

Analysis of microcontroller performance in drone flight control applications

Damian Kąkol

Abstract—This paper presents the results of measurements and analysis of microcontroller performance in the context of drone flight control applications. Main focus is placed on the speed and consistency of executing orientation estimation algorithms and sensor data acquisition. The study evaluates several commonly available microcontrollers with varying architectures and clock speeds and compares their absolute and relative performance. The findings aim to provide insights into the most critical characteristics of microcontrollers when selecting hardware for drone flight controllers, particularly considering low-cost and easily integratable solutions.

Keywords—drones; embedded systems; microcontrollers; benchmark

I. INTRODUCTION

DRONE control is a major application field for microcontrollers and orientation estimation algorithms. Advancements in miniaturization of processors and sensors enabled a fast growth of the drone market in the last decade. Drones are no longer exclusively used for military purposes. They have found their way into civilian applications thanks to the reduction in cost and size of their major components. Lately, we can see an inverse trend where civilian applications are adopted for military use precisely because of the advancements that have been made in the civilian sector. The major factor in this development is the cost reduction of the computational units, which now consist most often of a single microcontroller or a System on a Chip (SoC). This reduction of complexity is possible due to the massive increase in computational power of microcontrollers in the last decade.

In the wake of ongoing cost reduction of computational units many of them became accessible to the general public and hobbyists. At first 8-bit microcontrollers were the only ones commonly available and used in, for example, popular Arduino development boards. Nowadays, 32-bit microcontrollers are widely available and affordable. The performance they provide is sufficient for many applications that were previously reserved for more powerful and expensive systems. The availability of cheap and easy to use development boards and kits also plays an important role in their accessibility (Fig. ??). Many such microcontrollers are capable of running a Real-Time Operating System (RTOS), machine vision algorithms or performing complex computations for orientation estimation.

D. Kąkol is with Department of Signals and Systems Faculty of Electronics, Telecommunications and Informatics, University of Technology, Gdańsk, Poland (e-mail: damian.kakol@pg.edu.pl).

Drones are inherently unstable systems that require constant control input to maintain a stable flight. Most drones implement a closed-loop control system to generate the necessary control signals for the actuators and motors. This system relies on accurate information about the current orientation (pose) of the drone in 3D space. Many approaches exist to estimate this orientation using data from various sensors. In theory, an array of three accelerometers is sufficient to determine the orientation of a body. This is possible because, at standstill, the accelerometers measure acceleration caused by gravity in three axes. The resulting vector of acceleration points towards the center of gravity of the Earth, and thus can be used to determine the direction of the vertical axis. This approach, however, is not sufficient for real-world applications. Accelerometers are prone to noise caused by vibrations. Additionally, if the object is in motion (or, more precisely, accelerating in different directions), the accelerometers will not be able to distinguish between acceleration caused by gravity and that caused by movement. This results in erroneous orientation estimates.

To overcome these limitations, additional sensors are used to compensate for the weaknesses of accelerometer-only approach. Usually, a gyroscope and a magnetometer array are added to compliment the measurements. All of these sensors are usually combined into a single inertial measurement unit (IMU) chip manufactured in MEMS technology. The gyroscope measures angular velocity in three axes. By integrating the angular velocity over time, the change in orientation can be determined. This measurement is not affected by linear acceleration, but, due to the integration process, is prone to drifting over time. The error builds up, and, after a relatively short time, the gyroscope-only pose measurement would become useless. The magnetometer measures the strength of the magnetic field in three axes. This measurement can be used to determine the heading (yaw) of the system using the Earth's magnetic field as a reference. They, however, have to be calibrated to account for local soft and hard iron sources of magnetic interference [1]. All of these sensors have their strengths and weaknesses, and by combining their measurements, a more accurate and reliable orientation estimate can be obtained. Tools used to perform this task are called sensor fusion algorithms. Many such algorithms exist with different levels of complexity. Some of the most commonly used such tools are the Complementary filter [2], Kalman filter [3], and, most recently, various implementations of the Madgwick filter [4]. A complementary filter is a simple



algorithm that combines high-pass filtered gyroscope data with low-pass filtered accelerometer and magnetometer data. This crude solution allows for weighing the gyroscope data as more important during quick movements, and then revert to accelerometer information when at standstill. The Kalman filter is a much more complex algorithm that uses a series of measurements over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. It is computationally intense and requires tuning of variables specific to the sensors used to accurately model their response. The Madgwick filter is an algorithm that uses a gradient descent approach to minimize error in the orientation estimate. It is efficient and uses quaternions to represent orientation, which avoids issues like gimbal lock associated with Euler angles.

This paper aims to compare the performance of various microcontrollers when running the Madgwick sensor fusion algorithm for orientation estimation, as this algorithm is well-suited for real-time applications on resource-constrained devices. The comparison will be focused on execution time of the estimation loop and data acquisition time. Additional factors will be taken into account that may affect how the algorithm can be implemented into a system, such as the core count and multithreading support.

Many researchers have explored the field of effective benchmarking of embedded systems. Such works often focus on general performance of a device using simple algorithms, such as a sliding window filter [5]. This is a good approach to determine the performance of a microcontroller in real-world applications. It, however, does not provide insight into the performance of specific algorithms used in drone applications and cannot be linearly scaled to estimate it. Another aspect that is often left unexplored in such works is the consistency of execution time, which is crucial for real-time applications. Work [6] explores various time measurement techniques for microcontroller performance analysis. External and internal measurement methods are compared, where external methods use additional hardware to measure execution time, while internal methods use built-in peripherals of the microcontroller. The paper concludes that time measurements using internal modules of the microcontroller are more accurate, however, they may introduce constraints on the code being measured. This is an important aspect to consider when designing a benchmarking procedure, as measurement should not influence the performance of the algorithm being tested. Often omitted in works related to benchmarking performance is proper statistical analysis of the results. In paper [7] many such errors are highlighted and proper statistical methods for performance analysis are proposed. This includes many of the best practices implemented in this work, such as using geometric mean instead of arithmetic mean for averaging execution times, highlighting outliers and determining the variance and its significance in the results.

II. METHODOLOGY

The basis for a good benchmarking test is a well-defined and repeatable methodology. This section describes how the

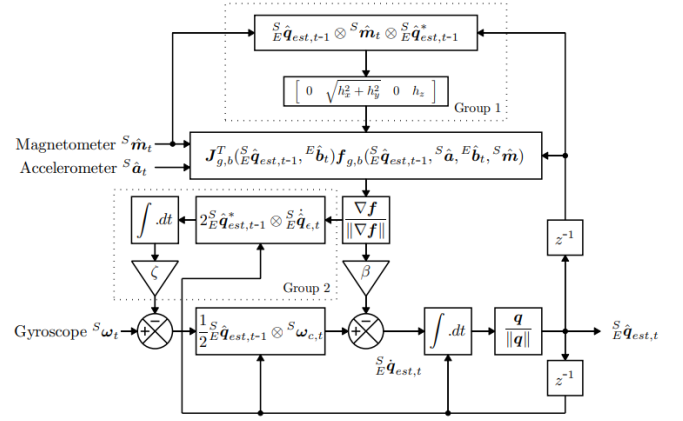


Fig. 1. Diagram of the calculations required for computing orientation based on IMU readouts using Madgwick filter. Many floating-point operations make it an ideal choice for benchmarking purposes. The graphic was published in the original Madgwick filter paper [4].

benchmark was designed and conducted in order to ensure accurate and applicable results.

A. Algorithm selection

The first step in this process was selecting the algorithm to be tested. The Madgwick filter algorithm was chosen due to the relatively low computational requirements and it being invented relatively recently, making it a relevant choice for modern Attitude and Heading Reference System (AHRS) applications. The algorithm's complexity is independent of the data length, as it is comprised only of basic floating point arithmetic and does not contain any recursive or data size dependent calculations (shown on Fig. 1). The algorithm was implemented in C programming language to ensure compatibility with all the selected microcontrollers while maintaining control over low-level resource management. The implementation is a fairly standard Madgwick filter with slight modifications to optimize its performance, such as using pre-computed constants. In total, the algorithm is comprised of 394 floating point operations, five square roots, two atan2 and one asin function calls (from the standard math C library). The algorithm was structured to separate data acquisition from processing, allowing for accurate measurement of both components independently.

B. Test group selection

In order to provide a comprehensive comparison a set of microcontrollers to be tested was carefully selected. Taking into account criteria mentioned in the introduction, such as affordability and availability, four popular microcontrollers were chosen for the evaluation:

- ESP32-S3 - a dual-core 32-bit microcontroller made primarily for IoT applications. Two Xtensa LX7 cores clocked at up to 240 MHz. A single-precision FPU is available. Low cost integrated modules are available.

- RP2040 - a dual-core 32-bit processor designed by the Raspberry Pi Foundation. Contains two ARM Cortex-M0+ cores with a maximum clock speed of 133 MHz, but can be overclocked beyond this value.
- RP2350 - an upgraded version of the RP2040 with clock speed of 150 MHz. However, the system was changed to include two cores based on the ARM Cortex-M33 or Hazard 3 RISC-V architecture. The core configuration can be selected in firmware.
- STM32F411 - a popular microcontroller from STMicroelectronics based on a single ARM Cortex-M4 core. Clocked at up to 100 MHz with a single-precision FPU. Heart of the commonly used Blackpill development boards.

All selected microcontrollers are widely available and affordable, which means the community support and development board availability is very high. These characteristics make them suitable for use in low-cost low-volume drone applications. Many architectures are represented in the selection, including Xtensa, ARM and RISC-V. Some microcontrollers feature multiple cores, significance of which will be discussed later in the paper.

In order to test the algorithm in a real world scenario, physical sensors with data to read out were needed. The choice of these sensors was dictated by the prevalence of them in common IMU modules used in drone applications. The following sensors were selected:

- MPU-6050 - a popular 6-axis IMU module containing a 3-axis accelerometer and a 3-axis gyroscope. Communicates via I²C interface.
- HMC5883L - a widely used 3-axis magnetometer module. Communicates via I²C interface.

Both sensors are low-cost and commonly used in drone applications while being compatible with all selected microcontrollers.

C. Benchmark design

The main purpose of the benchmark test is to be an universal, platform-agnostic test of performance. This is easily achievable when testing a software algorithm that does not require use of any processor-specific modules, such as peripheral interfaces. However, all the microcontrollers are programmed using their own development framework which includes ways to interact with these modules. This paper aims to evaluate both the performance of the raw algorithm as well as the speed of communication and data readout from the IMU sensor. As such, two quantities were measured for each test: the time taken to execute the benchmark algorithm and the time taken by both the algorithm and the communication steps, as illustrated in Figure 2.

After the processor initialization a peripheral initialization is performed. In all Devices Under Test (DUTs) this included starting the I²C peripheral device and setting the mode of its operation to Fast-mode. After this process is complete, the benchmark loop is started. This loop includes (Fig. 2) communication with the IMU sensors via I²C interface to read out accelerometer, gyroscope and magnetometer data. Then

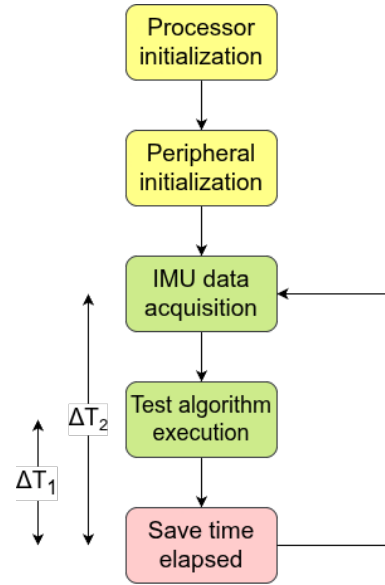


Fig. 2. Benchmark structure diagram. Two different time deltas ΔT_1 and ΔT_2 correspond to the two measurement methods. The benchmark sequence is repeated N times.

the Madgwick filter test algorithm is executed using this data. This process is repeated N times to gather sufficient data for statistical analysis. Two times are being measured when conducting the benchmark test. Period ΔT_1 is the amount of time taken to execute the test algorithm alone, while ΔT_2 indicates time taken by both data acquisition and processing. The measurement is conducted by getting the system clock value [6] at the beginning and end of the measurement period. Then, the two values are subtracted from each other to get a time delta. When the benchmark loop is finished, raw execution times are output via a serial interface for further analysis.

The target execution time for ΔT_2 is under 1 ms. This allows for a loop rate of 1 kHz or more, which is commonly used in drone flight controllers. The loop rate impacts the vehicle's responsiveness, which is important when operating an inherently unstable UAV.

D. Data processing

For each set of measurements basic statistical analysis was performed. This included standard measures, such as the geometric mean to indicate normal performance and standard deviation which hints at the stability of execution times. These results, however, show only the absolute performance of the devices under test. In order to compare them more accurately, a normalised performance metric has been implemented. This metric takes into account not only the time taken to execute a task, but also the clock speed at which the processor was running. This allows us to compare how well the microcontrollers would run, if they all had the same clock speed. The metric was calculated as follows:

$$P = \frac{1}{tf}$$

Where P is the normalised performance, t is the task execution time, and f is the clock frequency. This value can be interpreted as the frequency at which the loop would run, if the clock speed was set to 1 Hz (higher is better).

III. EXPERIMENT RESULTS

Benchmark test for each of the selected microcontrollers was conducted. The test consisted of 1000 test loops done in sequence. First 3 measurements were considered to be outliers and were removed from the dataset. Based on the data, histograms were prepared and basic statistics were calculated [7]. All values shown are in terms of milliseconds unless specified otherwise. All of the tests were performed using I²C running in fast mode. The results are shown in Table I.

All the devices implement the time measurement by counting the amount of clock cycles elapsed between two points in code. Then, the amount of cycles is divided by the clock frequency to get the time. The resolution of the measurement is highlighted in Table I and depends on the specific device. All of the microcontrollers besides the STM have a built in clock with 1 μ s resolution which was used for testing. The STM time was measured by accessing the register responsible for counting at 16 MHz frequency. This means the resolution of the STM measurement is 16 times better. The clock is very fast compared to the loop frequency, so the measurement resolution stays an order of magnitude below the measured time deltas.

In the case of ESP32-S3, the CPU was set to run at 240 MHz and the compiler mode was set to optimize for performance. It was found that this configuration decreased the loop time by around 12 % for ΔT_1 and 14 % for ΔT_2 compared to the stock SDK configuration. This change however increased the variability of the time deltas, as can be seen by comparing Fig. 3 and Fig. 4. In further deliberations the author will focus on the optimized performance values. The code executed on ESP32-S3 was utilizing FreeRTOS in order to select the cores responsible for executing the code and for interprocess communication. This introduces overhead in execution times, which is absent from other benchmarks conducted in this paper. FreeRTOS was used because the Espressif IDF implements it by default in its hardware drivers.

It is highly complex to use this microcontroller without using FreeRTOS and would require modifying the toolchain.

The RP2040 tests were conducted using stock configuration of 133 MHz clock speed and written using bare metal programming (without introducing overhead of any abstraction layer or RTOS). This is partly the cause of the very low standard deviation seen in benchmark data (Fig. 5). Otherwise the results were slightly worse than those of ESP32-S3.

As mentioned in previous sections, RP2350 can be configured to feature either ARM or RISC-V cores. Tests for both configurations were conducted at 150 MHz clock speed. Regarding the ARM results, there is a major improvement over the previous generation chip in the madgwick-only benchmark. This is due to the FPU added in this generation, which speeds up floating-point operations massively. The I²C results were also excellent, beating the ESP32-S3. As for the RISC-V core configuration, the RP2350 falls behind both ΔT_1 and ΔT_2 measurements, offering only slightly better performance than its predecessor. Both of these configurations show very good concentration of execution times, which is evident in the standard deviation values for the data. Overall, the processor performed very well in the benchmarks.

The I²C communication for STM32F411 was implemented using the STM32 HAL library, which is a high-level abstraction layer code library. The clock speed was set at 100 MHz for the best test results. This is consistent with the usage of vendor-specific implementations of this feature for the other microcontrollers. CubeMX was used to configure the I²C high speed bus with fast mode. The processor contains an FPU for enhanced floating point performance. In spite of this feature, the results for this microcontroller were worse than all of the above. The consistency of the calculation time was good, with relatively low standard deviation, as shown in Fig. 8.

The performance of microcontrollers varies depending on the task they are performing. When analyzing pure madgwick filter performance, without the I²C overhead, the RP2350 in ARM configuration is the clear winner. With excellent 0.0128 ms execution time, it is 4.37x faster than its closest competitor, the ESP32-S3, at performing the calculation (see Figures 6 and 4). The powerful Cortex-M33 cores clearly offer an

TABLE I
RESULTS OF BENCHMARK

MCU	Measurement	Geometric mean [ms]	Standard Deviation [ms]	Normalised Performance [10^{-6}]	Measurement Resolution [μ s]
ESP32-S3	ΔT_2	0.490	0.00474	8.5	1
RP2040	ΔT_2	0.453	0.000662	16.6	1
RP2350 ARM	ΔT_2	0.217	0.000461	30.7	1
RP2350 RISC-V	ΔT_2	0.409	0.000349	16.3	1
STM32F411	ΔT_2	1.780	0.00420	5.6	0.0625
ESP32-S3	ΔT_1	0.056	0.00594	74.4	1
RP2040	ΔT_1	0.234	0.000681	32.1	1
RP2350 ARM	ΔT_1	0.013	0.000443	512.8	1
RP2350 RISC-V	ΔT_1	0.193	0.000461	34.5	1
STM32F411	ΔT_1	0.709	0.00178	14.1	0.0625

advantage over the Xtensa LX7 of the Espressif. As mentioned before, the ESP32 was also slowed down by the overhead of FreeRTOS. The impact of this overhead was somewhat mitigated by the usage of the second core for running the benchmark task, as this core is not running any other tasks by default. In case of the combined I²C and Madgwick workload, RP2350 still came out as the fastest processor to do the job, in spite of using built-in functions for I²C communication. The execution time was also remarkably consistent for this chip, both in ARM and RISC-V configurations shown in Figures 6 and 7.

When comparing the normalised performance metric in Table I, the performance of the RP2350 becomes even more clear. In both the I²C and madgwick only scenarios, its performance was exceptional. Comparing this metric also allows us to see that the ESP32-S3 does not utilize its cores as well as the ARM equipped RP2350, being second only to the STM32. The gap between processors is the biggest when comparing raw Madgwick calculations, without I²C overhead. There, an order of magnitude difference can be noticed between RP2350 and the rest of the devices. It is also apparent that the RP2350 in RISC-V configuration shows almost the same performance per hertz of clock speed as the RP2040.

The RP2350 is the clear winner of this benchmark, satisfying the need for a fast microcontroller that performs well in real-time tasks. It is worth noting that a dedicated port of FreeRTOS is available for this processor, which allows for implementation of complex real-time systems utilizing both M33 cores. The ESP32-S3 also delivered solid performance, being the second fastest processor in the roundup. It also offers additional benefits, such as built-in wireless communication, which may be worth the performance sacrifice in drone-specific implementations. RP2040 and STM32F411 performed the worst of the tested devices, lagging significantly behind their competitors.

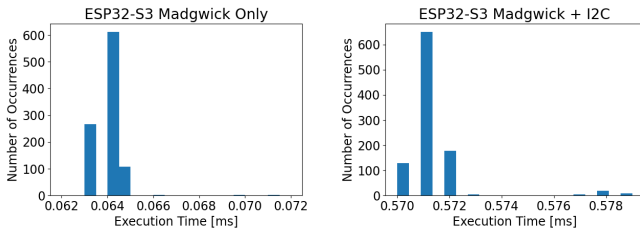


Fig. 3. Histogram of ESP32-S3 execution times with stock SDK settings.

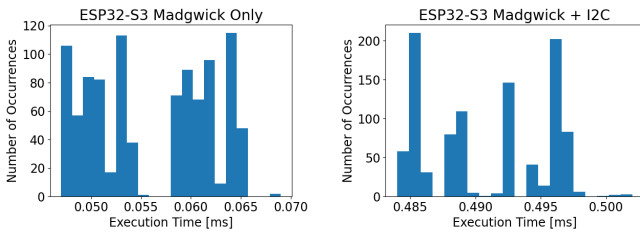


Fig. 4. Histogram of ESP32-S3 execution times with optimized performance.

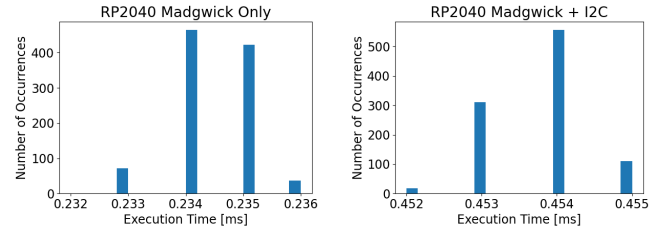


Fig. 5. Histogram of RP2040 execution times.

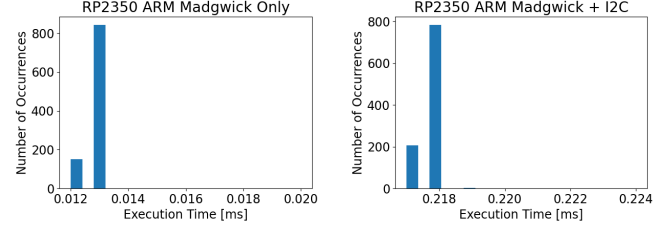


Fig. 6. Histogram of RP2350 execution times with ARM core selected.

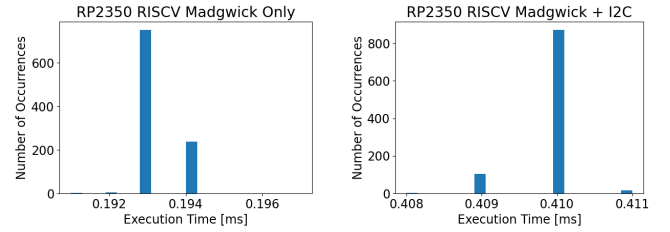


Fig. 7. Histogram of RP2350 execution times with RISC-V core selected.

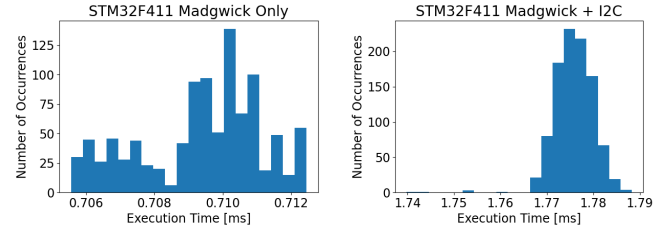


Fig. 8. Histogram of STM32F411 execution times.

IV. CONCLUSION AND FUTURE WORKS

Widely available, affordable microcontrollers are offering better performance than ever before. The only chip that did not match the 1 ms execution time for ΔT_2 was the STM32F411. Most of the devices under test performed well, but it is crucial to note the differences between them. The performance between similarly priced microcontrollers with similar clock speeds ranged over an order of magnitude. It should be noted, that the tests were conducted on close to stock system configurations for a given microcontroller. This made the tests fair and gave equal opportunity for all the vendors. For example, the RP2040 has a factory clock speed of 133 MHz, but users have successfully overclocked the chip even above 500 MHz. This is however not officially supported and has unknown effects on the longevity of the product [8].

Real time performance is a complex topic that cannot be described only by clock speed or the core count of a system. This study has proven, that inexpensive and easily available microcontrollers are well capable of running a task like attitude estimation at a good enough speed to be used in real-time systems, such as UAV controllers. The RP2350 from the makers of the popular Raspberry Pi single board computers performed exceptionally well. This relatively new microcontroller has many additional features, that make it suitable for real-time operation, such as the additional PIO state machines that facilitate offloading input/output signal processing to a dedicated hardware unit. This allows for implementation of obscure or not widely supported communication protocols (such as those used by hobby grade Electronic Speed Controllers) without introducing overhead. Additionally, FreeRTOS is available for this MCU, which makes it a great choice for multi-tasking.

In future works, the author aims to build a drone control unit utilizing this microcontroller that implements FreeRTOS and PIO to communicate with speed controllers and a main Single Board Computer, estimate position based on IMU data and conduct measurements of the surrounding area.

ACKNOWLEDGMENT

I would like to thank my peers, who brought me on a path leading to the publication of this paper. I am further grateful

to my family and co-workers for the motivation to conduct research.

REFERENCES

- [1] S. Lian-yan, H. Qing, and S. Wen-yuan, "A research of an improved ellipse method in magnetoresistive sensors error compensation," in *2009 International Conference on Mechatronics and Automation*, 2009, pp. 4105–4109. [Online]. Available: <https://www.researchgate.net/publication/224591641>
- [2] P. Gui, L. Tang, and S. Mukhopadhyay, "Mems based imu for tilting measurement: Comparison of complementary and kalman filter based data fusion," 06 2015, pp. 2004–2009. [Online]. Available: <https://www.researchgate.net/publication/308850497>
- [3] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME-Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
- [4] S. O. H. Madgwick, "An efficient orientation filter for inertial and inertial / magnetic sensor arrays," 2010. [Online]. Available: <https://api.semanticscholar.org/CorpusID:2976407>
- [5] N. Žilvinas, "Embedded microcontrollers benchmarking using sliding window algorithm," *Elektronika ir Elektrotechnika*, pp. 1392–1215, 01 2007.
- [6] A. Buturuga, R. Constantinescu, and D. Stoichescu, "Time measurement techniques for microcontroller performance analysis," *2017 IEEE 23rd International Symposium for Design and Technology in Electronic Packaging (SIITME)*, pp. 38–43, 10 2017.
- [7] P. Fleming and J. Wallace, "How not to lie with statistics: The correct way to summarize benchmark results." *Commun. ACM*, vol. 29, pp. 218–221, 03 1986.
- [8] M. Thothadri, "An analysis on clock speeds in raspberry pi pico and arduino uno microcontrollers," *American Journal of Engineering and Technology Management*, vol. 6, pp. 41–46, 06 2021.