*Sensors* **2015**, *15*, 7016-7039; doi:10.3390/s150307016

OPEN ACCESS

*sensors*

**ISSN 1424-8220**

www.mdpi.com/journal/sensors

*Article*

# Improving the Precision and Speed of Euler Angles Computation from Low-Cost Rotation Sensor Data

**Aleš Janota *, Vojtech Šimák, Dušan Nemec and Jozef Hrbček**

Department of Control & Information Systems, Faculty of Electrical Engineering, University of Žilina, Univerzitná 8215/1, Žilina 010 26, Slovakia; E-Mail: vojtech.simak@fel.uniza.sk (V.Š.); nemec.dusan666@gmail.com (D.N.); jozef.hrbcek@fel.uniza.sk (J.H.)

* Author to whom correspondence should be addressed; E-Mail: ales.janota@fel.uniza.sk; Tel.: +421-41-513-3356; Fax: +421-41-513-1515.

Academic Editor: Stefano Mariani

*Received: 5 January 2015 / Accepted: 17 March 2015 / Published: 23 March 2015*

**Abstract:** This article compares three different algorithms used to compute Euler angles from data obtained by the angular rate sensor (e.g., MEMS gyroscope)—the algorithms based on a rotational matrix, on transforming angular velocity to time derivations of the Euler angles and on unit quaternion expressing rotation. Algorithms are compared by their computational efficiency and accuracy of Euler angles estimation. If attitude of the object is computed only from data obtained by the gyroscope, the quaternion-based algorithm seems to be most suitable (having similar accuracy as the matrix-based algorithm, but taking approx. 30% less clock cycles on the 8-bit microcomputer). Integration of the Euler angles' time derivations has a singularity, therefore is not accurate at full range of object's attitude. Since the error in every real gyroscope system tends to increase with time due to its offset and thermal drift, we also propose some measures based on compensation by additional sensors (a magnetic compass and accelerometer). Vector data of mentioned secondary sensors has to be transformed into the inertial frame of reference. While transformation of the vector by the matrix is slightly faster than doing the same by quaternion, the compensated sensor system utilizing a matrix-based algorithm can be approximately 10% faster than the system utilizing quaternions (depending on implementation and hardware).

**Keywords:** gyroscope; Euler angle; inertial navigation; MEMS; rotational matrix

## 1. Introduction

Micro-Electro-Mechanical systems (MEMS) represent the integration of mechanical elements, sensors, actuators, and electronics on a common silicon substrate through the utilization of microfabrication technology [1]. The number of MEMS used in various applications is permanently growing due to the small dimensions, light weight, lower power consumption, higher reliability, and relatively low cost which makes them commercially available. Typical MEMS-based low-cost products are accelerometers, gyroscopes, pressure sensors, microphones, digital mirror displays, micro pumps, *etc.* For the purpose of low-cost navigation solutions MEMS-based inertial sensors (accelerometers and gyroscopes) have been developed since orientation of an object in the three-dimensional space is key information needed for navigation, guidance and control tasks. MEMS inertial sensors may be found in variety of applications from traditional ones (navigation and positioning of various transport means and/or robots) to sensing of human body walking and movement [2–5], daily life surveillance [6] or new commercial applications available through smart phones [7]. Most studies on MEMS gyroscopes are focused on their performance, and common methods to improve the performance [8]. Unlike non-micro devices MEMS sensors experience more errors that build up over time, corrupting the precision of the measurements and eventually rendering the navigation solution useless [9,10]. Thus the first and easiest-to-measure performance criterion of a gyro is its static readout as a function of time. Accuracy is usually limited by electrical noise, systematic errors and/or mechanical thermal noise [11,12]. The static compensation of sensor inaccuracies can be enabled by proper calibration methods designed for MEMS gyroscopes and accelerometers [13]. The principle of recently developed micro-machine gyroscopes, their structures and classification can be found in [14].

Generally, gyroscopes measure rotational rate, which can be integrated to yield changes in orientation. An effective method most used to parametrize the orientation space is based on usage of so called Euler angles. Euler angles are used as a framework for formulating and solving the equations for conservation of angular momentum. This article has been written with motivation to analyze and show how precision and speed of computations of Euler angles could be improved when processing data from the MEMS gyroscope. It is organized as follows: Section 1 (Introduction) describes theoretically several methods of notation to express rotation of a body (particularly the rotation matrix, Euler angles, rotation around arbitrary axis, and quaternion). Section 2 (Experimental Section) is focused on comparison of errors occurring when algorithms utilizing described notations process data from the gyroscope. If applicable, more versions of the same algorithm are considered (focused either on accuracy or fastness of computation). At the end of the section there is discussion on how errors presented in real gyroscopes could be compensated. Section 3 (Results and Discussion) summarizes analyzed properties and gives final comparison and overview of obtained results. Finally, Section 4 gives the conclusions. The article is an extended version of the conference paper [15], elaborated and supported by the VEGA1/0453/12 grant and used with kind permission of Springer Science + Business Media. Article extensions resulted from the work under another project as stated in the Acknowledgments section.

The purpose of the inertial navigation in the 3D space is to determine six independent variables: translation of an object in three axes and its rotation in three axes, relative to the inertial frame of the

reference body. In this article we describe possible ways how to express rotation (attitude) of the object and calculate it from angular velocity measured by the gyroscope.

We consider the Cartesian (orthonormal) right-hand coordinate system oriented by convention NED, *i.e.*, North-East-Down. Moving object axes' orientations are $x \rightarrow$ forward, $y \rightarrow$ right and $z \rightarrow$ down (Figure 1). Two reference frames are used:

- Frame of reference joined with Earth (considered to be approximately inertial), marked *S*. All variables measured with respect to Earth will be marked without a dash.
- Frame of reference joined with rotating object, marked S'. All variables measured onboard the moving object will be marked with a dash.


**Figure 1.** Orientation of the coordinate system axes.

First we will analyze four used methods of notation that allow us to express rotation of a body. Differences among those individual approaches can be seen in data redundancy and consumption of computer time during processing of raw data from the gyroscope and during conversion from one notation to another (which has direct impact on algorithm efficiency).

### *1.1. Euler Angles*

Euler angles are expressing rotation of the object as a sequence of three rotations around objects' local coordinate axes. This way of rotation expression is most interpretative and has zero data redundancy because only three real numbers are needed. Different sequence of axis rotation produces different resultant rotation; therefore Euler angles are defined according to chosen sequence (convention). In aviation the most used convention is *z-y-x* convention (sometimes called Yaw-Pitch-Roll convention or 3-2-1, see Figure 2):

1. Rotate the object around its *z*-axis by angle Yaw (marked $\gamma$);
2. Rotate the object around its new $y_1$-axis by angle Pitch (marked $\beta$);
3. Rotate the object around its new $x_2$-axis by angle Roll (marked $\alpha$).

Rotation order of *z-y-x* convention can be expressed by the following operator:

$$\Re_{\alpha,\beta,\gamma} = \Re_{\alpha}^{x_2}\left(\Re_{\beta}^{y_1}\left(\Re_{\gamma}^{z}\right)\right) \tag{1}$$

Inverse rotation is given by the reversed rotation order by inverted angles:

$$\Re_{\alpha,\beta,\gamma}^{-1} = \Re_{-\gamma}^{z_1}\left(\Re_{-\beta}^{y_2}\left(\Re_{-\alpha}^{x'}\right)\right) \tag{2}$$

Main disadvantage of representing object's rotation by Euler angles is a lack of the simple algorithm for vector transformation. This can be realized by transferring Euler angles to the rotation matrix by Equation (10) and following application of Equation (4). Trivial chaining (adding) of two rotations represented by Euler angles is not possible.

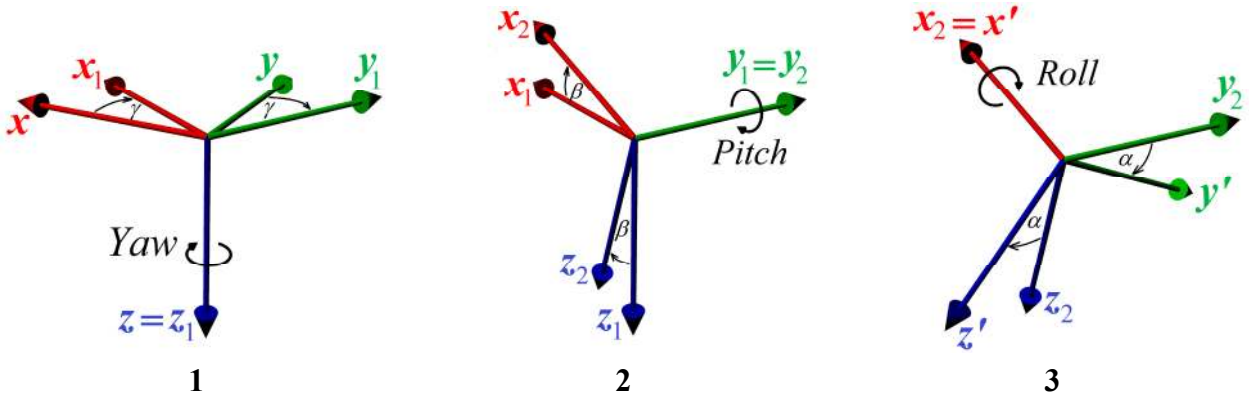


**Figure 2.** Euler angles for 3-2-1 convention.

*1.2. The Rotational Matrix*

The rotation matrix defines change of coordinates of the object in the coordinate system $S$ during rotational movement. It is a typical representation of object's attitude (very often used, e.g., in computer graphics). It is clear that this form has the greatest data redundancy due to needs of saving nine real numbers:

$$\boldsymbol{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{3}$$

Transformation of coordinates from the system $S$ to the system $S'$ can be done by multiplication of the position column vector $\boldsymbol{r}$ by the rotation matrix:

$$\boldsymbol{r}' = \boldsymbol{R} \cdot \boldsymbol{r} \tag{4}$$

Result of rotation $\boldsymbol{R}_1$ followed by $\boldsymbol{R}_2$ is given by matrix multiplication:

$$\boldsymbol{R} = \boldsymbol{R}_2 \cdot \boldsymbol{R}_1 \tag{5}$$

Inverse rotation is given by the transposed matrix:

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^T \tag{6}$$

While the original vector $\boldsymbol{r}$ has the same length as the resultant vector $\boldsymbol{r}'$, the rotational matrix has to be orthogonal with its determinant equal to 1. The matrix is orthogonal when all its row or column vectors are perpendicular to each other. The following algorithm can be used to normalize the matrix to be pure rotational [16]:

1. Calculate deviations $e_{ik}$ from orthogonality of the matrix columns:

$$\boldsymbol{X} = \begin{bmatrix} R_{11} \\ R_{21} \\ R_{31} \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} R_{12} \\ R_{22} \\ R_{32} \end{bmatrix} \quad \boldsymbol{Z} = \begin{bmatrix} R_{13} \\ R_{23} \\ R_{33} \end{bmatrix} \qquad \begin{aligned} e_{xy} &= \boldsymbol{X} \cdot \boldsymbol{Y} \\ e_{yz} &= \boldsymbol{Y} \cdot \boldsymbol{Z} \\ e_{zx} &= \boldsymbol{Z} \cdot \boldsymbol{X} \end{aligned} \tag{7}$$

2. Distribute errors among all columns:

$$\begin{aligned} \boldsymbol{X}_{\text{ort}} &= \boldsymbol{X} - \frac{e_{xy}}{2}\boldsymbol{Y} - \frac{e_{zx}}{2}\boldsymbol{Z} \\ \boldsymbol{Y}_{\text{ort}} &= \boldsymbol{Y} - \frac{e_{xy}}{2}\boldsymbol{X} - \frac{e_{yz}}{2}\boldsymbol{Z} \\ \boldsymbol{Z}_{\text{ort}} &= \boldsymbol{Z} - \frac{e_{zx}}{2}\boldsymbol{X} - \frac{e_{yz}}{2}\boldsymbol{Y} \end{aligned} \tag{8}$$

3. Normalize columns to be unit vectors. Approximate formula (first order Taylor series) can be used only if normalization is performed incrementally by small steps (after each updating of the rotational matrix):

$$\begin{aligned} \boldsymbol{X}_{\text{norm}} &= \frac{\boldsymbol{X}_{\text{ort}}}{\left|\boldsymbol{X}_{\text{ort}}\right|} \approx \frac{1}{2}\left(3 - \boldsymbol{X}_{\text{ort}} \cdot \boldsymbol{X}_{\text{ort}}\right)\boldsymbol{X}_{\text{ort}} \\ \boldsymbol{Y}_{\text{norm}} &= \frac{\boldsymbol{Y}_{\text{ort}}}{\left|\boldsymbol{Y}_{\text{ort}}\right|} \approx \frac{1}{2}\left(3 - \boldsymbol{Y}_{\text{ort}} \cdot \boldsymbol{Y}_{\text{ort}}\right)\boldsymbol{Y}_{\text{ort}} \\ \boldsymbol{Z}_{\text{norm}} &= \frac{\boldsymbol{Z}_{\text{ort}}}{\left|\boldsymbol{Z}_{\text{ort}}\right|} \approx \frac{1}{2}\left(3 - \boldsymbol{Z}_{\text{ort}} \cdot \boldsymbol{Z}_{\text{ort}}\right)\boldsymbol{Z}_{\text{ort}} \end{aligned} \tag{9}$$

Conversion from 3-2-1 Euler angles to the rotational matrix is given by the following formula [17]:

$$\boldsymbol{R} = \boldsymbol{R}_x(\alpha) \cdot \boldsymbol{R}_y(\beta) \cdot \boldsymbol{R}_z(\gamma) = \begin{bmatrix} c_\beta c_\gamma & c_\beta s_\gamma & -s_\beta \\ s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma & s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha c_\beta \\ c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha c_\beta \end{bmatrix} \tag{10}$$

where:

$$\begin{aligned} c_\alpha &= \cos\alpha \quad s_\alpha = \sin\alpha \\ c_\beta &= \cos\beta \quad s_\beta = \sin\beta \\ c_\gamma &= \cos\gamma \quad s_\gamma = \sin\gamma \end{aligned} \tag{11}$$

Conversion from the rotational matrix to 3-2-1 Euler angles can be done by the following algorithm:

$$\begin{array}{lll} \text{if } R_{13} \le -1: & \text{else if } R_{13} \ge 1: & \text{else}: \\ \quad \alpha = 0 & \quad \alpha = 0 & \quad \alpha = \text{atan2}\left(R_{23}, R_{33}\right) \\ \quad \beta = \pi/2 & \quad \beta = -\pi/2 & \quad \beta = \arcsin\left(-R_{13}\right) \\ \quad \gamma = -\text{atan2}\left(R_{21}, R_{31}\right) & \quad \gamma = \text{atan2}\left(-R_{32}, R_{22}\right) & \quad \gamma = \text{atan2}\left(R_{12}, R_{11}\right) \end{array} \tag{12}$$

The function atan2(y, x) is a four quadrant inverse tangent function, *i.e.*, arctangent function extended to the output angle interval from –π to π. Inputs x and y are coordinates of any point in 2D plane, output is an oriented angle between x-axis and the vector [x, y]. Function is supported by many

programming languages by standard (e.g., C-language), having two arguments. The purpose of using two arguments instead of one is to gather information on the signs of the inputs in order to return the appropriate quadrant of the computed angle, which is not possible for the single-argument arctangent function.

### *1.3. Rotation around Arbitrary Axis*

According to the Euler theorem it is possible to replace every rotation representation by simple rotation around angle $\theta$ around the arbitrary axis given by the unit vector $\boldsymbol{n} = \boldsymbol{n}' = [n_x, n_y, n_z]$ (length of the axis vector is $|\boldsymbol{n}| = 1$). Note that the axis vector has the same coordinates in the inertial system $S$ and the body-fixed system $S'$.

Transformation of the vector $\boldsymbol{r}$ from the system $S$ to $S'$ is expressed by the Rodriguez rotation formula:

$$\boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{n} \times \left[\boldsymbol{r} \sin\theta + (\boldsymbol{n} \times \boldsymbol{r})(1 - \cos\theta)\right] \tag{13}$$

Inverse rotation is expressed by the identical axis $\boldsymbol{n}$ and opposite angle $-\theta$. Chaining of two rotations around non-parallel axes of rotation is impossible to implement trivially, transformation to another type of expression is needed.

### *1.4. Quaternion*

Quaternion (invented by sir William Rowan Hamilton in 1843) is a modification of rotation around arbitrary axis expression utilizing algebra of complex numbers expanded to three imaginary dimensions with the complex units $i, j, k$, for which it is valid:

$$\begin{array}{lll} \mathbf{i}.\mathbf{i} = -1 & \mathbf{i}.\mathbf{j} = \mathbf{k} & \mathbf{i}.\mathbf{k} = -\mathbf{j} \\ \mathbf{j}.\mathbf{i} = -\mathbf{k} & \mathbf{j}.\mathbf{j} = -1 & \mathbf{j}.\mathbf{k} = \mathbf{i} \\ \mathbf{k}.\mathbf{i} = \mathbf{j} & \mathbf{k}.\mathbf{j} = -\mathbf{i} & \mathbf{k}.\mathbf{k} = -1 \end{array} \tag{14}$$

Based on the expanded Euler's formula, the rotation for quaternion around the axis $\boldsymbol{n} = [n_x, n_y, n_z]$ by angle $\theta$ is defined as follows:

$$\boldsymbol{q} = \left(n_x.\boldsymbol{i} + n_y.\boldsymbol{j} + n_z.\boldsymbol{k}\right)\sin\frac{\theta}{2} + \cos\frac{\theta}{2} = x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k} + w \tag{15}$$

While the axis $\boldsymbol{n}$ is a unit 3D vector, quaternion must follow unit constraint to be pure rotational:

$$|\boldsymbol{q}| = \sqrt{x^2 + y^2 + z^2 + w^2} = 1 \tag{16}$$

Normalization of quaternion is done by the similar way like normalization of any vector. An approximate formula (like matrix normalization) can be used only if normalization is performed after each update of the quaternion:

$$\boldsymbol{q}_{\text{normalized}} = \frac{\boldsymbol{q}}{\sqrt{x^2 + y^2 + z^2 + w^2}} \approx \frac{\left(3 - x^2 - y^2 - z^2 - w^2\right)}{2}\boldsymbol{q} \tag{17}$$

The advantage of quaternions is quick computing of chaining of rotation $\boldsymbol{q}_1$ followed by $\boldsymbol{q}_2$ utilizing Hamilton's product:

$$\begin{aligned}\boldsymbol{q} &= \boldsymbol{q}_2\boldsymbol{q}_1 = (x_2\boldsymbol{i} + y_2\boldsymbol{j} + z_2\boldsymbol{k} + w_2)(x_1\boldsymbol{i} + y_1\boldsymbol{j} + z_1\boldsymbol{k} + w_1) \\ &= (x_1w_2 + w_1x_2 + z_1y_2 - y_1z_2)\boldsymbol{i} + \\ &+ (y_1w_2 - z_1x_2 + w_1y_2 + x_1z_2)\boldsymbol{j} + \\ &+ (z_1w_2 + y_1x_2 - x_1y_2 + w_1z_2)\boldsymbol{k} + \\ &+ (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2)\end{aligned} \tag{18}$$

There are two basic variants of vector transformation utilizing quaternion. The first one takes the transformed vector as a quaternion $\mathbf{r} = r_x\mathbf{i} + r_y\mathbf{j} + r_z\mathbf{k} + 0$:

$$\mathbf{r}' = r_x'\mathbf{i} + r_y'\mathbf{j} + r_z'\mathbf{k} = \mathbf{qrq}^{-1} \tag{19}$$

Concerning speed it is better to use the following formula:

$$\mathbf{r}' = \mathbf{r} + 2\hat{\mathbf{q}} \times [\mathbf{r}w + (\hat{\mathbf{q}} \times \mathbf{r})] \tag{20}$$

where $\hat{\boldsymbol{q}}$ is a vector part of quaternion:

$$\hat{\boldsymbol{q}} = [x, y, z] = \boldsymbol{n}\sin\frac{\theta}{2} \tag{21}$$

Conversion from 3-2-1 Euler angles to unit quaternion is given by the following formula:

$$\boldsymbol{q} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} s_x c_y c_z - c_x s_y s_z \\ c_x s_y c_z + s_x c_y s_z \\ c_x c_y s_z - s_x s_y c_z \\ -c_x c_y c_z - s_x s_y s_z \end{bmatrix} \quad \text{where} \quad \begin{aligned} s_x &= \sin(\alpha/2) \quad c_x = \cos(\alpha/2) \\ s_y &= \sin(\beta/2) \quad c_y = \cos(\beta/2) \\ s_z &= \sin(\gamma/2) \quad c_z = \cos(\gamma/2) \end{aligned} \tag{22}$$

Conversion from quaternion to 3-2-1 Euler angles can be done by the following algorithm:

$$\begin{aligned} &R_{13} = 2(xz + yw) \\ &\text{if } R_{13} \geq -1: \\ &\quad \alpha = 0 \\ &\quad \beta = \pi/2 \\ &\quad \gamma = -\text{atan2}(xy + yz,\ xz - yw) \\ &\text{else if } R_{13} \leq 1: \\ &\quad \alpha = 0 \\ &\quad \beta = -\pi/2 \\ &\quad \gamma = \text{atan2}(-yz - xw,\ 2 - x^2 - z^2) \\ &\text{else}: \\ &\quad \alpha = \text{atan2}(yz - xw,\ 2 - x^2 - y^2) \\ &\quad \beta = \arcsin(-R_{13}) \\ &\quad \gamma = \text{atan2}(xy - zw,\ 2 - y^2 - z^2) \end{aligned} \tag{23}$$

## 2. Experimental Section

In this section we compare errors of Euler angle estimation caused by algorithms processing gyroscopic data and being based on different rotation notations. These errors increase during run-time and depend on sampling frequency. The gyroscope firmly joined with the moving object $S'$ is measuring angular velocity as a tri-component vector $\boldsymbol{\omega}' = [\omega'_x, \omega'_y, \omega'_z]$. These data are sampled with given sample frequency $f_{\text{sample}} = 1/\Delta T$. The sensor system has to process data sample by sample in real-time (Figure 3). As mentioned above, outputs of the algorithm are Euler angles $\alpha$, $\beta$, $\gamma$, the system should also provide utility of the transformation of the vector from the $S$ to $S'$ coordinate system.

In order to eliminate influence of the sensor itself a model of the ideal digital-output gyroscope with the following properties was used for algorithm testing:

- Gyroscope output in each axis is a signed integer with 16-bit precision (like in many of available low-cost gyroscopes). Full-scale range of the output angular rate is ±500°/s.
- No noise is present at gyroscope output; also sampling frequency is absolutely precise (we want to examine errors of data processing algorithms, not precision of data itself). Therefore, data simulation was used instead of real experiment.

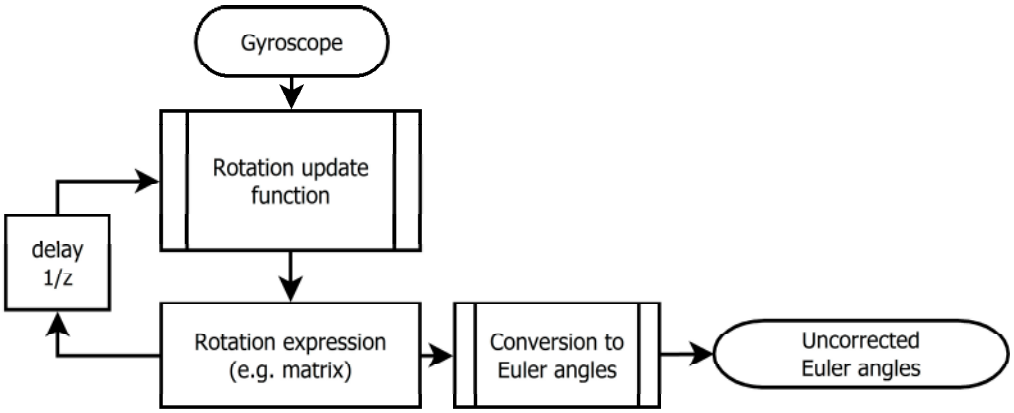


**Figure 3.** Schematics expressing principle of real-time gyroscope data processing.

In order to obtain comparable results, simulated movement of the object has to be exactly the same for all experiments. Therefore the pre-defined non-random movement has to be simulated. As a test input for algorithms we used a model of precession motion with perpendicular precession axis (see Figure 4). Such rotational movement is easy to define and also it is possible to analytically compute object's attitude (Euler angles) at any time.

Angular velocity of primary rotation and precession was chosen $A = 1$ rad·s$^{-1}$. Simulated angular velocity of the object (measured in its frame of reference) is then given by following:

$$\omega'_x(t) = A; \quad \omega'_y(t) = A\sin(At); \quad \omega'_z(t) = A\cos(At) \tag{24}$$

Simulation time corresponds to 20 turns ($t_{\text{end}} = 40\pi/A \approx 2$ min). Euler angles during simulated movement are shown in Figure 5. Initial rotation is $\{\alpha_0 = 0°, \beta_0 = 60°, \gamma_0 = 0°\}$. Euler angles (3-2-1 convention) during defined movement are given by following:

$$\begin{aligned}\alpha(t) &= At + \text{atan2}(\sin\beta_0 \sin At\ ,\ \cos\beta_0)\\ \beta(t) &= \arcsin(\sin\beta_0 \cos At)\\ \gamma(t) &= \text{atan2}(\sin At\ ,\ \cos\beta_0 \cos At)\end{aligned} \tag{25}$$

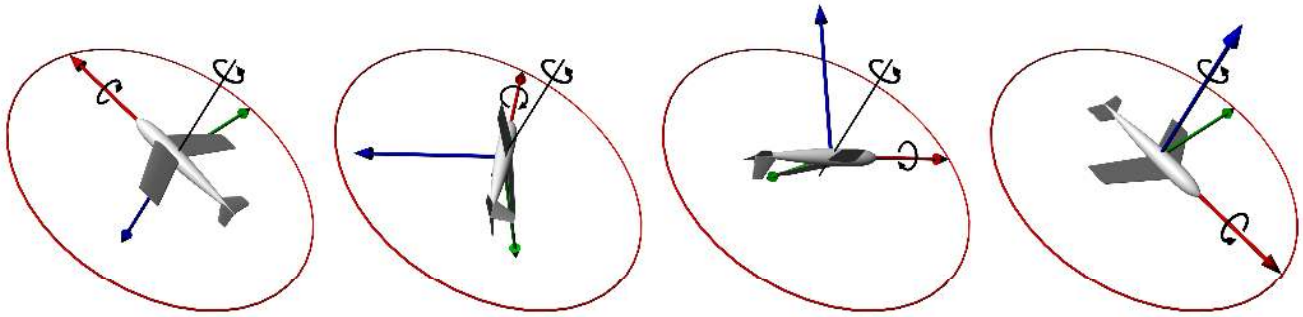

**Figure 4.** Half turn of the simulated precession movement.

Resulting error of the algorithm is considered to be the maximal deviation of Euler angles $\tilde{\alpha}(t), \tilde{\beta}(t), \tilde{\gamma}(t)$ estimated by the algorithm from the correct Euler angles $\alpha(t), \beta(t), \gamma(t)$ during the simulation:

$$err = \max\left(\left|\tilde{\alpha}(t) - \alpha(t)\right|, \left|\tilde{\beta}(t) - \beta(t)\right|, \left|\tilde{\gamma}(t) - \gamma(t)\right|\right) \tag{26}$$

Note that the difference between two angles has to be computed as angular difference (e.g., difference between 180° and −180° is zero) and the maximal shown error is 180°.

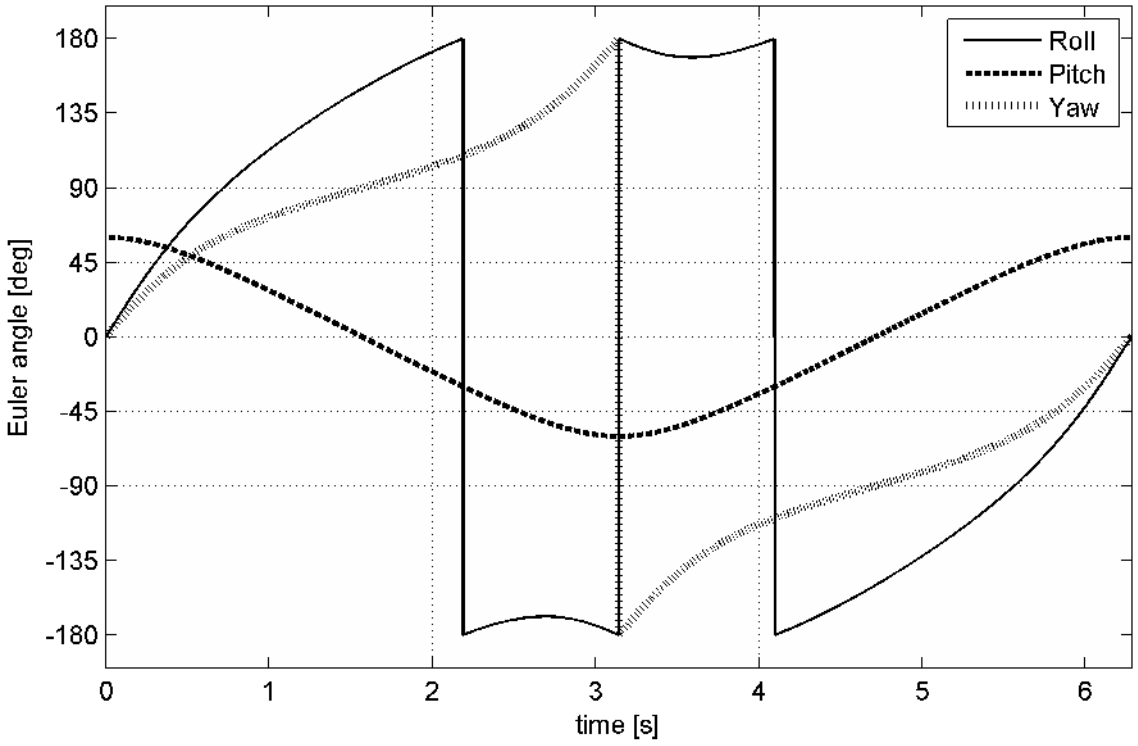


**Figure 5.** Euler angles during one turn of the simulated movement.

### *2.1. The Algorithm Based on Updating of the Rotational Matrix*

The first version of the algorithm for processing of measured angular velocity is utilizing a matrix as a primary expression of rotation. The principle of this method is shown in Figure 6.

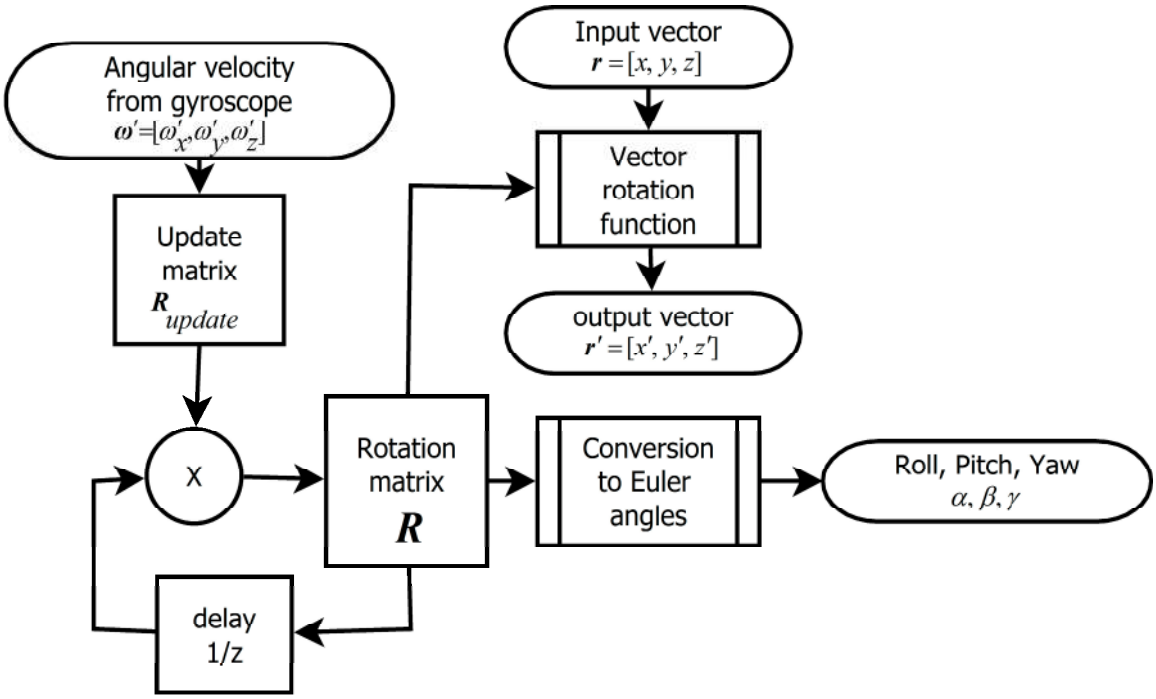


**Figure 6.** Precise version of the algorithm based on a rotation matrix.

The original rotation matrix $\boldsymbol{R}_n$ is multiplied by the update matrix $\boldsymbol{R}_{\text{update}}$:

$$\boldsymbol{R}_{n+1} = \boldsymbol{R}_{\text{update}} \cdot \boldsymbol{R}_n \tag{27}$$

The update matrix defines rotation of the object between 2 recent samples of the angular velocity vector $\boldsymbol{\omega}'$ (samples $\boldsymbol{\omega}_{n-1}$ and $\boldsymbol{\omega}_n$) with time span $\Delta T$. It is possible to create the update matrix from angular velocity by two ways—precise and fast. Fast version uses linear approximation of sine and cosine functions which significantly reduces computational demands; precise version uses non-linear goniometric functions. There is a possibility of using Taylor series of higher order as an approximation of sine and cosine functions.

#### 2.1.1. Precise Version

We can assume that between 2 samples there is constant angular velocity, so its direction defines rotation axis and magnitude multiplied by sample period $\Delta T$ defines the angle of rotation:

$$\begin{aligned} \mathbf{n} &= \frac{\boldsymbol{\omega}'}{|\boldsymbol{\omega}'|} = \frac{\boldsymbol{\omega}'}{\sqrt{\omega_x'^2 + \omega_y'^2 + \omega_z'^2}} = [n_x, n_y, n_z] \\ \theta &= |\boldsymbol{\omega}'|\Delta T \end{aligned} \tag{28}$$

The corresponding update matrix is:

$$\boldsymbol{R}_{\text{update}} = \begin{bmatrix} c + n_x{}^2(1-c) & n_x n_y(1-c) + n_z s & n_x n_z(1-c) - n_y s \\ n_y n_x(1-c) - n_z s & c + n_y{}^2(1-c) & n_y n_z(1-c) + n_x s \\ n_z n_x(1-c) + n_y s & n_z n_y(1-c) - n_x s & c + n_z{}^2(1-c) \end{bmatrix} \tag{29}$$

where $c = \cos\theta$ and $s = \sin\theta$.

If we use substitution:

$$\begin{aligned} &u = 1 - c \\ &u_x = n_x u \quad u_y = n_y u \quad u_z = n_z u \\ &s_x = n_x s \quad s_y = n_y s \quad s_z = n_z s \\ &r_{xx} = n_x u_x \quad r_{yy} = n_y u_y \quad r_{zz} = n_z u_z \\ &r_{xy} = n_x u_y \quad r_{yz} = n_y u_z \quad r_{zx} = n_z u_x \end{aligned} \tag{30}$$

We obtain:

$$\boldsymbol{R}_{\text{update}} = \begin{bmatrix} r_{xx} + c & r_{xy} + s_z & r_{zx} - s_y \\ r_{xy} - s_z & r_{yy} + c & r_{yz} + s_x \\ r_{zx} + s_y & r_{yz} - s_x & r_{zz} + c \end{bmatrix} \tag{31}$$

2.1.2. Fast Version

In case of high sampling frequency, we can use the infinitesimal rotation matrix based on the first order approximation of trigonometric functions:

$$\lim_{x \to 0} \sin x = x \quad \lim_{x \to 0} \cos x = 1 \tag{32}$$

The update matrix has a form of the infinitesimal rotational matrix:

$$\boldsymbol{R}_{\text{update}} = \begin{bmatrix} 1 & \omega'_z dT & -\omega'_y dT \\ -\omega'_z dT & 1 & \omega'_x dT \\ \omega'_y dT & -\omega'_x dT & 1 \end{bmatrix} \tag{33}$$

Because of the linearity of equations (there is no need for calculation of trigonometric functions or normalization of the axis vector) this is the fastest of all mentioned methods (it is about 3-times faster than precise version, depending on the used hardware). However, the main disadvantage is low accuracy, which constrains this algorithm for systems with high sampling frequency.

Figure 7 compares the fast and precise versions by their relative errors with respect to sampling frequency. Expression of rotation based on rotational matrices does not contain any singularities; therefore it is working with constant precision for every tilt. The advantage is also the quick algorithm of vector transformation. In order to maintain rotation matrix orthogonality, normalization is strongly recommended if fast version of the matrix-based algorithm is used. Shown results are computed after normalization in each step.

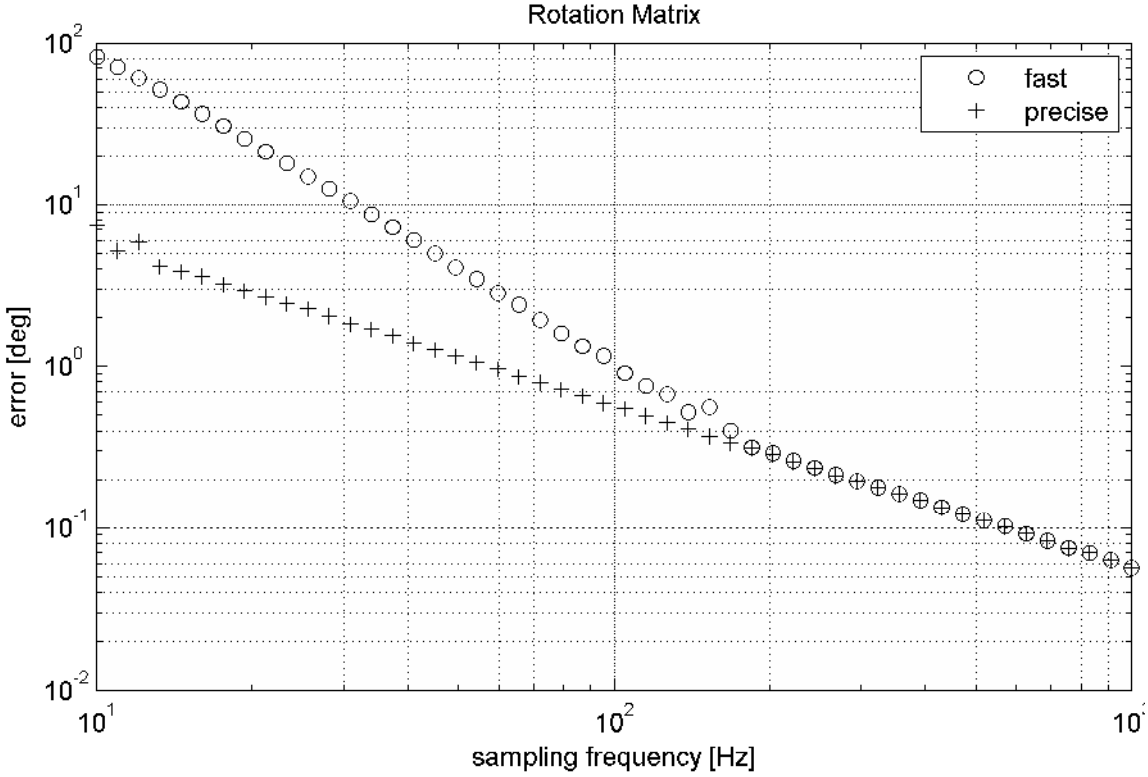


**Figure 7.** Errors of the matrix-based algorithms during simulated movement.

*2.2. The Algorithm Based on the Integration of the Euler Angle Rates*

Using this algorithm it is possible to avoid intermediate expression of rotation (e.g., by the matrix) and following need for conversion to Euler angles. The principle is shown in Figure 8. This version uses relation between angular velocity $\boldsymbol{\omega}'$ measured in the coordinate system $S'$ and time derivations of Euler angles (Euler angle rates):

$$\begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} 1 & \dfrac{\sin\alpha\sin\beta}{\cos\beta} & \dfrac{\cos\alpha\sin\beta}{\cos\beta} \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \dfrac{\sin\alpha}{\cos\beta} & \dfrac{\cos\alpha}{\cos\beta} \end{bmatrix} \cdot \begin{bmatrix} \omega'_x \\ \omega'_y \\ \omega'_z \end{bmatrix} \tag{34}$$

By integration of Euler angle rates $\dot{\alpha}, \dot{\beta}, \dot{\gamma}$ we get resulting Euler angles. There are two algorithms of numerical integration used in real-time processing:

Step integration:

$$\alpha_{n+1} = \alpha_n + \dot{\alpha}_{n+1}\Delta T \tag{35}$$

Trapezoidal integration:

$$\alpha_{n+1} = \alpha_n + (\dot{\alpha}_{n-1} + \dot{\alpha}_n)\frac{\Delta T}{2} \tag{36}$$

Although trapezoidal integration is usually more precise than simple step integration, according to Figure 9 step integration is in case of Euler angle rates little more precise. This is caused by non-linearity of transformation Equation (34). The algorithm is precise enough only at high sampling frequency.

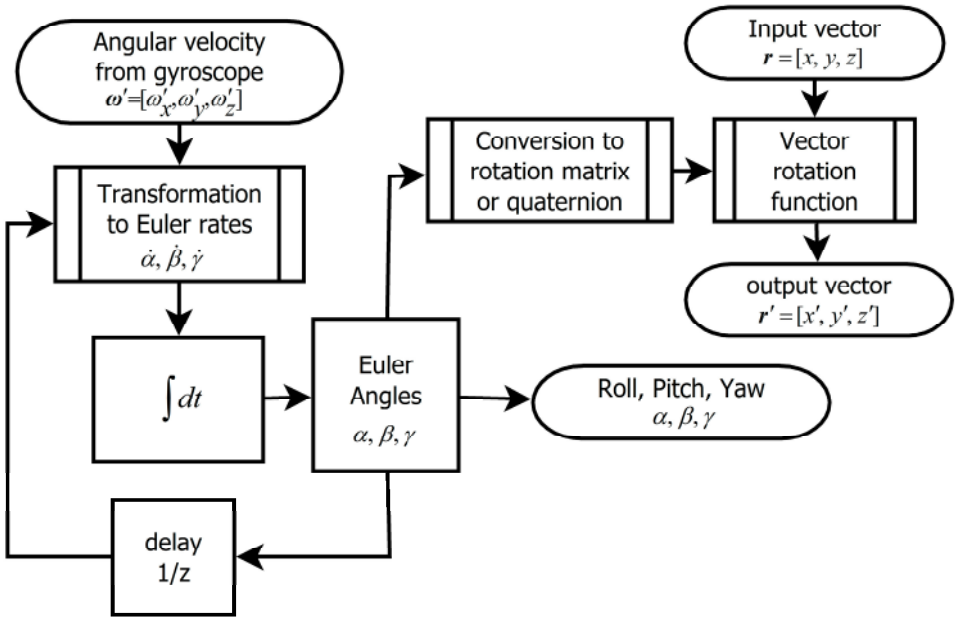


**Figure 8.** The algorithm based on Euler angle rates integration.

The main disadvantage of this algorithm is singularity of expression Equation (34) in case of $\cos\beta = 0$ called gimbal-lock, which is representing the state, when $x$-axis is pointing downwards or upwards ($\beta = 90°$ or $\beta = -90°$ respectively). In surroundings of this singularity numerical error is rising. In case that position reaches this singularity, information about two DoF is lost (see Figure 10).

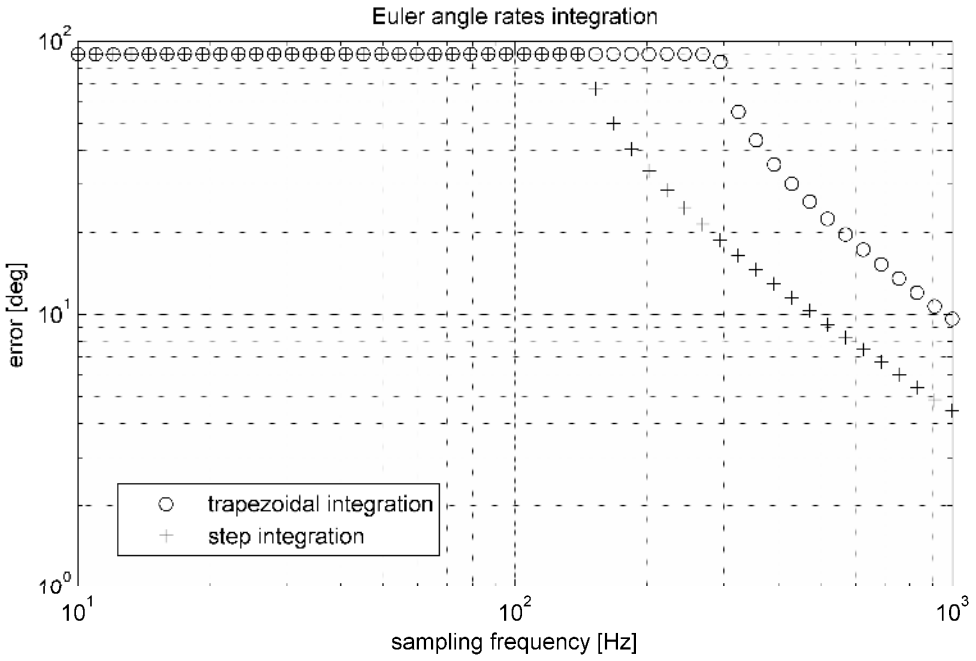


**Figure 9.** Relation between error and sampling frequency in the algorithm based on the integration of the Euler angle rates.

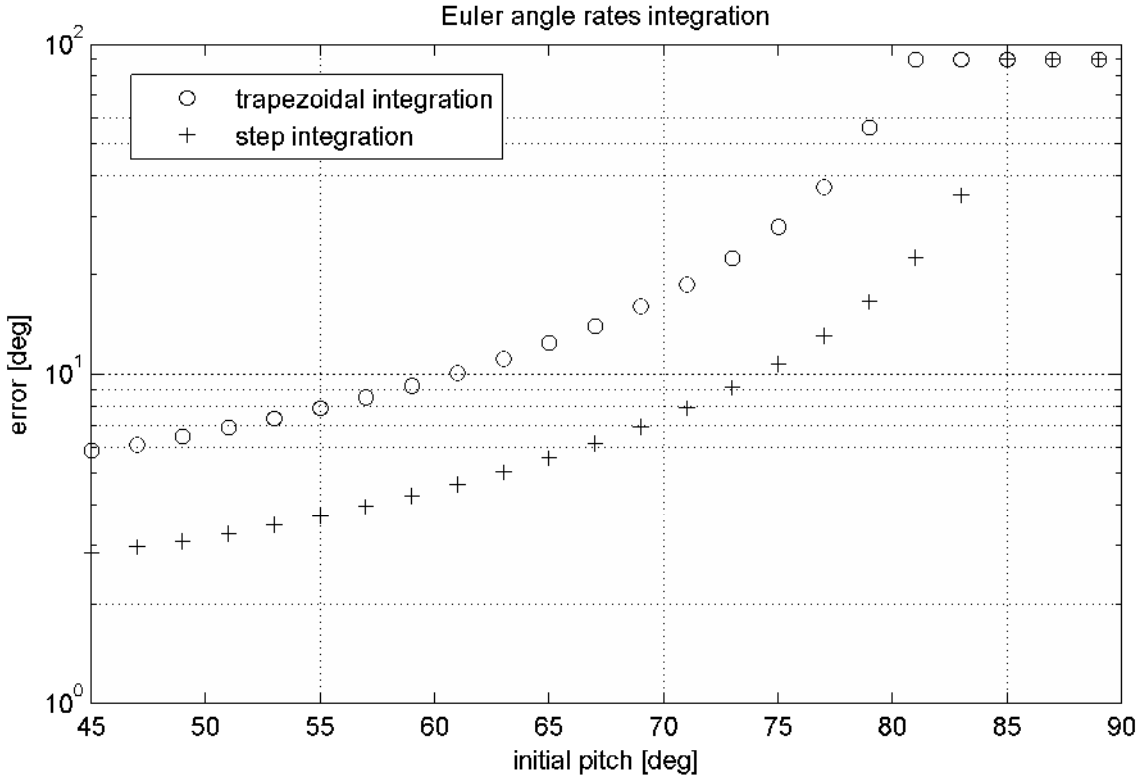


**Figure 10.** Relation between error and the initial pitch angle $\beta_0$ at $f_{\text{sample}}$ = 1000 Hz of the Euler angle rates-based algorithm.

This error can be avoided by early conversion to another Euler convention which reaches singularity in other points (for example conversion to 1-2-1, 1-3-1, 2-3-1, 3-1-2, 3-1-3 or 3-2-3 Euler angle convention [18]). After calculation of Euler angles in substitute convention, they are transformed back to the primary convention. Accuracy is then achieved in the whole angle range. This is computation demanding non-linear operation [18].

### *2.3. The Algorithm Based on Quaternion*

The third possibility is to utilize primary expression of rotation using quaternion. The principle is expressed by Figure 11. Similarly as in the case of the rotational matrix, two variants of calculation are possible.

#### 2.3.1. Precise Version

It is an analogy of the precise matrix-based algorithm. The form of update quaternion is following:

$$\begin{aligned}\mathbf{q}_{\text{update}} &= \left(n'_x\mathbf{i}+n'_y\mathbf{j}+n'_z\mathbf{k}\right)\sin\left(\frac{\omega'\Delta T}{2}\right)+\cos\left(\frac{\omega'\Delta T}{2}\right)\\ &= \frac{\left(\omega'_x\mathbf{i}+\omega'_y\mathbf{j}+\omega'_z\mathbf{k}\right)}{\omega'}\sin\left(\frac{\omega'\Delta T}{2}\right)+\cos\left(\frac{\omega'\Delta T}{2}\right)\end{aligned} \tag{37}$$

where $\omega' = |\boldsymbol{\omega}'| = \sqrt{\omega'^2_x + \omega'^2_y + \omega'^2_z}$ .

Then it is valid:

$$\boldsymbol{q}_n = \boldsymbol{q}_{update} \cdot \boldsymbol{q}_{n-1} \tag{38}$$

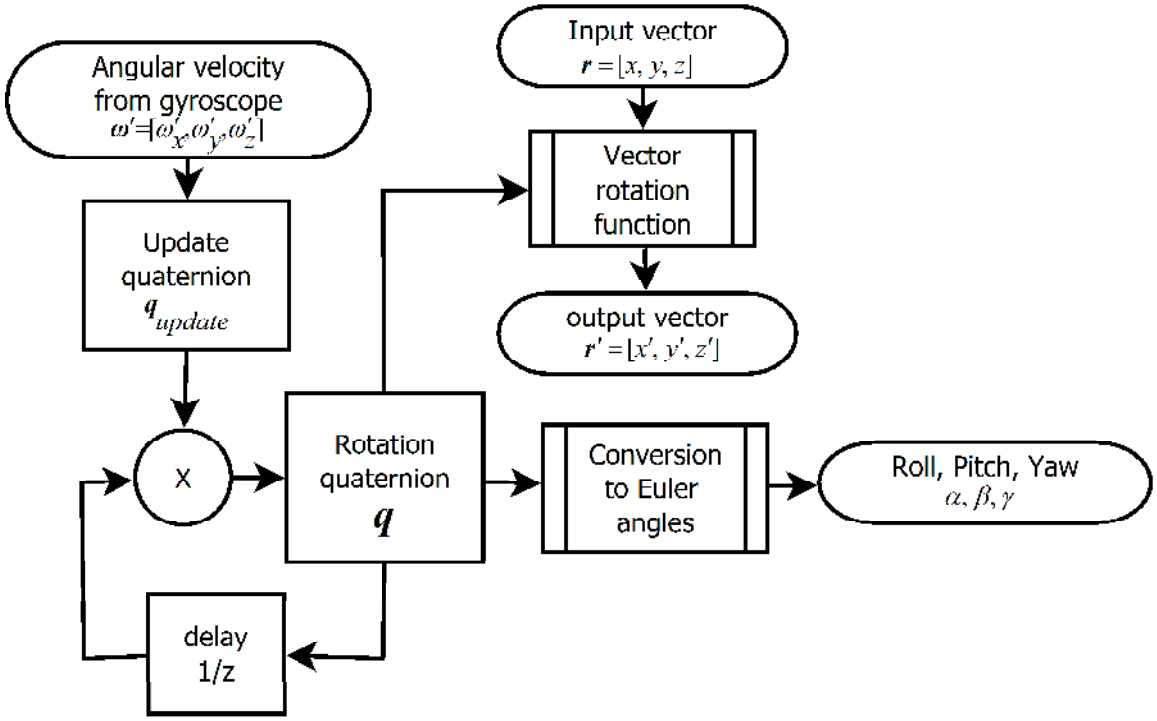


**Figure 11.** Principle of the quaternion-based algorithm.

2.3.2. Fast Version

Neglecting higher order members, using approximations:

$$\frac{\omega_i'}{\omega'}\sin\left(\frac{\omega'\Delta T}{2}\right) \approx \frac{\omega'\Delta T_i}{2} \qquad \cos\left(\frac{\omega'\Delta T}{2}\right) \approx 1 \tag{39}$$

We obtain update quaternion in the form:

$$\boldsymbol{q}_{\text{update}} = \left(\frac{\omega_x' dt}{2}\boldsymbol{i} + \frac{\omega_y' dt}{2}\boldsymbol{j} + \frac{\omega_z' dt}{2}\boldsymbol{k} + 1\right) \tag{40}$$

Then according to Equation (38) it is valid:

$$\begin{aligned} \boldsymbol{q}_2 &= \boldsymbol{q}_{update} \cdot \boldsymbol{q}_1 \\ &\approx \left(\frac{\omega_x' dt}{2}\boldsymbol{i} + \frac{\omega_y' dt}{2}\boldsymbol{j} + \frac{\omega_z' dt}{2}\boldsymbol{k} + 1\right)\boldsymbol{q}_1 \\ &\approx \boldsymbol{q}_1 + \frac{dt}{2}\left(\omega_x'\boldsymbol{i} + \omega_y'\boldsymbol{j} + \omega_z'\boldsymbol{k}\right)\boldsymbol{q}_1 = \boldsymbol{q}_1 + d\boldsymbol{q} \end{aligned} \tag{41}$$

which results in:

$$\frac{d\boldsymbol{q}}{dt} = \frac{1}{2}\left(\omega_x'\boldsymbol{i} + \omega_y'\boldsymbol{j} + \omega_z'\boldsymbol{k}\right)\boldsymbol{q} \tag{42}$$

By integration of quaternion derivation by time we get resulting rotation quaternion. Figure 12 compares precision of fast and precise versions of the algorithm. Like the fast matrix-based algorithm also the fast quaternion-based algorithm requires normalization of quaternion after each step. Normalization of rotation quaternion is described by Equation (17). Presented results are obtained with normalization.

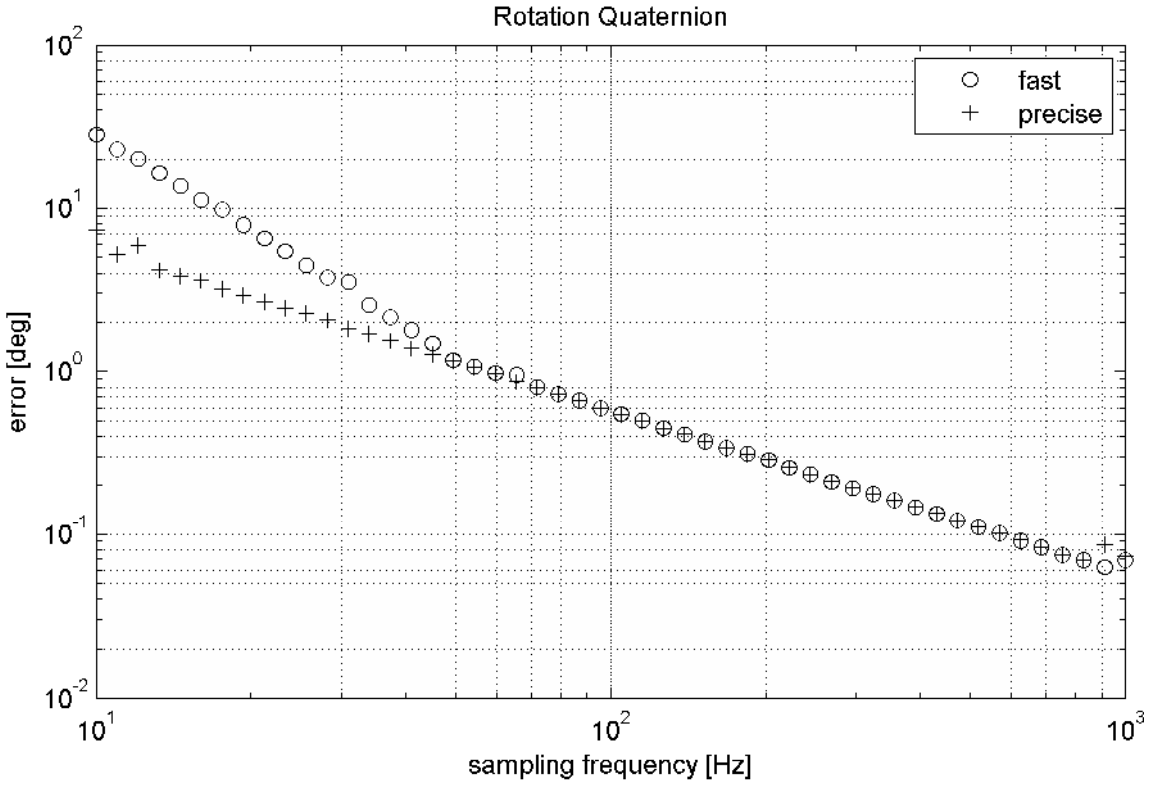


**Figure 12.** Errors of the quaternion-based algorithms during simulated movement.

*2.4. Compensation of MEMS Gyroscope Data Using a MEMS Accelerometer and Magnetic Compass*

Results given above are valid in an ideal case when gyroscope data are absolutely precise. Real MEMS gyroscope readings are noisy and sensitive to vibrations. The greatest impact on precision of Euler angles estimation has offset of the gyroscope. Due to variance of parameters of an electro-mechanical system with temperature the offset is also temperature dependent. The aim is to use secondary sensor (accelerometer, magnetic compass) to compensate increasing (offset-caused) error of the gyroscope-only system.

The accelerometer is sensing its acceleration (3D vector) relative to inertial frame of reference. In gravitational field the accelerometer is sensing gravity as acceleration upwards. Reading of the accelerometer is (see Figures 13 and 14):

$$\boldsymbol{a}_{\text{acc}} = \boldsymbol{a}' - \boldsymbol{g}' + \boldsymbol{a}_{\text{noise}} = [a_{\text{acc}X}, a_{\text{acc}Y}, a_{\text{acc}Z}] \tag{43}$$

where $\boldsymbol{a}'$ is own acceleration of the object expressed in the coordinate system $S'$, $\boldsymbol{g}'$ is a vector of gravitational acceleration (depending on locality near Earth) transformed to the coordinate system of the object $S'$ based on data concerning object rotation and $\boldsymbol{a}_{\text{noise}}$ is the noise caused by:

- Vibrations of this object
- Thermal noise of the sensor
- Quantization noise of the A/D converter

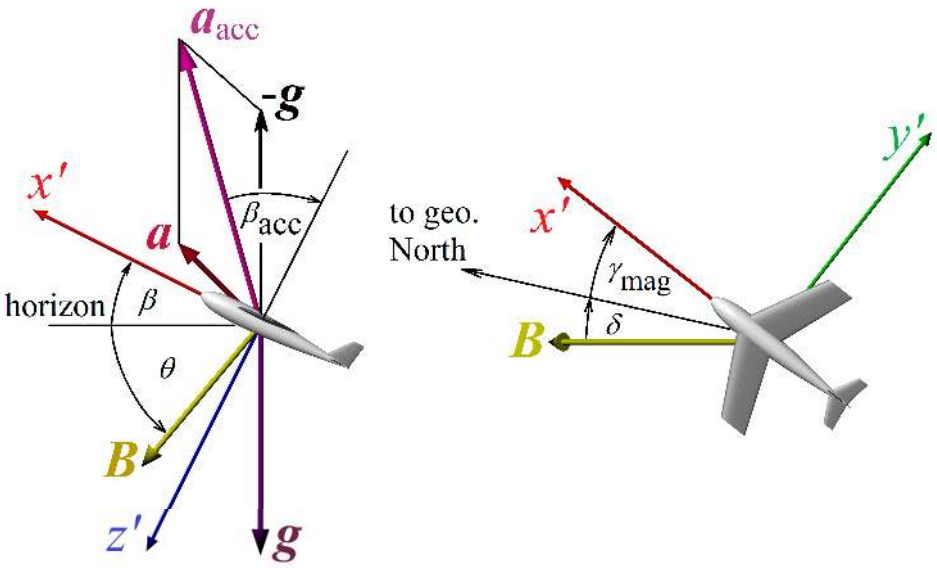


**Figure 13.** Accelerometer and magnetic compass readings at non-zero pitch $\beta$ and yaw $\gamma$. Acceleration $\boldsymbol{a}_{acc}$ is measured by the on-board accelerometer as a sum of the gravity acceleration $\boldsymbol{g}$ and object's acceleration $\boldsymbol{a}$. Earth's magnetic field induction $\boldsymbol{B}$ has inclination $\theta$, declination $\delta$ and its horizontal complement points to magnetic North.

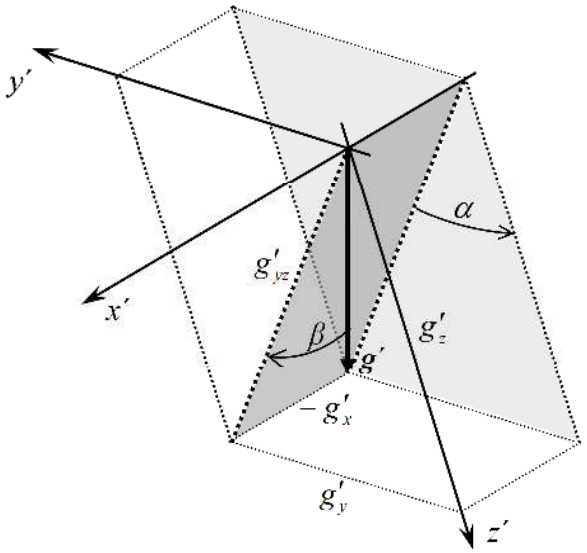


**Figure 14.** Roll and pitch calculation from measured gravity acceleration. Object pitches up and rolls right (axis $x'$ points forward). The vector $\boldsymbol{g}'$ defines vertical direction.

According to Figure 14 for roll and pitch angle we obtain:

$$\alpha_{acc} = \text{atan2}(g'_y, g'_z) \approx \text{atan2}(-a_{accY}, -a_{accZ}) \tag{44}$$

$$\beta_{acc} = \text{atan2}(-g'_x, g'_{yz}) = \text{atan2}(-g'_x, \sqrt{g'^2_y + g'^2_z}) \approx \text{atan2}(a_{accX}, \sqrt{a_{accY}^2 + a_{accZ}^2}) \tag{45}$$

If we assume that noise $\boldsymbol{a}_{\text{noise}}$ has zero mean value and lower limiting frequency $f_{\text{min}}$, then the noise can be effectively suppressed by the low pass filter.

Since we cannot determine the rotation around vertical $z'$-axis (yaw $\gamma$) from accelerometer data, it is necessary to add a magnetic compass to the sensor system. For ensuring proper function of the system for all rotations of the object, the magnetic sensor has to determine magnetic induction $\boldsymbol{B}'$ of the Earth's magnetic field in all three axes (compass output is the vector $\boldsymbol{B}' = [B'_x, B'_y, B'_z]$). For yaw rotation calculated from readings of the magnetic sensor it is valid:

$$\gamma_{\text{mag}} = \text{atan2}(-B_{y1}, B_{x1}) - \delta_m \tag{46}$$

where $\delta_{\text{m}}$ is magnetic declination (offset between magnetic and geographic north direction, depending on actual position on Earth), $B_{x1}$ and $B_{y1}$ are components of measured magnetic induction after transformation to the coordinate system $S_1$ (inverted $x$- and following $y$- rotation, see definition of Euler angles) according to the formula:

$$\boldsymbol{B}_1 = [B_{x1}, B_{y1}, B_{z1}] = \Re^{-1}_{\alpha,\beta}(\boldsymbol{B}') \tag{47}$$

In terms of avoiding preparation of partial inverse rotation, it is more convenient to determine the difference between yaw $\gamma_{\text{gyro}}$ calculated from gyroscope data and yaw from the magnetic compass $\gamma_{\text{mag}}$ as:

$$\Delta\gamma_{\text{mag}} = \gamma_{\text{mag}} - \gamma_{\text{gyro}} = \text{atan2}(-B_y, B_x) - \delta_m \tag{48}$$

where $B_x$ a $B_y$ are components of measured magnetic induction after transformation to the coordinate system $S$ (inverted $x$-, $y$- and $z$- rotation), which are:

$$\boldsymbol{B} = [B_x, B_y, B_z] = \Re^{-1}_{\alpha,\beta,\gamma}(\boldsymbol{B}') \tag{49}$$

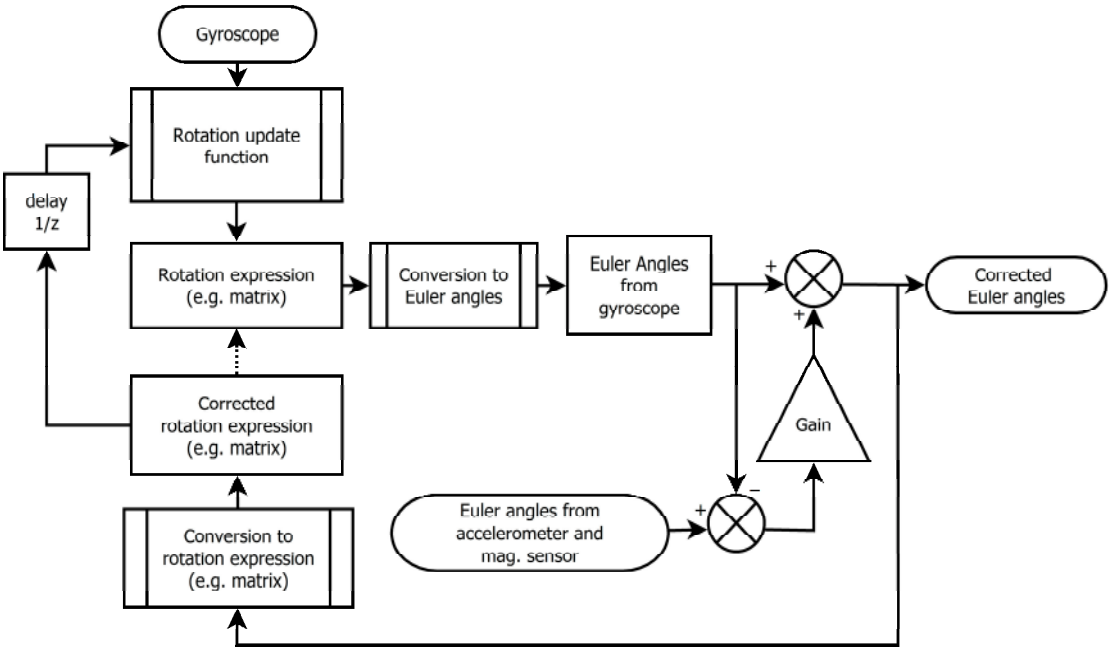


**Figure 15.** Data fusion of the gyroscope, accelerometer and magnetic sensor.

For fusion of Euler angles measured by the gyroscope as a primary sensor and accelerometer and magnetic compass as secondary sensors we can use the algorithm as shown in Figure 15. Gain $K << 1$

expresses relative weight of the accelerometer with respect to the gyroscope (if $K = 0$, the accelerometer does not affect output Euler angles). Delay block and gain forms the first order discrete low pass filter in the accelerometer signal path with cutoff frequency:

$$f_{\max} \approx K f_{\text{sample}} \tag{50}$$

The fusion schema does not filter out any noise from gyroscope reading; it suppresses increasing error of estimated Euler angles in long term caused mainly by offset.

While the schematics in Figure 15 contains the reverse conversion block from Euler angles to the rotation matrix, normalization of the matrix is no longer needed.

## 3. Results and Discussion

Effect of sensor fusion is more significant after longer time (especially at low angular velocities). Figure 16 shows effect of using fusion of gyroscope, accelerometer and magnetic sensor readings. Simulated rotation was slowed down 100-times ($A = 0.01$, compare with Equation (24)). Fusion gain was $K = 0.01$, noise in secondary sensor data has SNR = 0dB. The precise quaternion-based algorithm at sampling frequency 1 kHz was used. Due to gyroscope offset the estimation error continuously increases with time. The low pass filter within the data fusion algorithm suppresses noise in secondary sensor data and roll angle obtained by fusion slightly oscillates around actual roll.

As can be seen in Figure 17, sensor fusion with weak bound of secondary absolute but noisy sensor can effectively suppress error of estimation caused by sensor offsets. Fusion gain $K$ has to be set according to offset variance of the gyroscope (the more precise the gyroscope is the lesser fusion gain can be obtained).

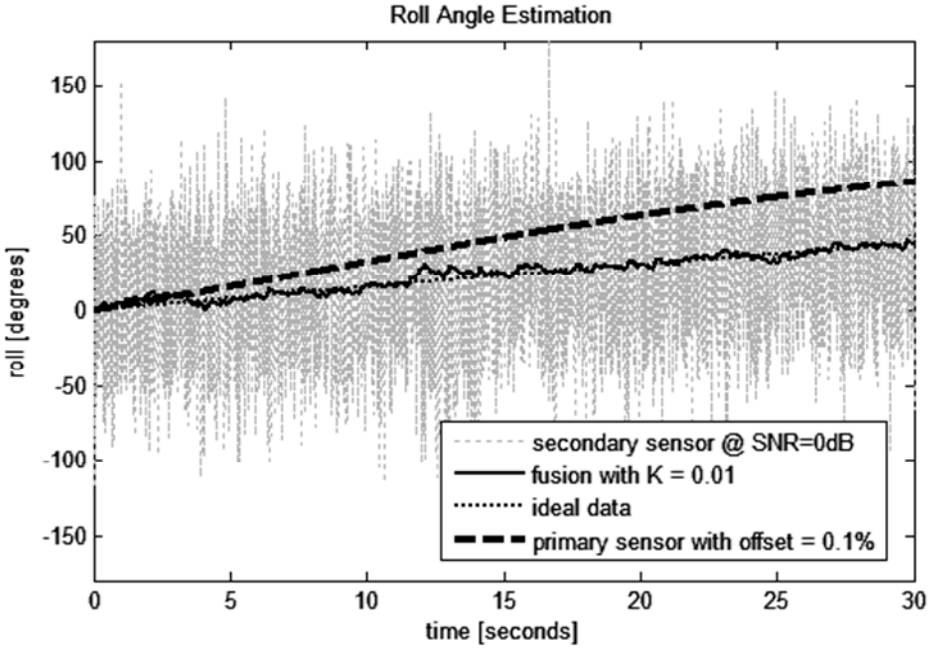


**Figure 16.** Estimation of roll angle with gyroscope offset 0.1% of full range (500°/s).

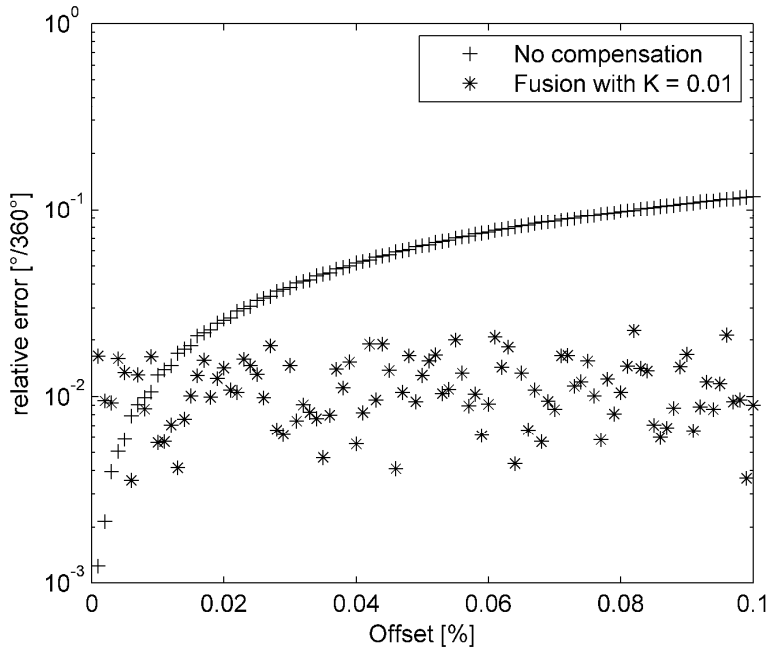


**Figure 17.** Relative error of roll angle estimation with respect to gyroscope offset.

The second great aspect of the algorithm is its computational time. Two types of reference hardware were used:

- 8-bit low-cost microprocessor (Atmel ATmega1284P running at 20 MHz);
- 32-bit microprocessor with FPU and DSP support (Atmel UC3C1512C running at 48 MHz).

Table 1 compares computational time of algorithms in terms of the CPU cycles of 8-bit low-cost microprocessor. The mentioned cycle counts are average values from 1000 random inputs, using the mathematical library optimized for AVR 8-bit microcontrollers. Algorithms are using software-implemented single precision floating point arithmetic (according to IEEE 754) due to the fact that AVR microcontrollers do not contain the floating point unit (FPU). Using highly optimized implementation of the matrix-based algorithm including fusion of the gyroscope with accelerometer and magnetic compass allows algorithm sampling rate up to approximately 200 Hz (running on AVR 8-bit core @ 20 MHz).

Table 2 shows the same algorithms running on the 32-bit microprocessor. Utilization of the 32-bit microcontroller with FPU significantly reduces the count of needed clock cycles (in case of adding and multiplication of real numbers approx. 30 times depending on the used processor). While the representation of numbers is the same for all architectures (32-bit floating point number), accuracy of the algorithm does not depend directly on the used microcontroller. However, decreasing time needed for one cycle of the algorithm allows higher maximal sample rate (up to maximal sample rate of gyroscope itself). Increasing sample rate will improve accuracy significantly. For example, increasing sampling rate from 200 Hz to 1 kHz will decrease error caused by the algorithm by approx. 50% (see Figures 5, 7 and 10).

**Table 1.** Comparison of methods in terms of 8-bit AVR processor clock cycles.

| Algorithm | Updating of the Rotational Matrix | Integration of the Euler Angle Rates | Updating of the Quaternion |
|---|---|---|---|
| Redundancy (count of variables) | ** 9 | **** 3 | *** 4 |
| Gyroscope data processing (rotation update) | *** + 17,230 (6034 +) | *** 14,750 | **** 11,462 (5120 +) |
| Normalization | ** 12,265 | ***** 0 | **** 1972 |
| Vector transformation | **** 2301 | * 15,231 [3]) | *** 4321 |
| Transformation to the rotational matrix | ***** 0 | ** 12,930 | **** 3536 |
| Transformation to Euler angles | **** 7820 | ***** 0 | *** 10,673 |
| Transformation to quaternion | *** 3370 | ** 13,020 | ***** 0 |
| Clock cycles for the gyroscope-only system [1]) | 37,315 (26,119 +) | 14,750 | 24,107 (17,765 +) |
| Clock cycles for the compensated system [2]) | 40,281 (29,085 +) | 29,981 [3]) | 39,476 (33,134 +) |

Legend: + Fast version of the algorithm; * Improper; ** Usable; *** Good; **** Excellent; ***** No demands on computing time; [1]) Cycles needed for gyroscope data processing, normalization (if needed) and conversion to Euler angles; [2]) Cycles needed for gyroscope data processing, transformation of the magnetic induction vector (compass), conversion to Euler angles and back; [3]) Euler angles were converted to the rotational matrix which was used for vector transformation.

**Table 2.** Comparison of methods in terms of 32-bit UC3C processor clock cycles.

| Algorithm | Updating of the Rotational Matrix | Integration of the Euler Angle Rates | Updating of the Quaternion |
|---|---|---|---|
| Gyroscope data processing (rotation update) | *** 4511 (247 +) | ** 10,900 | **** 4219 (169 +) |
| Normalization | ** 226 | ***** 0 | **** 66 |
| Vector transformation | **** 58 | * 13,757 [3]) | *** 182 |
| Transformation to the rotational matrix | ***** 0 | ** 13,669 | **** 752 |
| Transformation to Euler angles | **** 10,014 | ***** 0 | *** 10,648 |
| Transformation to quaternion | **** 1049 | ** 11,337 | ***** 0 |
| Clock cycles for the gyroscope-only system [1]) | 14,751 (10487 +) | 10,900 | 14,933 (10,883 +) |
| Clock cycles for the compensated system [2]) | 28,252 (23988 +) | 24,657 [3]) | 26,386 (22,336 +) |

Legend: + Fast version of the algorithm; * Improper; ** Usable; *** Good; **** Excellent; ***** No demands on computing time; [1]) Cycles needed for gyroscope data processing, normalization (if needed) and conversion to Euler angles; [2]) Cycles needed for gyroscope data processing, transformation of the magnetic induction vector (compass), conversion to Euler angles and back; [3]) Euler angles were converted to the rotational matrix which was used for vector transformation.

If the microcontroller with hardware support of floating-point calculations is used, linear (fast) versions of algorithms are much faster than the precise non-linear algorithms. Results given in Tables 1 and 2 strongly depend on implementation of the discussed algorithms (execution speed can be improved by using optimized mathematical libraries for hardware supporting floating-point calculations). Number of the clock ticks is shown mainly for simple comparison purposes.

The Euler angle rates integration can be faster than the remaining two algorithms but it has significantly worse accuracy at the same sampling frequency and also has intrinsic singularity. Therefore the choice should be between the matrix- and quaternion-based algorithms. If the sensor system should be able to quickly transform many vectors between inertial and local frame of reference the matrix-based algorithm can be a better choice (2–3 times faster vector transformation than by quaternion).

## 4. Conclusions

By comparing relative errors of each mentioned algorithm we can see that the worst algorithm is direct Euler angles integration due to its singularity. Precise version of the quaternion-based algorithm is slightly faster than the precise matrix-based algorithm. Fast version of the quaternion-based algorithm at lower sampling frequency is also more accurate than the matrix-based algorithm (see Table 3). Difference in accuracy between fast and precise versions of the same algorithm decreases with sampling frequency (see Figures 7 and 12). The choice of the proper algorithm depends on:

- Available computational power (CPU) and maximal sampling frequency of the sensors (which reflects in overall cost of the sensor system and its accuracy). At lower sampling frequency the fast quaternion-based algorithm is more precise than the fast matrix-based algorithm.
- Precision requirements (in order to achieve long-term stability the compensated system with sensor fusion has to be used).
- Amount of vectors transformed from the non-rotated coordinate system to rotated coordinates and vice versa (transformation performed by the matrix is faster).

**Table 3.** Accuracy of the algorithms.

| Sampling Frequency | Maximal Error of the Algorithm during 120 s of Simulated Movement | | | | |
|---|---|---|---|---|---|
| | Matrix-Based Algorithm | | Integration of Euler Angle Rates | Quaternion-Based Algorithm | |
| | Fast | Precise | Step Integration | Fast | Precise |
| 10 Hz | >180° | 8° | >180° | 30° | 8° |
| 50 Hz | 4° | 1° | >180° | 1° | 1° |
| 100 Hz | 1° | 0.6° | >180° | 0.6° | 0.6° |
| 500 Hz | 0.1° | 0.1° | 8° | 0.1° | 0.1° |
| 1000 Hz | 0.06° | 0.06° | 4° | 0.06° | 0.06° |

## Acknowledgments

The paper was elaborated with support of the Slovak grant agency KEGA, grant No. 010ŽU-4/2013 Modernization of didactic equipment and teaching methods with a focus on the area of robotics.

## Author Contributions

The work presented in this paper was carried out in collaboration with all authors. Aleš Janota has defined the research topic, guided the research goals, edited and reviewed the paper. Vojtech Šimák conceived and designed the experiments. Dušan Nemec performed the experiments and wrote the paper. Jozef Hrbček contributed with compensation of MEMS gyroscope data.

## Conflicts of Interest

The authors declare no conflict of interest.

## References

1. Park, M.; Gao, Y. Error and Performance Analysis of MEMS-Based Inertial Sensors with a Low-Cost GPS Receiver. *Sensors* **2008**, *8*, 2240–2261.
2. Catalfamo, P.; Ghoussayni, S.; Evins, D. Gait Event Detection on Level Ground and Incline Walking Using a Rate Gyroscope. *Sensors* **2010**, *10*, 5683–5702.
3. Zeng, H.; Zhao, Y. Sensing Movement: Microsensors for Body Motion Measurement. *Sensors* **2011**, *11*, 638–660.
4. Formento, P.C.; Acevedo, R.; Ghoussayni, S.; Ewins, D. Gait Event Detection during Stair Walking Using a Rate Gyroscope. *Sensors* **2014**, *14*, 5470–5485.
5. Shoaib, M.; Bosch, S.; Incel, O.D.; Scholten, H. Fusion of Smartphone Motion Sensors for Physical Activity Recognition. *Sensors* **2014**, *14*, 10145–10176.
6. Zhang, S.; Ang, M.H., Jr.; Xiao, W.; Tham, C.K. Detection of Activities by Wireless Sensors for Daily Life Surveillance: Eating and Drinking. *Sensors* **2009**, *9*, 1499–1517.
7. Tawk, Y.; Tomé, P.; Botteron, C.; Stebler, Y.; Farine, P.-A. Implementation and Performance of a GPS/INS Tightly Coupled Assisted PLL Architecture Using MEMS Inertial Sensors. *Sensors* **2014**, *14*, 3768–3796.
8. Si, Ch.; Han, G.; Ning, J.; Yang, F. Bandwidth Optimization Design of a Multi Degree of Freedom MEMS Gyroscope. *Sensors* **2013**, *13*, 10550–10560.
9. Bhatt, D.; Aggarwal, P.; Bhattacharya, P.; Devabhaktuni, V. An Enhanced MEMS Error Modeling Approach Based on Nu-Support Vector Regression. *Sensors* **2012**, *12*, 9448–9466.
10. Quinchia, A.G.; Falco, G.; Falletti, E; Dovis, F.; Ferrer, C. A Comparison between Different Error Modeling of MEMS Applied to GPS/INS Integrated Systems. *Sensors* **2013**, *13*, 9549–9588.
11. Robert, P.L. Mechanical-Thermal Noise in MEMS Gyroscopes. *IEEE Sens. J.* **2005**, *5*, 493–500.
12. Yang, B.; Wang, S.; Li, H.; Zhou, B. Mechanical-Thermal Noise in Drive-Mode of a Silicon Micro-Gyroscope. *Sensors* **2009**, *9*, 3357–3375.
13. Stančin, S.; Tomažič, S. Time- and Computation-Efficient Calibration of MEMS 3D Accelerometers and Gyroscopes. *Sensors* **2014**, *14*, 14885–14915.
14. Xia, D; Yu, Ch.; Kong, L. The Development of Micromachined Gyroscope Structure and Circuitry Technology. *Sensors* **2014**, *14*, 1394–1473.

15. Šimák,V.; Nemec, D.; Hrbček, J.; Janota, A. Inertial navigation: Improving precision and speed of Euler angles computing from MEMS gyroscope data. In *Transport Systems Telematics 2013, Communications in Computer and Information Science*; Springer-Verlag Berlin Heidelberg 2013; Volume 395, pp. 163–170.
16. Premerlani, W.; Bizard, P. Direction Cosine Matrix IMU: Theory, Available online: http://gentlenav.googlecode.com/files/DCMDraft2.pdf (accessed on 21 December 2014).
17. Diebel, J. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. Stanford University, Stanford, CA, USA, 2006. Available online: http://www.astro.rug.nl/software/kapteyn/_downloads/attitude.pdf (accessed on 21 December 2014).
18. Singla, P.; Mortari, D.; Junkins, J.L. How to Avoid Singularity When Using Euler Angles? Available online: http://lairs.eng.buffalo.edu/pdffiles/pconf/C10.pdf (accessed on 21 December 2014).