Möglichkeit der Darstellung werden 4 Parameter benötigt, 3 für die Achse $\vec{u}$ und einen für den Winkel α. Ein Quaternion ist eben ein solches 4-Tupel der Form q $(q_0, \vec{q})$. Mit Hilfe eines solchen Quaternions lässt sich eine Rotation nun wie folgt darstellen:

$$\vec{w} = q^* \vec{v} q$$

Wobei q^* das zu q Konjugierte Quaternion darstellt. Implementieren lässt sich eine Quaternionrotation wie folgt:

```
Vector3D v(1,1,1) // Ausgangsvektor
Quaternion q(1,0,0,0) // Identity

Vector3D rotated = v.qRotate(q) // w = q*vq
```

4 Rotation mit Eulerwinkeln

Gemäß Eulers Theorem kann jede Rotation in 3 Teilrotationen um die Koordinatenachsen zerlegt werden, Die Drehwinkel dieser Rotationen werden als Eulerwinkel bezeichnet. In der Luftfahrt werden vorallem Yaw, Pitch und Roll verwendet um dies zu charackterisieren. Um mit Hilfe dieser Winkel eine Rotation auszuführen muss zunächst das zugehörige Quaternion oder die dazugehörige Rotationsmatrix erzeugt werden, und anschließend kann analog zu den oben beschriebenen Methoden rotiert werden.

Um Yaw,Pitch und Roll in eine Matrix, bzw. in ein Quaternion umzuwandeln stehen folgende Methoden zur Verfügung:

```
YPR y(M_PI/2,M_PI/4,M_PI/6);
Quaternion q = y.toQuat();
Matrix3D R = y.toMatrix3D();
```

Die Rotation wird wie folgt durchgeführt:

```
Vector3D v(1,1,1);
Vector3D rotated = mRotate(R);
Vector3D rotated = qRotate(q);
```

5 Rotation um beliebige Achse und Winkel

Kennt man nun seine Drehachse und den dazugehörigen Drehwinkel um diese Achse, so erzeugt man sich am leichtesten ein AngleAxis un führt mit diesem, gemäß Rodriguez Rotationsformel die Rotation durch:

```
Vector3D v(1,1,1);
Vector3D u(1,0,0) // Drehachse
double angle = M_PI/2 // Drehwinkel
AngleAxis u_phi(u,angle);

Vector3D rotated = v.vecRotate(u_phi);
```