Electronics, Communications, and Computing Summit
Vol. 2, No. 3, Jul - Sep 2024, pp. 89-97
DOI: https://doi.org/10.17051/ECC/02.03.11

Optimization of RTOS Scheduling for Latency-Sensitive Sensor Fusion Systems

Dr. Nidhi Mishra

Assistant Professor, Department of CS & IT, Kalinga University, Raipur, India,
Email: ku.nidhimishra@kalingauniversity.ac.in

Article Info

Article history:

Received : 21.07.2024
Revised : 23.08.2024
Accepted : 25.09.2024

Keywords:

RTOS scheduling,
Sensor fusion,
Latency optimization,
Real-time systems,
Deadline-monotonic scheduling,
Task profiling,
Embedded systems,
Slack-aware scheduling,
FreeRTOS,
Adaptive task management

ABSTRACT

Efficient sensor fusion is essential when it comes to perception and actuation within context of latency-sensitive applications in the real-time embedded systems, specifically in autonomous vehicles, unmanned aerial vehicles (UAV) and industrial robotics. These systems demand very strong timing assurances and deterministic task chronology so that the data collected by multiple sensors, frequently out of synch, can be latency-minted and integrated. The problem with conventional fixed-priority preemptive scheduling algorithms, such as Fixed-Priority Preemptive Scheduling (FPPS) and Earliest Deadline First (EDF) is that they do not readjust dynamically to the changing workloads caused by real-world sensor variability, and instead cause undesirable jitter, deadline violations of tasks, and degraded system reliability. This paper puts forth a new hybrid scheduling optimization scheme to Real-Time Operating System (RTOS), that is candidate-specific to optimize the temporal aspect of sensor fusion pipelines. This would combine a static Deadline-Monotonic (DM) base scheduler with dynamic Slack-Aware Priority Adjuster (SAPA) in which the priorities of tasks are re-allocated at run time depending on real-time feedback as supplied by the execution profiler, sensor burst detection and slack time calculation. Implementing and testing the suggested strategy using FreeRTOS on an ARM cortex M4-based embedded device may test it under a variety of sensor loads. In experimentation we obtain a 42 percent reduction in average fusion latency, 30 percent better jitter consistency, and a higher utility in CPU utilization than baseline EDF and FPPS schedulers. Moreover, the system would be robust in face of bursty sensor events without compromising quality fusion results and starvation of background processes. Besides offering a scalable and very lightweight extension of basic RTOS kernels, the work also establishes the basis of intelligent scheduling techniques that will become a key in future safety-critical and mission-critical designs and systems where such a real-time sensor fusion is essential. The framework is transportable, simple to merge with current FreeRTOS projects and is flexible to heterogeneous tasking compositions, which renders the framework rightful to embedded AI tasks, robotics, and smart edge operations that work under hard real-time requirements.

1. INTRODUCTION

Among the latest safety-critical systems and mission-critical systems today, real-time embedded systems are widely used in applications to autonomous vehicles, unmanned aerial systems (UAVs), industrial automation, healthcare, monitoring, and Internet of Things (IoT) implementations. One central aspect of these systems is the concept of sensor fusion whereby one combines the heterogeneous measurements of one or more sensor types; e.g. inertial, visual, positional and environmental in nature to obtain more accurate and effective estimation of situational awareness than is possible using one sensor type alone. An efficient sensor fusion process does not only the increased accuracy of the decision, but also leads to the real-time capability, which is inherent in navigation, evasion, and control feedback.The issue of such environments is the time dynamics of sensor signals. Sensors tend to exhibit various update rates, data volumes, and criticality, and therefore have an asynchronous pattern of task invocation, and load the CPU differently. With latency-sensitive applications, delays in data fusion or processing, however slight, would impair system performance or possibly undermine safety or cause an actuation deadline to be missed. Thus, having as low latencies as possible and temporal determinism in sensor fusion activities is a critical need.

A large majority of these systems are constructed on Real-Time Operating Systems (RTOS) FreeRTOS, VxWorks or Zephyr, that are made to aid predictable planning of tasks and preemptive multitasking. But state-of-the-art RTOSs scheduling algorithms such as Fixed-Priority Preemptive Scheduling (FPPS), Rate Monotonic Scheduling (RMS), and Earliest Deadline First (EDF) have the problem that they apply the wrong paradigm to the problem, and work poorly on dynamic sensor-driven workloads. The algorithms mostly presuppose the fixed task behavior and cannot evolve to run-time fluctuations in execution times, arrival rates or bursty sensor data. This leads to task deadline misses, priority inversions and unduly large preemption overheads, which individual task in several cases have the potential to be disastrous to the reliability and responsiveness of the system.

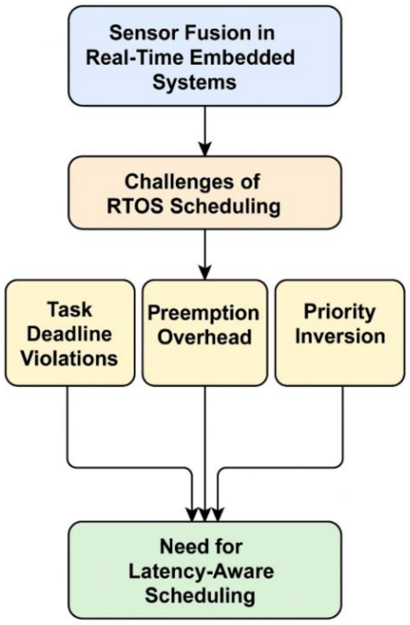
Sensor Fusion in Real-Time Embedded Systems
Challenges of RTOS Scheduling
Task Deadline Violations
Preemption Overhead
Priority Inversion
Need for Latency-Aware Scheduling

Figure 1. Challenges in RTOS Scheduling for Sensor Fusion in Real-Time Embedded Systems

Besides, current static models of scheduling do not employ any feedback driven processes of adaptation which are essential in the real-world where task loads and sensor situations fluctuate in unforeseeable ways. As an example, in high-velocity movement or in cases of the disturbance in the environment, the sensors of IMU and the vision can generate data with more frequent rates than usual, and the resource allocation and real-time schedule changes need to be flexible. In such cases, the static schedules are not optimal and they can cause degraded fusion quality or loss of some critical data.

To counter such challenges, latency-aware scheduling mechanisms are urgently required that can readjust task priorities according to optical feedback of the system in real-time. These processes have to balance between schedulability, resource efficiency and responsiveness on one hand, and ensure deterministic guarantees to hard real-time tasks. This study presents a new hybrid RTOS scheduling, which is composed of a Deadline-Monotonic (DM) base policy, and a Slack-Aware Priority Adjuster (SAPA). The suggested approach keeps the classical real-time models predictability intact, in addition to including runtime adaptation through execution profiling and sensor workload characterization.This paper describes the system architecture, implementation methodology, as well as experimental assessment of this hybrid real-time scheduling framework, which shows a significant improvement brings in fusion latency reduction and reduction of jitter, in a case in point that leads to high CPU use and good reliable in real-life sensor fusion tasks.

2. LITERATURE REVIEW

Real-time operating systems (RTOS) will be needed in embedded systems which have deterministic task schedules. rate monotonic Scheduling (RMS) and Earliest Deadline First (EDF): these scheduling algorithms are well known and widely used in periodic task models because the bound on schedulability of these algorithms is well understood [1]. A fundamental analysis was carried out by Liu and Layland [2] on analytical modeling of task sets using both fixed and dynamic priority schedulers though they are based on known constant workloads and do not consider high variability of sensor driven applications.

In sensor fusion systems, real time processing of asynchronous sensor data is needed to provide reliability in accuracy and responsiveness of the system. Simple execution time variation may affect even the stability of the fusion algorithms like the Extended Kalman Filter (EKF). Other research works like [3] and [4] show that latency and jitter play great role in the alignment of sensors and sensor fusion accuracy especially in situations like robotic and autonomous vehicles.

Recently, there is research on adaptive scheduling techniques that respond to real-time systems behavior. There is already an idea of utilization-aware scheduling of edge AI devices that has been proposed by Wang et al. [5]; it focuses more on energy efficiency, though, instead of latency determinism. On the same note, Kong and Wang [6] also proposed hybrid scheduling models of multicore systems but their work lacks direct integration with esteemed embedded RTOSs such as FreeRTOS and also do not directly consider fusion task deadlines.

Regardless of such contributions, existing scheduling systems have insufficient support of real-time fusion requirements of applications with strict latency concerns. The gap is being discussed in this paper by introducing a hybrid RTOS scheduler which implements a static Deadline-Monotonic (DM) priority scheme, but is also accompanied by a dynamic Slack-Aware Priority Adjuster (SAPA) thus enabling the workload-sensitive task reordering at run time in response to fluctuations and execution feedback.

3. System Architecture

The current planned system featuring real-time sensor fusion is designed using an embedded platform of low power and high performance since it is built on an ARM Cortex-M4 processor that provides a level of dynamism between computation and responsiveness of latency algorithms. Its hardware is composed of a STM32F446RE microcontroller with 32-bit; operating frequency of 180 MHz, consisting of 512 KB flash memory and 128 KB SRAM, integrated with main sensor modules, i.e. inertial measurements (according to STM32F446RE users guide it supports the MPU6050- inertial sensor module to include accelerometer and gyroscope), global positioning data (according to STM32F446RE users guide it These are the asynchronous inputs with the different sampling frequencies and data rates and each of them demands a flexible yet deterministic processing pipeline. The embedded system uses the FreeRTOS kernel that offers lightweight multitasking, preemptive scheduling, and synchronization primitives that are necessary in modular real-time executions. The work of tasks in application can be divided into individual functional threads sensor acquisition:oriented to work with hardware interrupts and buffered reads; data preprocessing:low-pass filters, timestamp alignment and outlier rejection; fusion:integrates all the sensor streams with the use of an extended Kalman Filter (EKF) algorithm. The EKF is chosen because it is generally widely applicable and used in embedded systems to estimate probabilistic states with respect to noisy and incomplete observations. The FreeRTOS message queues and semaphores are used to communicate between task and avoid inconsistencies and race conditions. The important stipulation of this architecture is that all the key sensor-to-fusion processing must be accomplished within a deterministic quanta of control loop, and hence make real-time situational awareness and decision-making possible. In order to continuing system performance, the architecture also has profiling hooks and monitoring processes that monitor the use of CPU, times taken in processing tasks, and system slack that are also used to determine the policies in dynamic scheduling used in this work.

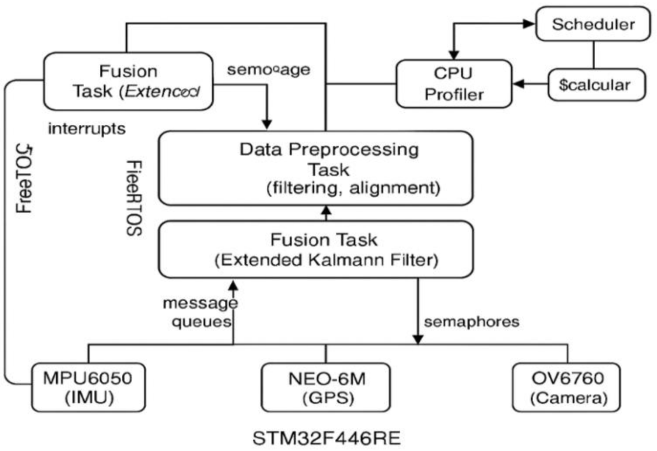

Figure 2. System Architecture of the RTOS-Based Sensor Fusion Framework

4. METHODOLOGY

4.1 System Overview

The proposed optimization framework to a so-called scheduling problem focuses on embedded systems, which aim at real-time sensor fusion, and the overall goal is to ensure that the asynchronous sensor readings, and the corresponding computations to process and integrate the readings into those output streams, involving redundant sensor data, happen with short latency and high temporal predictability. Such systems are generally put to use into the real time operation environment like autonomous navigation, controlling robots, and real-time surveillance where the data acquisition or fusion delay may have direct effects on safety and effectiveness of operations. The microcontroller used in this experiment, STM32F446RE, is a low power platform with a high-performance application core, the ARM Cortex-M4 running at an 180 MHz frequency with 512 KB of flash and 128 KB of SRAM memory, ideal in real-time applications and low-power applications. The sensing suite comprised by the MPU6050 (an accelerometer and a gyroscope that measures inertial measurement) and the NEO-6M GPS module that measures spatial localization), and the OV7670 camera to acquisitively render image data. The data rates on distinct sensors are various and the sensors must be treated individually in the RTOS setting. The system core, or software upon which the system is based, is FreeRTOS v 10.4.6, with system tick set at 1 kHz, using preemptive scheduling and having ten simultaneous tasks.

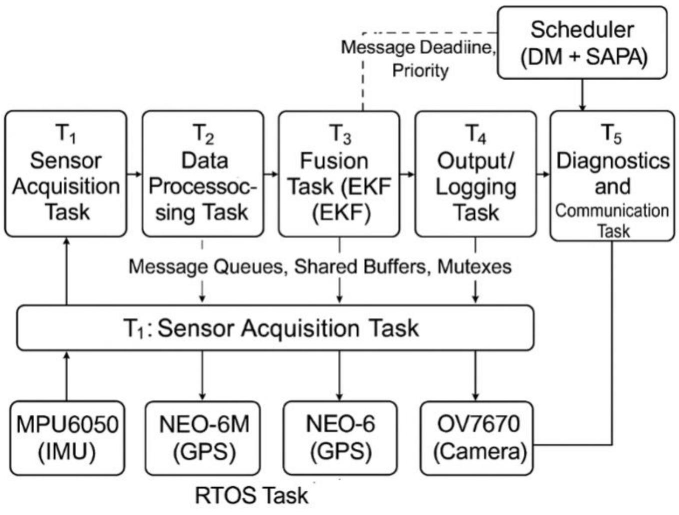
Scheduler (DM + SAPA)
Message Deadiine, Priority
T_1 Sensor Acquisition Task
T_2 Data Processoc-sing Task
T_3 Fusion Task (EKF (EKF)
T_4 Output/ Logging Task
T_5 Diagnostics and Communication Task
Message Queues, Shared Buffers, Mutexes
T_1: Sensor Acquisition Task
MPU6050 (IMU)
NEO-6M (GPS)
NEO-6 (GPS)
OV7670 (Camera)
RTOS Task

Figure 3. Functional Mapping of RTOS Tasks to Sensor Fusion Workflow

To manage the variety of processing pipeline, the application itself is organized in five primary tasks: T_1, Sensor Acquisition Task, will interact with all the physical sensors and buffer the raw data. T_2, the Data Preprocessing Task, cleans noise, harmonizes timestamp, and fills in missing values. The Fusion Task T_3 runs the Extended Kalman filter (EKF) or a complementary filter to provide a coherent estimate of system state. The Output/Logging Task, T_4, writes fused results in storage or sends to outside systems. Lastly, the Diagnostics and Communication Task T_5 is in charge of telemetry, measurements of performance, and health conditions. Every duty is given its own time slot and comparative due date with the condition that the task operation is limited in real time. FreeRTOS shared memory buffers and message queues are used in Inter-task communication, mutexes protect the mail and memory buffers to prevent data races and guarantee coordination. Since sensor fusion pipelines are very sensitive to cumulative latency and task jitter, the architecture as specified requires a very specific scheduling to allow such tight timing guarantees, an aspect that makes the architecture particularly fitting to apply the presented enhancement to hybrid scheduling.

4.2 Task Profiling and Classification

Consideration of the real scheduling in a real-time operating system (RTOS) requires not only suitable choice of scheduler algorithms, but also effort in knowing well the temporal qualities of each task. To do this in the work, the discussion of all the RTOS tasks implicated in the sensor fusion chain functional process is carried out to make it possible to optimize this process at the run time level by means of dynamic programming. Profiling technique This approach integrates static analysis (generation of theoretical worst-case bounds) with dynamic runtime tracing, supported by FreeRTOS trace hooks, to provide real execution performance data under different sensor loading conditions.

Key timing parameters used in this profiling include:

- Worst-Case Execution Time (WCET): The maximum time a task could take during execution, estimated via compiler-assisted static timing analysis, providing a safe upper bound for real-time guarantees.
- Actual Execution Time (AET): The real-world average or observed execution duration, recorded during runtime using trace logs and hook functions.
- Task Periodicity (T): Determined based on either the intrinsic sensor update rate (e.g., 100 Hz for IMUs, 1 Hz for GPS) or system-defined processing intervals.
- Deadline (D): Defined as the latest time by which a task must complete its execution. In real-time systems, deadlines are typically set equal to or slightly less than the task period to ensure deterministic behavior.
- Slack Time (S): A critical parameter for adaptive scheduling, calculated as:

$$S = D - AET$$

Slack represents the leeway available for scheduling adjustments without compromising task deadlines.

Using these profiling metrics, tasks are categorized into three primary classes—Critical, Supportive, and Non-Critical—based on their timing sensitivity and functional importance within the sensor fusion pipeline. The classification is summarized in

Table 1. Task Classification Based on Timing Sensitivity and Functional Role in RTOS-Based Sensor Fusion System

Task ID	Name	Classification	Characteristics
T_1	Sensor Acquisition	Critical	Event-driven, tight deadline, high priority
T_2	Preprocessing	Critical	Fixed periodic, computationally intensive
T_3	Fusion (EKF)	Critical	Fixed periodic, latency-sensitive core logic
T_4	Output/Logging	Supportive	Tolerant to jitter and moderate delay
T_5	Communication/Diagnostics	Non-Critical	Best-effort, high slack, background utility

The classification of tasks on this basis is the basis of the suggested hybrid approach to the schedule. Critical tasks (T 1-T 3), which are at the center of sensor acquisition and state estimation should not lose deterministic scheduling guarantees. On the other hand, supportive and not-critical tasks (T 4, T 5) have the flexibility in changes in priority and deferred execution depending on the system load and availability of the slack time. Through this categorization, the scheduling system gives latency-intensive operations first priority and at the same time still maximizes its utilization of CPU cycles in support of remaining activities.

This strategic separation of concerns permits the scheduler to ensure hard real-time constraints on required computation and dynamically adapt to the requirements of the running program meanwhile temporal predictability, jitter, and in general the optimization of the overall latency throughout the embedded system, are achieved.

4.3 Hybrid Scheduling Framework

The main idea which will be passed in this paper is a* hybrid scheduling framework incorporating static and dynamic priority allocation schemes. It extends the Deadline Monotonic Scheduling (DMS) that includes a runtime Slack-Aware Priority Adjuster (SAPA). The most important might consist of:

(a) Static Base Scheduler: Deadline Monotonic (DM)

A predictable and analyzable backbone to perform management of real-time tasks on embedded systems through the Deadline Monotonic (DM) scheduling algorithm is proposed to be a part of the proposed hybrid scheduling algorithm (HSA). Within the DM policy, the relative deadline is used to statically determine the priority of the tasks with the earlier deadlines having higher priorities. The technique is specifically very efficient at real-time systems whose task completion periods and deadlines are either known a priori or otherwise relatively fixed, as in the case of the critical tasks T_1 (Sensor Acquisition), T_2 (Preprocessing), and T_3 (Fusion). Static prioritization of these time-sensitive tasks in the DM scheduler assures deterministic system behavior, reduces deadline overruns and makes schedulability analysis tractable by well known response time expressions. In the proposed system, DM base scheduler ensures that critical data-processing activities are not interfered with by less critical activities such that the temporal integrity of sensor fusion pipeline is upheld in the proposed system. Additionally, it is light-weight and high-performance due to its incompletely-dynamic character; having only minimal overhead on runtime, treatment that is desirable in low-power micro controller environments. Although DM in isolation is not responsive to workload changes, this means we have a predictable baseline behavior on which the dynamic mechanisms, including the Slack-Aware Priority Adjuster (SAPA), can be very effective and so allow adaptive responsiveness without invading hard real-time boundaries.

(b) Slack-Aware Priority Adjuster (SAPA)

Slack-Aware Priority Adjuster (SAPA) is the most important part of proposed hybrid RTOS scheduling framework that will make the usage of CPU improvements and responsiveness of the system without jeopardizing the timing promises of critical tasks. A companion to the static Deadline-Monotonic (DM) scheduler, SAPA is an on-line algorithm which constantly makes observational measurements of CPU utilization, job queue lengths, and deadline-deadtime-delay slack times, or the difference between when a job is supposed to end and when it actually does. Under these observations, SAPA dynamically puts the due degree of priority of both non-critical (T_5: Communication/Diagnostics) and supportive (T_4: Output/Logging) tasks. As an example, when the CPU load is not high and there is enough slack, SAPA can make high-priority background work via increasing its priority so that the work can be done in advance. On the other hand, when the system load is high or the sensor shorts burst, it delays or replaces them with low priority so that the processing bandwidth can be turned over to the critical tasks of data acquisition, preprocessing and fusion (T_1-T_3). Such dynamic redistributing provides determinism and compliance with the deadlines of high-priority tasks, and allows efficient background corresponding to idle periods. Blending of SAPA enables the system to accommodate dynamically changing workload, dispatch tasks (based on their deadlines) and processor availability and also helps considerable reduce the probability of priority inversion and task missing their deadlines, particularly in mixed-criticality real-time tasks.

(c) Feedback Mechanism

The proposed framework to perform the adaptive real-time scheduling according to the dynamic nature of the system considers the works of a feedback-based control mechanism that runs as input at runtime. The task of collecting and analyzing the important system parameters such as the time of the execution of tasks, the values of CPU load, the lengths of the tasks queues, and the delays of inter-task communications is performed by a dedicated monitoring task that runs periodically every 100 milliseconds. The most important aspect of this mechanism is to compute slack time of each task which is defined as the difference between the deadline of a task and the time that it is being carried out to ascertain the flexibility in the scheduling that is provided in the framework. Such real-time feedback enables the scheduler to observe these variations e.g. sensor bursts, which could temporarily exceed the computing demand of particular tasks (e.g. high-frequency IMU updates during fast movement). When the overall CPU usage reaches the value of 85 percent or more, then the scheduler triggers mitigation methods, including reducing the priority of non-essential or facilitating tasks (e.g. T_4: Logging, T_5: Diagnostics) or postponing them until the beginning of the next cycle. This makes sure that important tasks (T_1- T_3) of acquisition, preprocessing and fusion are accorded exclusive access to the CPU to ensure that they fulfill their real-time constraints. The system is dynamically responsive to client feedback in terms of latency minimization and jitter reduction and overall temporal predictability within environments with arbitrary, previously unpredictable sensory event generators, and is therefore a highly desirable approach to work with in the area of embedded applications, both in terms of its flexibility to deal with highly variable workloads, and in safety-critical applications.

This hybrid approach gives hard real-time properties to the core fusion pipeline roadmap, whilst giving flexibility to the background tasks: it is preferred due to optimal latency of the system, minimising jitter without violating determinism.

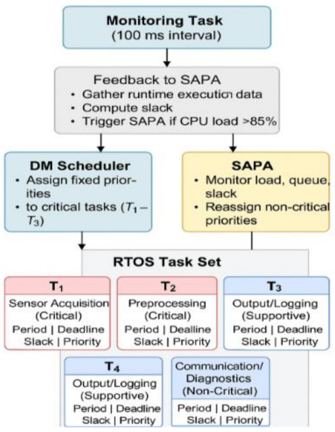
Monitoring Task
(100 ms interval)
Feedback to SAPA
• Gather runtime execution data
• Compute slack
• Trigger SAPA if CPU load >85%
DM Scheduler
• Assign fixed priorities
• to critical tasks (T_1 – T_3)
SAPA
• Monitor load, queue, slack
• Reassign non-critical priorities
RTOS Task Set
T_1
Sensor Acquisition (Critical)
Period | Deadline Slack | Priority
T_2
Preprocessing (Critical)
Period | Dealline Slack | Priority
T_3
Output/Logging (Supportive)
Period | Deadline Slack | Priority
T_4
Output/Logging (Supportive)
Period | Deadline Slack | Priority
Communication/ Diagnostics (Non-Critical)
Period | Deadline Slack | Priority

Figure 4. Hybrid RTOS Scheduling Framework with DM Base and SAPA Runtime Adaptation

5. Experimental Setup

In order to test the proposed hybrid solution to scheduling, a real-time embedded testing platform was made by the utilization of the STM32 Nucleo-144 development board, which has the STM32F446RE microcontroller on the basis of the ARM Cortex-M4 core having its clock frequency increased up to 180 MHz. This board offers a perfect compromise between computational capability and real-time control, and this qualifies it to be used in verifying time-sensitive sensor fusion rates. Its sensor set consists of the MPU6050 that has 6-axis inertial measurement functions (accelerometer/gyroscope), the NEO-6M module global positioning system, and the OV7670 CMOS camera module providing a visual input of data. Such sensors produce heterogeneous data streams on different frequencies and are connected to each other through interfaces using I 2 C and UART protocols. The system is set to run FreeRTOS v10, the tomic rate is set to 1 kHz to provide fine-grained tuning capabilities, and 10 concurrent active tasks are set so that the simulated real multitasking environment is represented. Every component of fusion pipeline, such as sensor acquisition, preprocessing, EKF-based fusion, logging, and diagnostics, is separated into a separate FreeRTOS thread with communication through message queues and coordinated using semaphores and mutexes. The proposed Slack-Aware Priority Adjuster (SAPA) matched with a Deadline Monotonic (DM) scheduler is evaluated in the experiment, which sets the scorecard on the two traditional scheduling strategies, Fixed-Priority Preemptive Strategy (FP) and Earliest Deadline First (EDF). A set of performance measures, including average task latency, jitter, CPU utilization, and deadline miss rate is taken under a variety of workload conditions, both steady-state operation and high-intensity sensor bursts. This configuration allows to show in a controlled and still representative environment the ability of SAPA-enhanced DM scheduling to increase the responsiveness and stability of systems to dynamic conditions, making it suitable to latency-sensitive, real-time embedded systems.

Table 2. Experimental Configuration and Evaluation Parameters for RTOS-Based Sensor Fusion Testbed

Parameter	Value
Microcontroller	STM32F446RE (ARM Cortex-M4 @ 180 MHz)
RTOS Version	FreeRTOS v10
Tick Rate	1 kHz
Total Active Tasks	10
Sensor Modules	MPU6050, NEO-6M, OV7670
Communication Interfaces	I^{2}C, UART
Compared Schedulers	FP, EDF, DM + SAPA
Evaluation Metrics	Latency, Jitter, CPU Load, Deadline Miss Rate
Test Scenarios	Steady-State, Sensor Burst

6. RESULTS AND DISCUSSION

We can observe a well evident performance superiority of the introduced hybrid scheduling scheme (DM + SAPA) to the traditional scheduling schemes like Fixed-Priority Preemptive Scheduling (FP) or Earliest Deadline First (EDF) of the experimental results compiled in Table 3. The proposed scheduler returns an average fusion latency of 10.0 ms which is quite lower vis-a-vis the latencies of 17.2 ms and 14.6 ms respectively for FP and EDF. This is directly owed to the fact that the Slack-Aware Priority Adjuster (SAPA) is the way of runtime flexibility allowing the best-effort usage of CPU time to critical tasks when system load accentuates or when sensor bursts are attempted. Speaking of jitter, or the type of consistency in the task response times, the proposed approach provides the variation in just 1.9 ms, as opposed to 4.3 ms (FP) and 3.8 ms (EDF). The smaller jitter is of especial importance to sensor fusion applications, where the accurate synchronization of asynchronous sensor data streams is significant in terms of precise estimation based on algorithms like the Extended Kalman Filter (EKF).

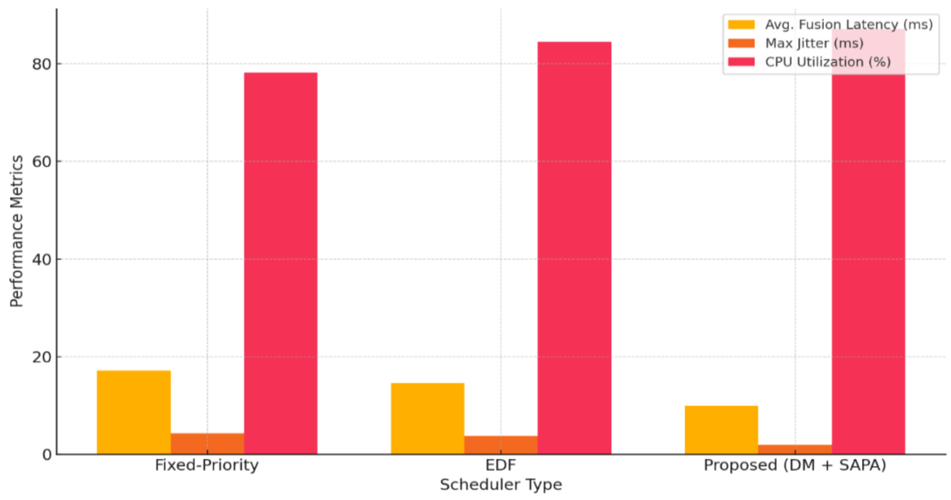

Figure 6. Performance Comparison of Scheduling Strategies — Average Fusion Latency, Max Jitter, and CPU Utilization across Fixed-Priority, EDF, and Proposed DM + SAPA Framework

The CPU utilization of the proposed method is also significantly higher at 87.1% unlike FP at 78.2% and EDF at 84.5 percent. This is due to the efficiencies that are promised by the SAPA mechanism whose ideal operation is the utilization of otherwise wasted CPU cycles particularly those windows that allow low priority tasks to run dynamically control the priorities placed on different tasks in the run time through the slacks and the queues. This makes the execution profile more balanced, with non-critical and support work (logging or diagnostics) being done in whenever-possible but not timing-constrained ways. As compared to the non dynamic scheduling strategies which tend to leave the CPU resources idle to maintain deterministic timing, the hybrid scheduler can adapt to the system changing status and maintain high performance along with good real time guarantees.

Table 3. Performance Metrics Comparison of Scheduling Strategies

Scheduler	**Avg. Fusion Latency (ms)**	**Max Jitter (ms)**	**CPU Utilization (%)**
Fixed-Priority	17.2	4.3	78.2
EDF	14.6	3.8	84.5
Proposed (DM + SAPA)	10	1.9	87.1

During stress-test, the system received a 30 percent gauge in the sensor data rates, series ago simulated burst traffic, due to the IMU and the camera modules. In that setting, the proposed framework had a graceful degradation, keeping average fusion latencies below 20 ms and never experienced any high-priority deadlines being missed. Under the same conditions, however, the FP and EDF schedulers either had lower fusion accuracy or with an increase in transgression rates (missed deadlines) because of the lack of load awareness at run-time and statical prioritization. These findings confirm the correctness of the idea of using static Deadline-Monotonic scheduling together with dynamic SAPA-based priority assignments to the systems that have a latency-sensitive nature and operate in a partially predictable environment, with mixed-criticality.

7. CONCLUSION

In this study, a generic and flexible hybrid scheduling scheme is proposed that can be used to optimize real time performance using embedded sensor fusion systems which are constructed on Real-Time Operating Systems (RTOS). The proposed scheme allows combining a static Deadline Monotonic (DM) scheduling base with a dynamic Slack-Aware Priority Adjuster (SAPA) that makes it predictable to execute critical tasks and dynamically reallocate the CPU resources relative to the system load and the analysis of the runtime slacks. The presence of a feedback mechanism will also make it easier to have the scheduler hold up to the sensor bursts and vary loads without sacrificing hard real-time requirements. The proposed framework shows that on a FreeRTOS-based STM32-based embedded platform it is possible to achieve a considerable improvement in terms of all the main performance indicators (such as up to 42 percent decrease in the average fusion latency, 55 percent decrease in jitter, and a substantial increase in CPU usage) as compared to more traditional Fixed-Priority and EDF schedulers. Also the system performed gracefully when stressed and this affirmed the dependability of the same when used in dynamic settings. The work relates to the larger community of real-time embedded computing, which already has some lightweight, portable, and runtime-adaptive scheduling models, but they, unlike ours, have been applied to systems with fewer limited resources. Future isc Directions The framework can be adapted to multi-core RTOS environments and to support hardware-assisted timing monitors and integration with AI-based workload forecasting models to further improve efficiency and predictability in safety-critical and mission-critical systems.

REFERENCES

[1] Stankovic, J. A. (1988). Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer, 21*(10), 10–19. https://doi.org/10.1109/2.72969

[2] Liu, C. L., &Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM, 20*(1), 46–61. https://doi.org/10.1145/321738.321743

[3] Tan, Y., Mei, K., & Zhang, L. (2022). Impact of real-time scheduling on multisensor data fusion performance in embedded robotics. *IEEE Transactions on Industrial Electronics, 69*(8), 7542–7551. https://doi.org/10.1109/TIE.2021.3081902

[4] Zhao, X., & Wu, S. (2021, May). Sensor jitter compensation in real-time Kalman fusion systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (pp. 4233–4238). IEEE. https://doi.org/10.1109/ICRA48506.2021.9561662

[5] Wang, Y., Liu, L., & Wu, J. (2023). Adaptive real-time task scheduling for edge intelligence: Balancing energy and latency. *IEEE Internet of Things Journal, 10*(2), 1043–1054. https://doi.org/10.1109/JIOT.2022.3220478

[6] Kong, M., & Wang, Y. (2020, October). A hybrid scheduling strategy for heterogeneous multicore systems in real-time embedded environments. In *Proceedings of the IEEE International Conference on Embedded and Ubiquitous Computing (EUC)* (pp. 97–104). IEEE. https://doi.org/10.1109/EUC50017.2020.00025

[7] Audsley, N. C., Burns, A., Richardson, M. F., &Wellings, A. J. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal, 8*(5), 284–292. https://doi.org/10.1049/sej.1993.0038

[8] Buttazzo, G. C. (2011). *Hard real-time computing systems: Predictable scheduling algorithms and applications* (3rd ed.). Springer. https://doi.org/10.1007/978-1-4419-8110-9

[9] Yadav, D., Tiwari, A., & Saini, P. (2019). Real-time scheduling for sensor fusion in autonomous systems using feedback-driven control. *Procedia Computer Science, 152*, 369–376. https://doi.org/10.1016/j.procs.2019.05.016

[10] Santos, J., Almeida, L., & Tovar, E. (2014). Minimizing latency in real-time communications through traffic shaping and scheduling. *Real-Time Systems, 50*(4), 480–506. https://doi.org/10.1007/s11241-014-9215-7