THE DIRECTION COSINE MATRIX ALGORITHM IN FIXED-POINT: IMPLEMENTATION AND ANALYSIS

Alexandre Meirhaeghe, Jani Boutellier, and Jussi Collin

Faculty of Information Technology and Communication Sciences, Tampere University, Finland

ABSTRACT

Inertial navigation allows tracking and updating the position and orientation of a moving object based on accelerometer and gyroscope data without external positioning aid, such as GPS. Therefore, inertial navigation is an essential technique for, e.g., indoor positioning. As inertial navigation is based on integration of acceleration vector components, computation errors accumulate and make the position and orientation estimate drift. Even though maximum computation precision is desired, also efficiency needs consideration in the age of Internet-of-Things, to enable deployment of inertial navigation based applications to the smallest devices. This work formulates the Direction Cosine Matrix update algorithm, a central component for inertial navigation, in fixed-point and analyzes its precision and computation load compared to a regular floating-point implementation. The results show that the fixed-point version maintains very high precision, while requiring no floating point hardware for operation. The paper presents execution time results on three very different embedded processors.

***Index Terms*—** inertial navigation, mobile device, energy efficiency, fixed-point

1. INTRODUCTION

Nowadays GPS positioning has become mainstream and is available in many mobile consumer devices for reasonably accurate outdoor positioning. However, inside buildings, underground, and when occluded by large obstacles, the GPS signal is either not available at all, or offers only limited precision. In such places, other positioning techniques such as *inertial navigation* are however available.

Inertial navigation is based on accelerometer and gyroscope sensor (also known as inertial measurement unit, IMU) signals that are sampled at a frequency in the range of 10^2 ... 10^3 Hz. The IMUs contain typically three gyroscope vectors and three accelerometer vectors for three orthogonal axes, corresponding to angular velocity and linear acceleration. For each new set of samples, the orientation and position estimate is updated with respect to a known starting point, based on *strapdown inertial navigation* algorithms [1].

Compared to the processing power of laptop and smartphone processors, the required signal processing computations are not very demanding, despite their real-time requirements. However, with the recent Internet of Things trend, various smart objects with very little processing power can become the processing platform for inertial navigation [2]; moreover, such tiny microcontrollers are rarely equipped with floating-point computing hardware.

In this paper, a fixed-point formulation of a central signal processing algorithm for inertial navigation, updating of the Direction Cosine Matrix (DCM) is presented. The fixed-point version allows performing the computations using regular integer operations supported by every programmable processor. In addition to the fixed-point formulation, a variety of precision-increasing optimizations are presented and applied. Finally, the paper provides measurements that illustrate the processing time requirements of the fixed-point DCM implementations for three embedded processors.

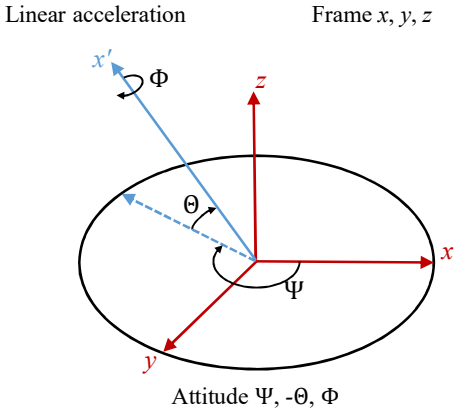

Fig. 1. Representation of the coordinate frame, the linear acceleration and the attitude.

The organization of the paper is as follows: Sections 2 and 3 explain the theoretical premises and previous work related to the paper; Section 4 presents our fixed-point formulation of the DCM update algorithm; Section 5 shows accuracy and execution time measurement results, and Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

To understand the existing research, the concepts of *coordinate frame* and *attitude* need explaination: a coordinate frame is an analytical abstraction defined by three unit vectors that are perpendicular to one another, often numbered consecutively. The frame can be visualized as a set of three perpendicular axes passing through a common point (*origin*) with the unit vectors starting from the origin along the axes. The attitude of an object is the orientation of an object in space. It can be represented as three angles of rotation (Euler angles) around the three axes of the frame as shown on the Fig. 1.

The inertial navigation algorithms discussed in this work have originally been presented in the seminal work of P. Savage [1]. As presented in [1], attitude parameters for representing the angular relationship between two coordinate frames are the direction cosine matrix $C_{A_2}^{A_1}$ and the rotation vector $\mathbf{p}$. The direction cosine matrix

transforms a vector from the reference frame A_2 to frame A_1:

$$\mathrm{C}_{\mathrm{A}_2}^{\mathrm{A}_1}\mathbf{v}^{\mathrm{A}_2} = \mathbf{v}^{\mathrm{A}_1}.$$

The rotation vector $\mathbf{p}$ defines an axis of rotation and its magnitude defines the value of rotation. Similarly as with the direction cosine matrix, the rotation vector can be used to define the attitude between frames A_2 and A_1. If frame A_1 is rotated about the rotation vector $\mathbf{p}$ through the angle $p = \sqrt{\mathbf{p}^T\mathbf{p}}$, the new attitude can be uniquely used to define frame A_2. The relationship between the direction cosine matrix and the rotation vector can be used to transform any rotation vector to uniquely define the direction cosine matrix:

$$\mathrm{C}_{\mathrm{A}_2}^{\mathrm{A}_1}(\mathbf{p}) = \begin{cases} \mathrm{I} + \frac{\sin(p)}{p}(\mathbf{p}\times) + \frac{1-\cos(p)}{p^2}(\mathbf{p}\times)(\mathbf{p}\times) & \text{if } p \neq 0 \\ \mathrm{I} & \text{otherwise,} \end{cases}$$

where $\mathbf{p}\times$ is a matrix composed from values x, y and z of p.

Commonly used Euler angles are not good in computing kinematic orientations of object [3]. To specify the attitude of an object using known Euler angles *roll* Φ, *pitch* Θ and *yaw* Ψ (presented in the Fig. 1), direction cosine matrix can be converted as follows [4].

In a spin at yaw, the following matrix can be used to pass from a coordinate frame to another:

$$\mathbf{C}_\Psi = \begin{pmatrix} \cos(\Psi) & \sin(\Psi) & 0 \\ -\sin(\Psi) & \cos(\Psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

In a spin at pitch, respectively:

$$\mathbf{C}_\Theta = \begin{pmatrix} \cos(\Theta) & 0 & -\sin(\Theta) \\ 0 & 1 & 0 \\ \sin(\Theta) & 0 & \cos(\Theta) \end{pmatrix} \quad (2)$$

And finally, in a spin at roll :

$$\mathbf{C}_\Phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\Phi) & \sin(\Phi) \\ 0 & -\sin(\Phi) & \cos(\Phi) \end{pmatrix} \quad (3)$$

Thereby the direction cosine matrix (DCM) can be obtained as: $\mathrm{C}_{\mathrm{A}_2}^{\mathrm{A}_1} = \mathbf{C}_\Psi * \mathbf{C}_\Theta * \mathbf{C}_\Phi$.

As explained in [1], the columns (and rows) of the DCM represent orthogonal unit vectors. These vectors need to be unity in magnitude and mutually orthogonal to one another. For that, in addition to the basic DCM update algorithm, *renormalization* and *orthogonalization* algorithms are commonly applied.

2.1. Related work

The number of previous works that analyze the DCM computation part of [1] are few, although there are some works [5] [6] that study the complete inertial navigation algorithm without addressing the optimization of individual algorithm components. [7] provides a concise description of the physical principles of inertial navigation, the associated growth of errors and their compensation. Another branch of works, [8], [2] and [9], study the *quaternion* presentation for attitude and position update instead of DCM. Quaternions give a mathematical notion for representing orientations and rotations of objects in three dimensions similar to rotation matrices and Euler angles. The work [10] presents an implemented inertial navigation system, but is not based on the DCM approach. Furthermore, [11] relies on GPS signals in addition to inertial navigation. Regarding fixed-point INS computations on microcontrollers, [2] presents a quaternion-based approach, where zero-velocity test statistics are computed in fixed-point on a 32-bit microcontroller. However, other parts of the algorithm are implemented in floating-point.

3. THE DIRECTION COSINE MATRIX ALGORITHM

Algorithm 1 Direction Cosine Matrix algorithm

1: $\mathbf{in} = \mathbf{gyrodata}\,(i,:)' * \mathbf{T}$
2: $\mathbf{C}_1 = \begin{pmatrix} \mathbf{0} & -\mathbf{in}_3 & \mathbf{in}_2 \\ \mathbf{in}_3 & \mathbf{0} & -\mathbf{in}_1 \\ -\mathbf{in}_2 & \mathbf{in}_1 & \mathbf{0} \end{pmatrix}$
3: $\mathbf{C}_2 = \mathbf{C}_1 * \mathbf{C}_1$
4: $\mathbf{p} = \sqrt{\mathbf{in}' * \mathbf{in}}$
5: $\mathbf{out} = \mathbf{I} + \frac{\sin(\mathbf{p})}{\mathbf{p}} * \mathbf{C}_1 + \frac{1-\cos(\mathbf{p})}{\mathbf{p}^2} * \mathbf{C}_2$
6: $\mathbf{DCM} = \mathbf{DCM} * \mathbf{out}$

The initial formulation of the DCM algorithm is presented in Algorithm 1, where T represents the sampling period, and I represents the identity matrix. The variable *in* contains sampled gyroscope data for x, y and z axes. The algorithm is executed for each new set of (x, y, z) samples for updating the DCM matrix. This behavior makes the algorithm recursive and causes propagation of estimation errors to the following samples.

Considering real-time implementation on a very low-resource device, the computation of cosine and sine functions are complex to implement and therefore, the Taylor approximation (from [1]) of the algorithm presented in Equation 4 is used. The approximation turns line 5 of Algorithm 1 into

$$\mathbf{out} = \mathbf{I} + \left(1 - \frac{\mathbf{p}_2}{6} + \frac{\mathbf{p}_4}{120}\right) * \mathbf{C}_1 + \left(\frac{1}{2} - \frac{\mathbf{p}_2}{24} + \frac{\mathbf{p}_4}{720}\right) * \mathbf{C}_2 \quad (4)$$

where $\mathbf{p}_2 = \mathbf{in}' * \mathbf{in}$ and $\mathbf{p}_4 = \mathbf{p}_2 * \mathbf{p}_2$. Irrespective of whether Algorithm 1 is implemented by trigonometric functions or by Taylor approximation, its accuracy can be further improved by introducing renormalization of the DCM matrix as follows:

$$\mathbf{DCM} = \mathbf{DCM} + \frac{1}{2} * (\mathbf{I} - \mathbf{DCM} * \mathbf{DCM}') * \mathbf{DCM}.$$

This renormalization needs to be done just after the line 6 of Algorithm 1 and needs to be computed for each new set of (x, y, z) samples.

Going beyond the work of Savage [1], this paper presents a fixed-point formulation of Algorithm 1 based on the Taylor approximation, and renormalization. The key question in such fixed-point formulations, compared to straightforward floating-point, is to ensure that computation precision is maintained and overflow is avoided.

3.1. Transformation to fixed-point

A fixed-point number is an integer number, which corresponds to a floating-point number. As the name says, the fixed-point representation assumes a fixed number of bits before and after the radix point. This represents a trade-off between dynamic range and precision, dictated by the place of the radix point.

In order to create high-quality fixed-point implementations, the dynamic ranges of variables need to be determined. Computing of

the dynamic range can be performed either by simulation or by interval arithmetic [12]. Then, with the known dynamic range, it is possible to calculate the number of bits for the *integer part* (left side of the radix point) using the following equation:

$$\mathbf{Mx} = \mathbf{floor}(\mathbf{max}(\mathbf{abs}(\mathbf{x}))) + \mathbf{s} + \mathbf{1},$$

where $\mathbf{s} = 0$ if $\mathbf{x}$ is unsigned, and 1 otherwise, and $\mathbf{Mx}$ is the number of bits for the integer part. With the total number of bits and the number of bits for the integer part, it is possible to deduce the number of bits for the fractional part. After that, the floating point input data needs to be scaled by the number of bits of the fractional part of the data.

4. PROPOSED WORK

The proposed fixed point formulation of the DCM algorithm is presented in Algorithm 2. In this algorithm, every variable is represented with 32 bits. The right-hand side of the algorithm description shows the fixed-point format of each intermediate result in the standard *Qx* notation, where number on the left side of the radix point signifies the count of integer bits, and the number of the right side of the radix point the count of the fraction bits. Lower wordlengths (8, 16) were not considered as in inertial navigation very high precision is required.

In order to determine the position of the radix point, the scale of the input data needs to be fixed. In our case, the scale was determined by the maximum signal amplitude that the sensors were able to provide, which was -20.0 to 20.0. To represent this signal scale, 6 integer bits (including sign bit) are required, which leaves 26 bits for the fractional part. Below, the numeric precision of each intermediate result is presented in detail.

Algorithm 2 Direction Cosine Matrix algorithm in fixed point

1: $\mathbf{in} = \mathbf{gyrodata}\,(i,:)'$ **Q6.26**

2: $\mathbf{C}_1 = \begin{pmatrix} \mathbf{0} & -\mathbf{in}_3 & \mathbf{in}_2 \\ \mathbf{in}_3 & \mathbf{0} & -\mathbf{in}_1 \\ -\mathbf{in}_2 & \mathbf{in}_1 & \mathbf{0} \end{pmatrix}$ **Q6.26**

3: $\mathbf{C}_2 = \begin{pmatrix} \mathbf{C}_{11} * \mathbf{C}_{13} & & \\ +\mathbf{C}_{12} * \mathbf{C}_{16} & \mathbf{C}_{12} * \mathbf{C}_{17} & \mathbf{C}_{11} * \mathbf{C}_{15} \\ & \mathbf{C}_{13} * \mathbf{C}_{11} & \\ \mathbf{C}_{15} * \mathbf{C}_{16} & +\mathbf{C}_{15} * \mathbf{C}_{17} & \mathbf{C}_{13} * \mathbf{C}_{12} \\ & & \mathbf{C}_{16} * \mathbf{C}_{12} \\ \mathbf{C}_{17} * \mathbf{C}_{13} & \mathbf{C}_{16} * \mathbf{C}_{11} & +\mathbf{C}_{17} * \mathbf{C}_{15} \end{pmatrix}$ **Q10.22**

4: $\mathbf{p}_2 = \mathbf{in}_3 * \mathbf{in}_3 + (\mathbf{in}_2 * \mathbf{in}_2 + \mathbf{in}_1 * \mathbf{in}_1) >> \mathbf{1}$ **Q10.22**

5: $\mathbf{p}_4 = \mathbf{p}_2 * \mathbf{p}_2$ **Q19.13**

6: $\mathbf{Cff}_1 = \mathbf{2}^{31} + \left(\left(\mathbf{p}_4 * \frac{\mathbf{2}^{64}}{\mathbf{120}}\right) >> \mathbf{9} - \mathbf{p}_2 * \frac{\mathbf{2}^{46}}{\mathbf{6}}\right) >> \mathbf{7}$ **Q1.31**

7: $\mathbf{Cff}_2 = \mathbf{2}^{31} + \left(\left(\mathbf{p}_4 * \frac{\mathbf{2}^{67}}{\mathbf{720}}\right) >> \mathbf{9} - \mathbf{p}_2 * \frac{\mathbf{2}^{48}}{\mathbf{24}}\right) >> \mathbf{8}$ **Q0.32**

8: $\mathbf{C100} = \frac{\mathbf{2}^{37}}{\mathbf{100}}$ **Q − 5.37**

9: $\mathbf{C_2Cff_2} = \mathbf{C}_2 * \mathbf{Cff}_2 * \mathbf{C100}$ **Q3.29**

10: $\mathbf{out} = \mathbf{C}_1 * \mathbf{Cff}_1 + (\mathbf{C_2Cff_2}) >> \mathbf{3}$ **Q6.26**

11: $\mathbf{out} = \mathbf{I} + (\mathbf{out} * \mathbf{C100}) >> \mathbf{2}$ **Q1.31**

12: $\mathbf{DCM} = \mathbf{DCM} * \mathbf{out}$ **Q1.31**

13: $\mathbf{DCM} = \mathbf{DCM} + \frac{(\mathbf{I} - \mathbf{DCM} * \mathbf{DCM}') * \mathbf{DCM}}{\mathbf{2}}$ **Q1.31**

The matrix $\mathbf{C}_1$ is directly composed from the input data, and therefore its format is also Q6.26. The matrix $\mathbf{C}_2$, in contrast, is the square of $\mathbf{C}_1$, which requires increasing the number of bits reserved for the integer part, and respectively discarding some of the least significant bits (lsb). In the multiplications for computing $\mathbf{C}_2$, 30 lsbs are discarded from the product, and additions are not assumed to require additional integer bits as according to the dynamic range of the output, and the operands of theses additions, the same number of bits is needed to code the integer part of the output and the operands. Line 4: 30, 28 and 28 lsbs are discarded from the products of $\mathbf{in}_3 * \mathbf{in}_3$, $\mathbf{in}_2 * \mathbf{in}_2$ and $\mathbf{in}_1 * \mathbf{in}_1$, respectively. However, as the addition of $\mathbf{in}_2 * \mathbf{in}_2 + \mathbf{in}_1 * \mathbf{in}_1$ provides one additional bit, the sum needs to be shifted by 1 to align the radix points for addition with $\mathbf{in}_3 * \mathbf{in}_3$. On line 5, 31 lsbs are discarded from the product of $\mathbf{p}_2 * \mathbf{p}_2$. $\mathbf{Cff}_1$ is in Q1.31 and $\mathbf{Cff}_2$ in Q0.32 because their values are below 1 and 0.5, respectively. Shifts are needed to align the radix points for additions. On line 8, the sampling period of $\frac{1}{100}$ is represented in Q-5.37, i.e. the first significant bit of this number is the 6th bit of the decimal part. Thus to represent this number, the sign bit and the bits 6 ... 36 are needed, and $\frac{1}{100}$ is scaled by 2^{37}. On line 9, 31 lsbs are discarded from the products of $\mathbf{C}_2 * \mathbf{Cff}_2$ and $\mathbf{C100}$. Similarly, the 31 lsbs are discarded from the product of $\mathbf{C}_1 * \mathbf{Cff}_1$ on line 10. The output values on the lines 11, 12 and 13 are below 1, so only one bit is needed for representing the integer part of each output.

Divisions by constants are implemented by multiplication with a scaled fixed-point constant. For example, on line 6, division by 120 is implemented by multiplying $\mathbf{p_4}$ by $\frac{2^{64}}{120}$.

Algorithm 2 represents the fully featured version of the fixed-point DCM algorithm. However, in order to reduce the computational effort for low-end devices, also a couple of simplified DCM versions were created. The simplifications affect code lines 5, 6, 7 and 13 in Algorithm 2, and the versions are described below:

1. Basic: a basic fixed-point implementation that uses no rounding of computation results, but simply truncates results to the maximum supported bitwidth. In this version, the 4th degree polynomial component p_4 of Equation 4 and renormalization are not included in computations.
2. NN: an improved version of *basic*, where nearest-neighbor rounding is adopted in multiplications instead of truncation.
3. NN+P4: an improved version of *NN*, where the 4th degree polynomial component is included to computations.
4. NN+P4+ReNorm: same as the version *NN+P4* with renormalization of the DCM matrix.

Table 1 presents the computational complexity of each version in terms of additions and multiplications per one set (x, y, z) of input samples.

All the algorithm versions were written in plain C language without target-specific assembly. Intermediate results of fixed-point multiplication were stored to 64-bit integers before scaling back to 32-bits. The target processors had varying datapath bitwidths, but the target-specific C compilers automatically handled the arithmetic for bitwidths that surpass the target's datapath width.

5. EXPERIMENTS

The fixed-point version of the algorithm was evaluated with a dataset that had been acquired from a moving vehicle (a personal car). Since the original dataset contained relatively slow acceleration components, a second version of the dataset was generated by amplifying the signal such that the maximum dynamic range of the IMU sensors could be reached, i.e. it was converted to a *full-scale signal* prior to fixed-point conversion. The dataset was recorded at a frequency of

Table 1. Number of fixed-point additions and fixed-point multiplications for each algorithm version.

	Basic	***NN***	***NN*** +P4	***NN***+P4 +ReNorm
Additions	43	123	129	231
Multiplications	80	80	83	137

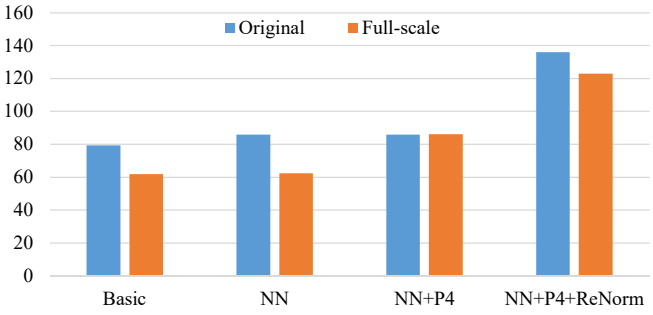

Fig. 2. Accuracy of the results as signal-to-noise ratio (in dB scale).

100 Hz and it consists of 66267 samples, which corresponds to 11 minutes in wall-clock time.

The fixed-point implementation needed to be compared to a reference implementation to be able to assess its accuracy. As the reference algorithm we used a double-precision floating point MATLAB implementation of the DCM algorithm that was based on trigonometric functions. As computation inaccuracies accumulate during the estimation process in inertial navigation, comparing the accuracy of the fixed-point version to the floating-point reference was possible simply by computing the signal-to-noise ratio (SNR) between the end result of the fixed-point version against the reference. Since the DCM matrix consisted of nine elements, each element was taken in account in the SNR computation using the equation $SNR = 20 * log_{10}((ref_0/fxp_0 + ref_1/fxp_1 + ... + ref_8/fxp_8)/9)$ dB, where *ref* and *fxp* are the DCM matrix values provided by the reference implementation, and by the fixed-point implementation, respectively. Subscripts indicate DCM matrix cell indices. When converted to linear scale, for example the SNR of 120 dB equals to a deviation of 10^{-6} in the DCM matrix cell value.

The graph of Fig. 2 presents the SNR of results for both the original input signal, and full-scale input signal that achieves the maximum IMU signal amplitude. Looking at Figure 2, it can be seen that the accuracy does not increase between versions *NN* and *NN+P4* for the original input signal. The reason for this is that the 4th degree polynomial component only has an effect when rapid acceleration changes are present, which is not the case for the original input.

This highlights the fact that when the DCM update algorithm is tailored for a low-resource device, the designer should be aware of a) the dynamic range of the input signal, but also b) the speed of signal variation. If it is known that the input signal can in no circumstances experience fast variations (e.g. a slowly moving robot), the 4th degree component can safely be omitted for somewhat increased computational efficiency.

Except for the aforementioned case, the fixed-point accuracy of algorithm versions steadily grows from version 1) to 4), however at the same time the computational complexity increases, which needs to be considered for ultra-low resource devices and for maximization of power efficiency and/or sample rate.

Table 2. Processing time results of *NN+P4+Renorm* on various platforms.

Device	***Achievable sample rate***
Microchip ATmega 2560 (8-bit)	0.275 kilosamples/s
SiFive Freedom E310 (32-bit)	12.68 kilosamples/s
Qualcomm Hexagon 682	2148 kilosamples/s

The computational complexity of the fixed-point algorithm was also measured on a variety of devices that are listed in Table 2. The devices reflect different categories of embedded processors, where the AtMega 2560 is a tiny 8-bit microcontroller, HiFive1 is a modern RISC-V microcontroller, and Hexagon 682 is a DSP commonly found in today's smartphones.

As the data wordlength of the AtMega 2560 is 8 bits, heavy software support (automatically provided by the C compiler) for computation with 32-bit integers is required. As a consequence of this software emulation of longer bitwidths, the achieved sample rate is only suitable for low frequency inputs, and even then it requires all the computation resources of the microcontroller.

The up-to-date RISC-V microcontroller can easily handle the processing at kHz rates, which means that such a processor can perform the DCM update besides other tasks. Finally, for the Hexagon DSP the fixed-point algorithm only yields a negligible computational burden.

6. CONCLUSION

In this paper, a fixed point version of the DCM update algorithm for inertial navigation has been provided, and its numerical accuracy has been measured for two different input signals with several accuracy-improving optimizations. The experimental results show that the accuracy of the fixed-point implementation can come very close to the floating-point version despite the fact that the algorithm is recursive, which causes accumulation of errors.

The results also show that considering the nature of the input signal is important in maximizing the computation efficiency; for example, if the input signal cannot be expected to contain fast variation, computations can be simplified without an impact on accuracy.

Measurements on three highly different target devices show that the fixed-point DCM update algorithm can be executed for low sample rates even on the smallest microcontrollers, and on more powerful devices the real-time computations only require a fraction of the processing time.

As future work, a quaternion-based solution of the algorithm should be implemented in fixed-point to see how this alternative formulation performs in terms of efficiency / accuracy. Also, as inertial navigation solutions inevitably drift quite rapidly, the numerical accuracy aspects of error constraining filters [13] such as extended Kalman filters should be also taken into account.

7. ACKNOWLEDGMENT

This work was partially supported by the ITEA3 project 16018 COMPACT.

8. REFERENCES

[1] Paul G. Savage, "Strapdown inertial navigation integration algorithm design part 1: Attitude algorithms," *Journal of guidance, control, and dynamics*, vol. 21, no. 1, pp. 19–28, 1998.

[2] John-Olof Nilsson, Amit K. Gupta, and Peter Händel, "Foot-mounted inertial navigation made easy," in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2014, pp. 24–29.

[3] Martin Nitschke and Ernst Heinrich Knickmeyer, "Rotation parameters - a survey of techniques," *Journal of surveying engineering*, vol. 126, no. 3, pp. 83–105, 2000.

[4] Robin B. Miller, "A new strapdown attitude algorithm," vol. 6, pp. 287–291, 07 1983.

[5] Moises J. Castro-Toscano, Julio C. Rodríguez-Quiñonez, Daniel Hernández-Balbuena, Lars Lindner, Oleg Sergiyenko, Moises Rivas-Lopez, and Wendy Flores-Fuentes, "A methodological use of inertial navigation systems for strapdown navigation task," in *IEEE International Symposium on Industrial Electronics (ISIE)*. IEEE, 2017, pp. 1589–1595.

[6] Jerome Vaganay, Marie-José Aldon, and Alain Fournier, "Mobile robot attitude estimation by fusion of inertial data," in *IEEE International Conference on Robotics and Automation*. IEEE, 1993, pp. 277–282.

[7] David Titterton, John L. Weston, and John Weston, *Strapdown Inertial Navigation Technology*, Electromagnetics and Radar Series. Institution of Engineering and Technology, 2004.

[8] Raymond Kristiansen and Per Johan Nicklasson, "Satellite attitude control by quaternion-based backstepping," in *American Control Conference*. IEEE, 2005, pp. 907–912.

[9] Shi Qiang Liu and Rong Zhu, "A complementary filter based on multi-sample rotation vector for attitude estimation," *IEEE Sensors Journal*, vol. 18, no. 16, pp. 6686–6692, 2018.

[10] Haoqian Huang, Xiyuan Chen, Bo Zhang, and Jian Wang, "High accuracy navigation information estimation for inertial system using the multi-model EKF fusing Adams explicit formula applied to underwater gliders," *ISA transactions*, vol. 66, pp. 414–424, 2017.

[11] Tariq S. Abuhashim, Mamoun F. Abdel-Hafez, and Mohammad Ameen Al-Jarrah, "Building a robust integrity monitoring algorithm for a low cost GPS-aided-INS system," *International Journal of Control, Automation and Systems*, vol. 8, no. 5, pp. 1108–1122, 2010.

[12] Eldon R. Hansen, "A generalized interval arithmetic," in *Interval Mathematics*, Karl Nickel, Ed., Berlin, Heidelberg, 1975, pp. 7–18, Springer Berlin Heidelberg.

[13] Jayaprasad Bojja, Jussi Collin, Simo Särkkä, and Jarmo Takala, "Pedestrian localization in moving platforms using dead reckoning, particle filtering and map matching," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1116–1120.