Analysis of Moving Average Filter for IMU Measurements on an 8-bit Microcontroller

Alessio Ghio, Sebastian Escalante and Jimmy Tarrillo
Universidad de Ingenieria y Tecnologia - UTEC, Lima, Peru.

***Abstract*—Noise is one of the biggest challenges in data acquisition and will be always present on every measurement made by any sensor, specially by Inertial Measurement Units (IMUs). Due to that, there are two approaches to solve this problem: analog or digital filtering. This second approach is usually the most taken, since very good precision can be achieved with mathematical operations performed by processors. However, depending on the application, the least energy consumption, time response or resources used are desirable. So, this paper presents the analysis of the well known Moving Average digital filter on an 8-bit AVR architecture microcontroller, because it is an interesting option as a result of its limited resources and its most likely probability of consuming less energy.**

I. Introduction

Applications that require signal processing for real measurements from a sensor will always encounter noise. A case of study for this problem is the Inertial Measurement Unit (IMU). An IMU is an electronic device, which is typically compounded by various sensors, for instance, an accelerometer, gyroscope, magnetometer, which are typically made of Micro electromechanical systems (MEMS) technology, because it is more attractive for embedded projects due to the capability of being miniaturized [1]. However, as stated before, error sources are always present as deterministic and stochastic error. Deterministic error sources can be bias, scale factor and misalignment. On the other hand, stochastic errors can be bias instability and scale factor instability [2]. In order to solve this noise problem, several techniques, algorithms and filters have been developed.

For example, Yi. et al.[3] used a low-cost IMU, a 3-axis accelerometer in combination with an optical wheel encoder to track the position of skid-steered mobile robots because of the complex dynamics interactions between the wheels and the ground. The previous work had a two-level hierarchy control system: in the top was the control algorithm and Kalman Filter, both implemented in a laptop carried by the robot, and a PID controller on the bottom. Though, it is well known that the measurements of the accelerometers are noisy and the the process to calculate position consists in integrating two times, the acceleration measured by the IMU will accumulate a vast amount of error. Several studies [4][5][6] involve attaching an IMU to a boot in order to track the wearers movement and do not specify where the data is processed. These studies suggest using filtering algorithms as the Zero velocity-update, which is a method to detect the persons stationary period and the use of magnetometer readings to improve the positioning accuracy and the Extended Kalman Filter (EKF), as the optimum filter, which could be replaced with the Unscented Kalman filter (UKF), because, in comparison, it provides higher calculation accuracy [7].

978-1-5386-5491-0/18/$31.00

Another widespread use of IMU systems is on wearables. Many applications of wearables, for instance, motion tracking, highly rely on the battery lifespan and on the accuracy of data measured by the IMU and processed, commonly by a microcontroller (uC) [8]. Although the filtering methods proposed on the works mentioned previously provide high accuracy on the final results, they require the use of heavy algorithms that would need 32 or 64-bit or floating point precision.

In other words, optimal filters have a high computational cost for an IMU and a uC. For instance, the use of lighter and less accurate filters is an option. e.g. Redhyka et al.[9] did a comparison of different parameters between Moving Average filter (MA) and Sensor Fusion (Complementary and Kalman filters) for an IMU on a uC based platform. These parameters were: overshoot, smoothness and rise time for a simulated step input, where the MA filter had one of the smoother response but a significant rise time in contrast with the others.

Since these kind of applications are specially challenging due to the need of autonomy and the low computational resources that embedded systems have such as an 8-bit uC in contrast with an 32-bit uC. In addition, floating point (float) data types are usually used on a vast amount of these applications, in which a wide interval of magnitudes is managed. However, many new programmers are not aware of the structure of the floating point (float) according to the IEEE 754 standard, which consists of 1 sign bit, 8 exponent bits and 23 fraction bits, for single precision and 1 sign bit, 11 exponent bits and 52 fraction bits, for double precision. In order to perform calculations with float data, extra hardware (floating point unit) or libraries that would increase the execution time and use more memory resources are needed.

For this reason, the aim of the present work is to analyze an implementation of a MA filter on an 8-bit AVR architecture that has no floating point unit, with different data types: 16-bit, 32-bit, 64-bit and float type. In this way, the impact of using specific data types can be determined. The structure of this work is as follows: first, the methodology is introduced in section II, then, the results are shown in section III and will be discussed in section IV. At the end, a conclusion is presented in section V.

II. Methodology

The process carried out to study and analyze a linear filter applied to raw IMU accelerometer data complete scheme is shown in Fig 1. In this section, there will be a brief summary about the implemented filter, secondly, it will be mentioned the error metric proposed and how are the use of resources determined. Finally, every block from Fig. 1 will be explained in detail.

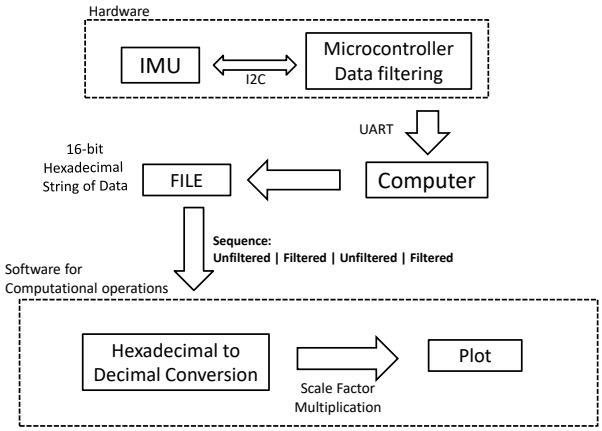

Fig. 1: Scheme of the process of data gathering, filtering and plotting.

A. Moving Average Filter

The Moving Average (MA) Filter was chosen to be implemented in the uC because of its simplicity, in which noise is reduced without compromising large amounts of data. The characteristic equation for this filter is as follows [10]:

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j] \tag{1}$$

Where y[i] is the output, and x[i] is the input, M is the number of readings that will be averaged value. Eq. 1 can also be described as a difference equation, which is also known as the recursive form:

$$y[i] = y[i-1] + x[i+p] - x[i-q] \tag{2}$$

$$p = \frac{M-1}{2} \tag{3}$$

$$q = p + 1 \tag{4}$$

This last equation is used in the process, which will be explained below, by the the uC and a software for computational operations, in this case, MATLAB. The use of this software is relevant due to its high accuracy, which will be used as a reference.

It is worth to mention that the frequency response of this filter is not optimal for signals that have valuable information in frequency domain due to its slow roll-off and poor stopband attenuation as it can be seen in Fig. 2 [10].

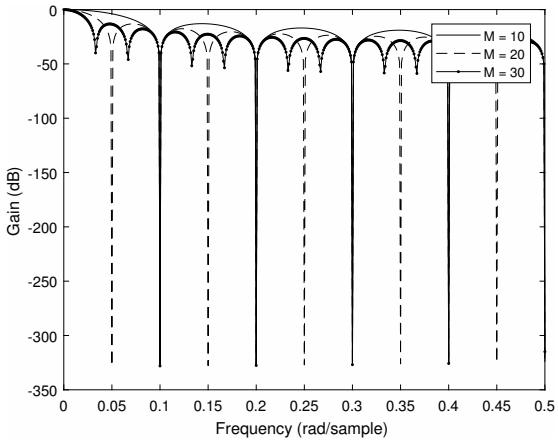

Fig. 2: Comparison of the frequency response of the Moving Average filter for different orders M.

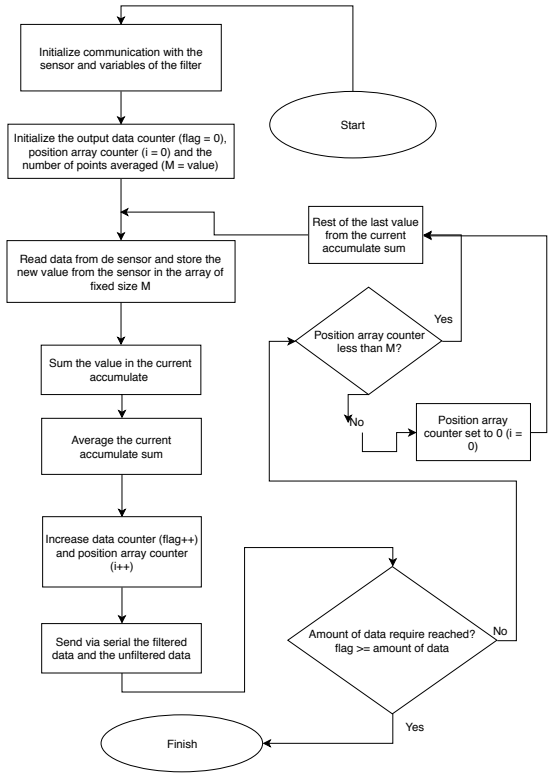

Fig. 3: Flux diagram of the MA filter implemented in the uC and the recollected data.

A graphical representation of the implementation in the uC is shown Fig. 3, this diagram is based in [11] and it is modified for periodical readings. Following this chart, the collected data is processed as explained in section II.E.

B. Error metric and use resources

The standard deviation (STD) is a dispersion or spread measurement about the arithmetic mean of any set of numerical values [12]. Since the sensor is settled in a fixed position with no movement applied to it, the ideal measurement should be a constant zero. However, due to the erratic nature of the IMU,

the signal will be noisy. So, applying this concept to the MA filter theory means that the bigger the window size, the less STD there is and the smoother the signal will be. On the other hand, for the use of resources on this specific analysis, the program memory usage is important. This value is calculated by the programming tool AtmelStudio, which includes the memory in bytes that is going to be used by the uC. Since the filtering algorithm of the MA filter is implemented using the recursive form it does not affect the use of resources. So, for this specific analysis, the window size lacks importance and as a consequence, this value was chosen arbitrarily.

C. Components, communication, software and data I/O

The first two blocks shown in Figure 1 represent the AVR architecture 8-bit uC and the MPU-6050 MEMS IMU used as main components. In addition, it shows that these two devices establish a communication by the I2C protocol. Then, in order to export data from the uC to a computer, the RS232 module along with a RS232 to TTL cable achieve that goal through serial communication. To visualize it, a Serial/TCP terminal [13] displays and also, saves the data as a string of hexadecimal characters.

D. Microcontroller raw data processing

The label Data filtering from the second block means that the uC performs, from data that it receives from the IMU, the MA filter on its recursive form with different types of data for the algorithm: int16, int32, int64 and float. Complete accelerometer measurements are 16-bit long, however, the IMU provides this information in two separate 8-bit data denominated as higher and lower acceleration. For further operations, these two are concatenated, filtered and then sent. However, all the information transferred through serial communication is not filtered, the uC also exports raw unfiltered data to a computer for later processing. On the other hand, only 8 bits are transferred by serial communication, so in order to export unfiltered raw data, the higher and lower acceleration are never concatenated and for the filtered raw data, 16-bit information is divided into two separate 8-bit data. The transmission sequence, shown in Fig. 1, is as follows: unfiltered, filtered, unfiltered, filtered and so on.

E. MATLAB raw data processing

As mentioned before, Serial/TCP terminal saves exported information by the uC on a file, which is then is loaded by MATLAB. An algorithm was developed to divide the hexadecimal string into groups of four (16-bit information) and subsequently, convert every group from hexadecimal to decimal signed integers. Later on, a MA filter is applied to the odd positioned data of the input vector, in other words, the unfiltered data is filtered. Then, information filtered by MATLAB and by the uC are multiplied by the accelerometer default configuration scale factor, which is 981.0/16384.0 and converts the raw data units from LSB/g to cm/s^2 [14]. Finally, filtered data from the uC and from MATLAB are compared by finding the euclidean distance of their subtraction and then, their respective STDs are calculated.

III. Results

A. Filter validation

In Fig. 4, the comparison of the filtered and unfiltered data from the uC readings is shown. As it can be observed the dotted line is smoother than the continuous line. It can be proved using the STD from each set of data. The unfiltered has an STD of 3.2235 and the filtered 0.4213.

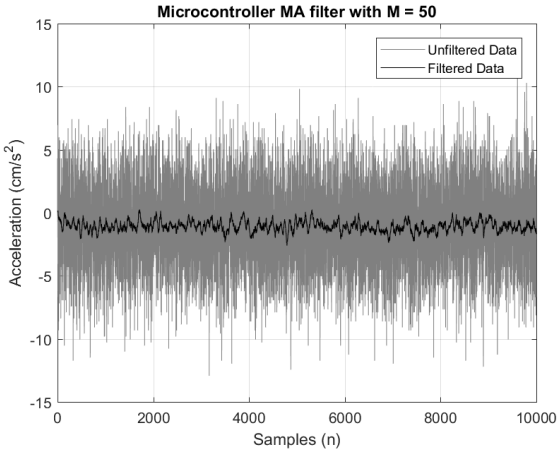

Fig. 4: Comparison of the filtered and unfiltered data from the uC with a window size of 50.

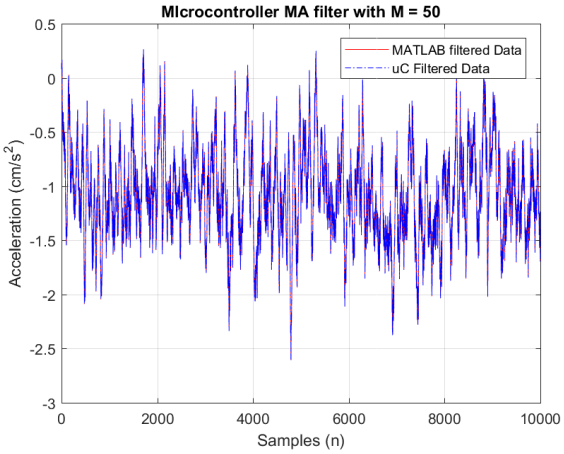

Fig. 5: Comparison of the filtered data from the uC and MATLAB with M = 10, only 300 of 1000 samples are shown for more visibility.

To verify the filtering performed by the uC, 11 sets of 10000 samples of filtered information by the uC and by MATLAB are compared. In addition, it should be emphasized that for these tests the uC used float data types. In Fig. 5, it can be observed that the signals overlap, meaning that both signals are identical or strongly alike. To validate this, the euclidean norm of the signals was taken for every set and afterwards, the mean value was calculated, resulting in the value 0.017. This implies that difference between the filtered data by the uC and by MATLAB is small. For this reason, the uC floating point data type filter will be used as reference for further comparisons.

B. Comparisons of filtering with different data types

A comparison of the number of Assembly instructions, memory usage and Euclidean norm of int16, int32, int64 and float data types is presented on Table. 1. Again, the Euclidean norm is used to determine how different the signals are. Hence, this is considered to be the error metric with respect to the new reference, mentioned on the previous subsection.

TABLE I: COMPARISON BETWEEN DIFFERENT DATA TYPES

	Data types			
	int16	**int32**	**int64**	**float**
Number of Instructions	122	143	190	458
Memory usage (bytes)	812	868	952	1374
Euclidean Norm (error)	3.3591	3.3591	3.3591	0

IV. Discussion

In Fig. 5, notice that readings are not positioned in zero because every time the IMU is used, it has to be calibrated so the mean of the data in a fixed position approximate to zero. Due to that, a calibration process in a uC is not trivial and can be done in several ways.

From Table 1 it can be observed that the number of instructions required for each data type increases as it changes from smaller to larger sizes. Using int32 data requires 17%; int64, 55%; and float, 275% more instructions than int16. This is consistent in sight of that the use of larger variables require more instructions to execute and that more bytes in memory will be used. It is important to take into account that more instructions are equivalent to more clock cycles, which means it increases the execution time and thus, more energy is consumed.

The Euclidean norm showed that, as expected, there is no variation between using int16, int32 or int64 data type in terms of precision and that there is a considerable difference between floating point and the other data types. Therefore, depending on the size of the values of the application and the accuracy to be achieved, it is of great importance to consider using a data type other than float, due to the trade offs offered. In this case, for the MA filter, it would be necessary to choose int32 or int64 data types if the window size is considerably big. This evaluation can be extrapolated to more complex filters where the use of certain data types could have a high impact on the code performance.

V. Conclusion

An analysis of the Moving Average filter implemented on an 8-bit uC is presented. It consists of the comparisons between the use of different data types with respect to a reference filter developed on a high programming language and then, the filter performed by the uC. Even though the tendency is to use 32 or greater bit uC architecture, it is possible for an 8-bit uC to perform said filter and obtain almost the same results as filtered data from a software for computational operations. On the other hand, in order to acquire such results, it is necessary to use float data types. Without a floating point unit, the number of instructions needed to perform operations increase by more than double than using int16 and in consequence, memory usage, execution time and energy consumption also increase. This might be a limiting factor for implementation of some applications in embedded systems. For this reason, it should be considered to use other data types variables, taking into account that accuracy is trade off with reducing the use of resources and energy consumption. Therefore, using data types different than float, depending on the application requirements, have a high impact on the performance of the controller.

VI. Acknowledgments

We want to thank very much Professor Renan Rojas for being always willing to help and most importantly, teach us throughout this work.

References

[1] "Introduction to Microelectromechanical Systems (MEMS) kernel description," https://compliantmechanisms.byu.edu/content/introduction-microelectromechanical-systems-mems, accessed: 2018-04-30.

[2] D. Unsal and K. Demirbas, "Estimation of deterministic and stochastic imu error parameters," in *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*. IEEE, 2012, pp. 862–868.

[3] J. Yi, J. Zhang, D. Song, and S. Jayasuriya, "Imu-based localization and slip estimation for skid-steered mobile robots," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, 2007, pp. 2845–2850.

[4] W. T. Faulkner, R. Alwood, D. W. Taylor, and J. Bohlin, "Gps-denied pedestrian tracking in indoor environments using an imu and magnetic compass," in *Proceedings of the 2010 International Technical Meeting of the Institute of Navigation*. Citeseer, 2010, pp. 198–204.

[5] J. Bird and D. Arden, "Indoor navigation with foot-mounted strapdown inertial navigation and magnetic sensors [emerging opportunities for localization and tracking]," *IEEE Wireless Communications*, vol. 18, no. 2, pp. 28–35, 2011.

[6] A. R. Jimenez, F. Seco, C. Prieto, and J. Guevara, "A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu," in *Intelligent Signal Processing, 2009. WISP 2009. IEEE International Symposium on*. IEEE, 2009, pp. 37–42.

[7] P. Zhang, J. Gu, E. E. Milios, and P. Huynh, "Navigation with imu/gps/digital compass with unscented kalman filter," in *Mechatronics and Automation, 2005 IEEE International Conference*, vol. 3. IEEE, 2005, pp. 1497–1502.

[8] J. Williamson, Q. Liu, F. Lu, W. Mohrman, K. Li, R. Dick, and L. Shang, "Data sensing and analysis: Challenges for wearables," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 2015, pp. 136–141.

[9] G. G. Redhyka, D. Setiawan, and D. Soetraprawata, "Embedded sensor fusion and moving-average filter for inertial measurement unit (imu) on the microcontroller-based stabilized platform," in *Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT), 2015 International Conference on*. IEEE, 2015, pp. 72–77.

[10] S. W. Smith *et al.*, "The scientist and engineer's guide to digital signal processing," 1997.

[11] LM. (2015) Arduino smoothing. [Online]. Available: https://www.arduino.cc/en/Tutorial/Smoothing

[12] J. M. Bland and D. G. Altman, "Statistics notes: measurement error," *Bmj*, vol. 313, no. 7059, p. 744, 1996.

[13] "Realterm: Serial/tcp terminal (version 2.0.0.70)," 2014. [Online]. Available: https://sourceforge.net/projects/realterm/

[14] *MPU-6000 and MPU-6050 Product Specification Revision 3.4*, InvenSense Inc., 8 2013, rev. 3.4.