

Edge Camera based on a Single-Board Computer: Hybrid Selective Parallelization in Real-Time Video Processing*

Oleksii Bahatskyi^{1,2,*}, Kyrylo Malakhov^{1,†}

¹ Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine, 40 Glushkov ave., Kyiv, 03187, Ukraine

² Borys Grinchenko Kyiv Metropolitan University, 18/2 Bulvarno-Kudriavska Str, Kyiv, Ukraine, 04053, Kyiv, Ukraine

Abstract

This paper presents the design, implementation, and experimental evaluation of a low-cost edge camera based on a single-board computer (SBC) capable of real-time video preprocessing. The objective of the study is to assess whether a resource-constrained SBC platform can sustain real-time performance under a sequence of primitive image-processing transformations, and to determine the effectiveness of selective parallelization in achieving this goal. A generalized hardware model of an SBC-based edge node is proposed, integrating a USB video sensor, system-on-chip processing unit, local storage, and network interface. On the software level, a user-space processing architecture implemented in C++14 and OpenCV is developed. The architecture combines a staged transformation chain with a thread pool mechanism to enable parallel execution of computationally intensive operations across multiple CPU cores. The prototype was implemented on an Orange Pi Zero 2W (Allwinner H618, quad-core Cortex-A53, 4 GB RAM) using a 640×480 USB camera stream. Three transformations – Laplacian filtering, histogram computation, and Gaussian blurring – were evaluated. Experimental results demonstrate that a fully sequential implementation achieves 24 FPS, which does not satisfy the 25 FPS real-time constraint. By parallelizing the Laplacian stage within a hybrid parallel-sequential pipeline, the system reaches 28 FPS, corresponding to a 16.7% throughput improvement. The findings confirm that low-cost SBC platforms can function as edge nodes for lightweight real-time preprocessing tasks, provided that computational workloads are carefully structured and selectively parallelized.

Keywords

IoT, SBC-Based Edge Camera, Single-board Computer, Real-Time Video Processing, Laplacian filter, Gaussian blur, OpenCV

1. Introduction

Edge computing has emerged as a fundamental architectural paradigm for distributed cyber-physical and IoT systems, particularly in scenarios involving high-volume sensor data such as video streams. In contrast to centralized cloud-based processing, edge architectures relocate computation closer to the data source, thereby reducing latency, minimizing bandwidth consumption, and improving responsiveness. For video-based systems – including surveillance, industrial monitoring, and intelligent transportation – these properties are critical for real-time analytics and event-driven decision-making.

Recent advances in *single-board computers* (SBCs) have enabled the deployment of compact, low-cost devices with multi-core processors and integrated network interfaces. Such platforms provide a practical foundation for implementing edge video nodes. However, their limited computational resources impose strict constraints on achievable throughput, especially when performing continuous image transformations at standard frame rates. The central challenge is therefore to design a software architecture that maximizes performance within the boundaries of constrained hardware.

This paper examines the capacity of a low-cost SBC platform to support real-time edge-camera operation and evaluates whether selective parallelization of primitive image-processing operations is sufficient to satisfy real-time constraints. In this work, real-time operation is defined as processing each frame within the

2nd Workshop "Intelligent information technologies" UkrProg-IIT'2026 co-located with 16th International Scientific and Practical Programming Conference UkrPROG'2026, May 5-6, 2026, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

✉ o.bahatskyi@kubg.edu.ua (O. V. Bahatskyi); MalakhovKS@nas.gov.ua (K. S. Malakhov)

ORCID 0000-0003-2969-6630 (O. V. Bahatskyi); 0000-0003-3223-9844 (K. S. Malakhov)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

temporal budget determined by the camera frame rate. For a 25 FPS stream, this corresponds to a maximum processing time of 40 ms/frame.

The contributions of this study are threefold. First, a generalized hardware model of an SBC-based edge camera is formulated, identifying its essential architectural components. Second, a user-space processing architecture implemented in C++ and OpenCV is proposed, combining a staged transformation chain with a thread pool mechanism for intra-stage parallelism. Third, an experimental evaluation on an *Orange Pi Zero 2W* platform demonstrates that hybrid parallelization increases throughput from 24 FPS to 28 FPS for the tested workload, thereby achieving real-time performance under the specified conditions.

1. Related Works

SBCs, particularly Raspberry Pi-class platforms and their analogues, are widely used as low-cost hardware for edge-oriented video processing and embedded computer vision. Prior studies have shown that such devices can support localized image acquisition, preprocessing, and application-specific analytics, thereby reducing dependence on centralized infrastructure and improving responsiveness in distributed video systems. In this context, SBC-based devices are increasingly treated not merely as peripheral capture units, but as compact edge nodes capable of performing nontrivial computation directly at the point of data generation.

Early work in this area demonstrated that SBC platforms can sustain basic image-processing operations for real-time applications. For example, [1] considered edge-detection procedures implemented on Raspberry Pi hardware, showing that compact embedded platforms can execute lightweight vision algorithms with acceptable performance characteristics. In a different application domain, Yildirim et al. [2] investigated real-time image and video processing for in-vehicle driver assistance systems, where timely analysis of road conditions, surrounding objects, and driver behavior is required. Their study further confirmed that SBC-class devices can be used in latency-sensitive scenarios, although computational resources remain a limiting factor.

A substantial portion of the literature is devoted to intelligent surveillance and recognition tasks. Asri et al. [3] proposed an SBC-based smart security camera combining OpenCV with Haar cascade object detection and LBPH-based recognition. Similarly, Ijaradar and Xu [4] developed a cost-efficient facial-recognition surveillance system for homes and small offices, in which faces are first localized in the frame and then matched against a predefined database. In a related direction, [5] presented an attendance management system based on Raspberry Pi and OpenCV, where facial recognition was used to automate identity registration. Collectively, these studies demonstrate the practical applicability of SBC platforms to real-time surveillance, monitoring, and recognition-oriented tasks.

SBC-based image processing has also been explored in domain-specific settings beyond surveillance. In particular, [6] addressed the classification of fruits by color and size in an agricultural context, illustrating that low-cost embedded vision platforms can support specialized classification workflows. Such results confirm that SBC-based systems are sufficiently flexible to serve as computational endpoints in a broad spectrum of applied vision scenarios, provided that the processing pipeline is appropriately matched to the available hardware resources.

Beyond camera-centered studies, recent work also supports the relevance of hybrid and heterogeneous processing strategies for edge-oriented systems. Scalable ECG clustering [7], ontology- and NLP-driven intelligent systems [8, 9, 10], semantic and RDF/XML processing frameworks [11, 12], and computational modeling based on finite element analysis and process simulation [13, 14] collectively show that modern intelligent systems increasingly depend on decomposing workloads across heterogeneous computational resources. Although these studies are not specific to SBC-based video cameras, they support the architectural premise of the present work: lightweight, latency-critical preprocessing should be executed locally on the SBC, while more computationally intensive tasks may be delegated to more capable distributed nodes.

Despite the breadth of prior work, the existing literature is predominantly oriented toward end-application functionality, such as detection, recognition, classification, or semantic processing, rather than toward the programming-level organization of the user-space processing pipeline itself. In particular, comparatively less attention has been paid to the extent to which selective parallelization of primitive transformations can

shift a low-cost SBC-based vision system from near-real-time to real-time operation under an explicitly defined frame-rate constraint. The present study addresses this gap by focusing not on a specialized recognition task, but on the implementation efficiency of a lightweight C++/OpenCV processing architecture for primitive video transformations on an SBC-based edge camera.

2. Materials and Methods

2.1. Real-Time Processing Criterion

For video-processing systems, real-time operation is defined relative to the frame acquisition rate of the camera. Let f_{cam} denote the camera frame rate, expressed in frames per second (FPS). Then the maximum admissible processing time per frame is determined by the frame period:

$$T_{frame} = \frac{1}{f_{cam}}. \quad (1)$$

A processing system satisfies the real-time requirement if the average time required to process one frame, denoted T_{proc} , does not exceed the available frame period:

$$T_{proc} \leq T_{frame}. \quad (2)$$

Equivalently, if the effective processing throughput is denoted by f_{proc} , then the real-time condition can be written as:

$$f_{proc} \geq f_{cam}. \quad (3)$$

In the present study, the input video stream is configured at 25 FPS, which corresponds to a commonly used frame rate in video transmission standards [15]. Therefore, using (1), the maximum admissible processing time per frame is:

$$T_{proc} = \frac{1}{25} = 0,04s = 40ms. \quad (4)$$

Accordingly, any implementation for which $T_{proc} \leq 40$ ms/frame is considered non-real-time under the adopted experimental conditions.

2.2. Hardware Architecture of the SBC-Based Edge Camera

The generalized model of the proposed edge-based camera is shown in (Figure 1). The model is derived from the earlier concept of an intelligent video camera [16] and is adapted here to the requirements of edge-oriented deployment. The architecture consists of four principal components.

The first component is the sensor subsystem, which acquires the video stream and delivers it to the processing node. Depending on the implementation, the sensor may be connected via dedicated interfaces such as CSI or via general-purpose interfaces such as USB. The sensor may also be represented by a single camera or by a group of coordinated sensing devices [17].

The second component is the video processing unit, which performs frame acquisition, transformation, and intermediate analysis. In the SBC-based implementation considered in this work, this role is performed by a system-on-chip (SoC) integrating multiple CPU cores and auxiliary processing resources.

The third component is the local storage subsystem, which provides persistent or temporary storage for the operating system, application binaries, intermediate results, and diagnostic data. In low-cost SBC configurations, this functionality is typically implemented using microSD or eMMC storage.

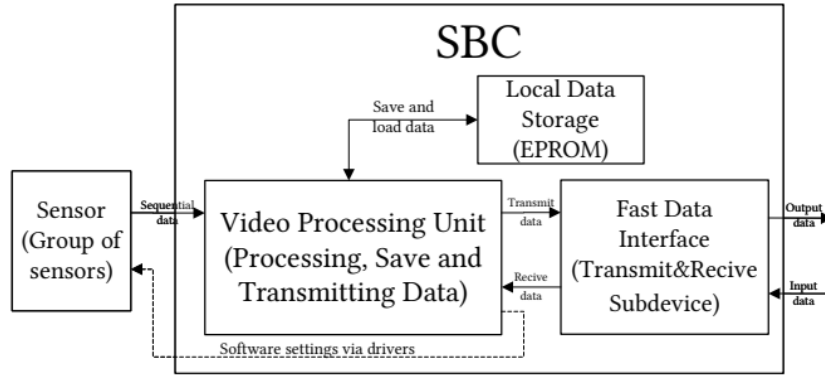


Figure 1: The generalized model of the proposed edge-based camera.

The fourth component is the communication interface, which transmits the processing results or selected frame data to external systems. In the simplest implementation, this function is provided by the integrated Ethernet or Wi-Fi interface of the SBC, although more advanced communication subsystems are also possible [18]. Thus, an SBC-based edge camera integrates sensing, local computation, storage, and communication within a compact and low-cost platform. However, the practical effectiveness of such a node depends critically on the software stack and on the computational organization of the video-processing workflow.

2.3. Software Stack and Device Interface

The basic software layer of the proposed system is a Linux-based operating system, which provides hardware abstraction, driver support, process scheduling, and memory management for user-space applications. In SBC platforms of the considered class, Linux distributions such as Ubuntu are commonly used due to their hardware support and compatibility with standard vision libraries.

Communication with the camera device is organized through the standardized Video4Linux2 (V4L2) interface [19], which provides system-level access to video capture devices and enables frame acquisition from the user-space application. This interface serves as the boundary between the camera hardware and the application-level processing logic implemented in C++.

The user-space application was developed in ISO/IEC 14882:2014-compliant C++ using the OpenCV library. This combination was selected because it provides low-level control over execution flow, support for efficient memory handling, compatibility with multithreaded programming, and a mature implementation base for image-processing operations.

2.4. User-Space Processing Architecture

The proposed user-space processing architecture is organized as a staged frame-processing workflow implemented in C++ and OpenCV, as shown in (Figure 2). Each input frame passes through a fixed sequence of transformations, and the result of each stage is forwarded to the next stage until the full processing chain is completed.

The architecture combines a transformation chain based on the Chain of Responsibility pattern [20] with a ThreadPool mechanism for bounded parallel execution [21]. The transformation chain defines the processing order, whereas the thread pool enables selected stages to exploit multiple CPU cores.

In the hybrid implementation, only computationally intensive transformations are parallelized. For such stages, the input frame is partitioned into fragments processed concurrently by worker threads, after which the partial results are merged. The remaining stages are executed sequentially. This design, schematically presented in (Figure 2), reduces processing time without introducing unnecessary synchronization overhead and is therefore well suited to resource-constrained SBC platforms.

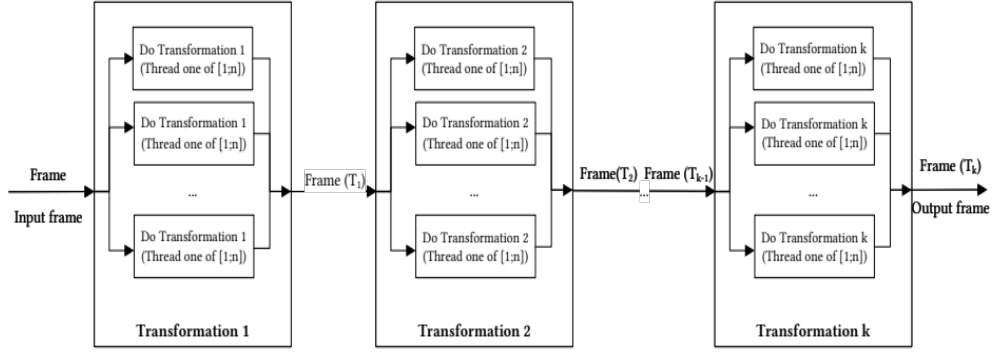


Figure 2: The proposed user-space processing architecture.

2.5. Primitive Video Transformations

Since the target hardware is a low-cost SBC, the methodological focus of this study is limited to primitive image transformations that are representative, computationally meaningful, and implementable under constrained resources.

The first transformation is Gaussian blurring, which performs local smoothing by convolving the image with a Gaussian kernel. This operation reduces high-frequency noise and is frequently used as a preprocessing step in vision pipelines [22].

The second transformation is Laplacian-based contour extraction [23]. The Laplacian operator is a second-order differential operator that emphasizes regions of rapid intensity variation and is therefore suitable for highlighting edges and structural transitions in the image. In the present study, a discrete 3x3 approximation of the Laplacian operator is used. The result is stored as a grayscale frame for subsequent analysis.

The third transformation is histogram computation, which determines the distribution of pixel intensities in the frame. Histograms are widely used as compact descriptors of image structure and may support subsequent tasks such as object localization, motion characterization, or scene-change analysis [24].

These operations were selected because they represent common low-level vision procedures while remaining sufficiently lightweight for controlled evaluation on SBC hardware.

2.6. Experimental Platform and Implementation Details

The experimental prototype was implemented using an Orange Pi Zero 2W SBC equipped with an Allwinner H618 processor (quad-core ARM Cortex-A53 at 1.5 GHz) and 4 GB RAM [25]. The operating system was Ubuntu 22.04.4 LTS with Linux kernel 6.1.31-sun50iw9. Persistent storage was provided by a 64 GB Class 10 microSD card.

Video acquisition was performed using a Logitech C170 USB camera configured for a frame resolution of 640 × 480 pixels with YUYV color encoding [26]. The application was compiled directly on the SBC using g++ 12.3.0 with support for the C++14 standard [27]. The computer-vision functionality was implemented using OpenCV 4.10.

The experimental application was developed to evaluate the compliance of the hardware-software configuration with the real-time criterion defined in Subsection 3.1. The implemented transformation chain consisted of three stages: Gaussian blurring, Laplacian filtering, histogram computation. The Laplacian stage employed the OpenCV filter2D function with the 3x3 mask:

$$K_{Laplas} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (5)$$

Histogram computation was implemented using calcHist, and Gaussian smoothing was implemented using GaussianBlur with a 3x3 kernel, that shown below at (5). The application allows the number of execution threads to be specified for transformations that admit partitioned processing. The measured

output metric is the average achieved FPS, displayed during execution and used as the primary performance indicator. Two execution modes were considered. In the first mode, all three transformations were executed sequentially, yielding a baseline implementation. In the second mode, the Laplacian stage was parallelized across CPU cores, while the remaining two stages were executed sequentially. This hybrid configuration was selected to assess whether selective parallelization of the most computationally intensive stage is sufficient to satisfy the real-time criterion on the target platform.

3. Results

Two execution modes were evaluated. In the first mode, all three transformations were executed sequentially. In the second mode, a hybrid parallel-sequential configuration was applied, in which the Laplacian filtering stage was parallelized across the available CPU cores of the SBC, whereas histogram computation and Gaussian blurring remained sequential. This design made it possible to assess whether selective parallelization of the most computationally intensive transformation is sufficient to achieve real-time performance on the target platform.

In the fully sequential configuration, the developed application achieved an average throughput of 24 FPS, as illustrated in (Figure 3a). This value corresponds to an average processing time of approximately 41.7 ms per frame, which exceeds the admissible threshold of 40 ms per frame for a 25 FPS stream. Therefore, the purely sequential implementation does not satisfy the adopted real-time requirement.

In the hybrid parallel-sequential configuration, the application achieved an average throughput of 28 FPS, as shown in (Figure 3b). This corresponds to an average processing time of approximately 35.7 ms/frame, which is below the admissible 40 ms limit. Under the experimental conditions of this study, the hybrid configuration therefore satisfies the real-time criterion.

The reported results were obtained using the proposed software architecture based on the Chain of Responsibility and ThreadPool patterns. However, not all primitive transformation stages benefit from parallelization; for some stages, thread-management overhead may outweigh the performance gain.

A comparison of the two execution modes shows that selective parallelization of the Laplacian stage increased throughput from 24 FPS to 28 FPS. This corresponds to a relative improvement of approximately 16.7%. The obtained result indicates that even partial parallelization of a single transformation stage can produce a measurable gain in end-to-end processing performance on a low-cost SBC platform.

The results also show that the proposed architecture is suitable for lightweight edge-side preprocessing tasks, particularly those based on primitive image transformations. At the same time, the observed performance margin remains limited, which suggests that more computationally demanding operations would likely require either additional optimization, lower-resolution input, hardware acceleration, or partial offloading to more powerful nodes within a distributed edge architecture. Thus, the experimental findings support the use of low-cost SBC-based cameras as compact edge nodes for simple real-time preprocessing tasks, while also indicating the practical limits of such devices under constrained computational resources.

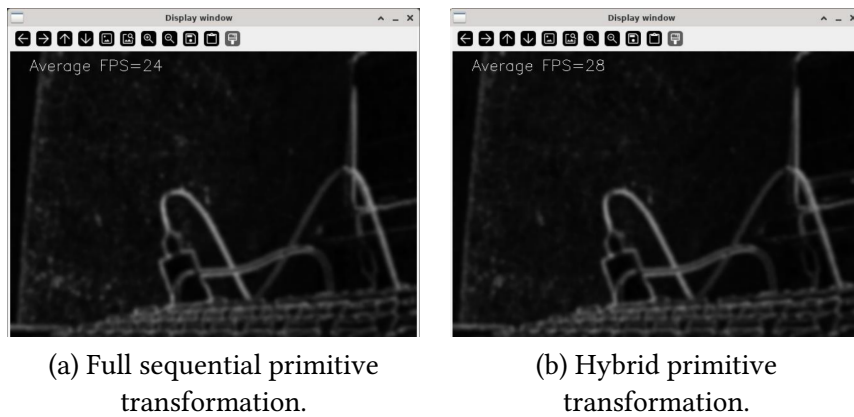


Figure 3: Comparison of sequential and hybrid frame-processing modes.

Table 1

Frequency of Special Characters

Execution mode	Average FPS	Processing time per frame (ms)	Real-time criterion
Sequential	24	41.7	Not satisfied
Hybrid	28	35.7	Satisfied

4. Discussion

The present study has several limitations. The evaluation was conducted on a single SBC platform, with one USB camera configuration, one input resolution (640×480), and a limited set of three primitive image transformations. The experiments did not examine energy consumption, thermal effects, long-duration stability, and network transmission performance.

5. Conclusion

This paper investigated a low-cost SBC-based edge camera for real-time video preprocessing. The experimental results showed that the sequential implementation achieved 24 FPS, whereas the proposed hybrid parallel-sequential approach increased performance to 28 FPS, thereby satisfying the adopted real-time criterion for a 25 FPS stream. These findings indicate that low-cost SBC platforms are suitable for lightweight edge-side video preprocessing, although more computationally demanding tasks would require additional optimization or more capable hardware.

6. CRediT Author Statement

OB: Conceptualization, Methodology, Software, Data curation, Writing– Original draft preparation. KM: Validation, Visualization, Writing – Reviewing and Editing.

7. Funding

These results were obtained as part of the project “Develop a hardware-software system for video data processing in different frequency ranges using edge computing” (State registration No. 0126U000623; <https://dir.ukrintei.ua/view/rk/3afdf57cf0459c15c1ca946d8e2a1a5a>), funded by the NAS of Ukraine. Additional support was provided by the R&D projects “To develop theoretical foundations and a functional model of a computer for processing complex information structures” (State registration No. 0124U002317; <https://nrat.ukrintei.ua/searchdoc/0124U002317/>) and “Develop Means of Supporting Virtualization Technologies and Their Use in Computer Engineering and Other Applications” (State registration No. 0124U001826; <https://nrat.ukrintei.ua/en/searchdoc/0124U001826>), funded by the NAS of Ukraine.

8. Data Availability Statement

The data and source code supporting the findings of this study are available from the authors upon reasonable request.

Declaration on Generative AI

During the preparation of this work, the authors used gpt-oss-120b (OpenAI’s open-weights LLM), deployed locally on institutional hardware, solely for grammar and spelling checks.

References

- [1] S. U. Hiremath, S. P. Baannadabavi, S. Kabbin, S. Shirakol, Edge detection algorithm using pi computer, *Bonfring International Journal of Research in Communication Engineering* 6 (2016) 79–82. doi:10.9756/BIJRCE.8206.
- [2] M. Yildirim, O. Karaduman, H. Kurum, Real-time image and video processing applications using raspberry pi, in: 2022 IEEE1stIndustrial Electronics Society Annual On-LineConference(ONCON), IEEE, kharagpur, India, 2022, p. 1–6. doi:10.1109/ONCON56984.2022.10127034.
- [3] M. M. Asri, H. Maulana, N. P. Hasbi Wisudawan, S. Murnani, Developing smart security camera with lbph method and haar-cascade based on raspberry-pi and telegram, in: 2023 International Workshop on Artificial Intelligence and Image Processing (IWAIPP), IEEE, Yogyakarta, Indonesia, 2023, p. 396–401. doi:10.1109/IWAIPP58158.2023.10462860.
- [4] J. Ijaradar, J. Xu, A cost-efficient real-time security surveillance system based on facial recognition using raspberry pi and opencv, *Current Journal of Applied Science and Technology* (2022) 1–12. doi:10.9734/cjast/2022/v41i531665.
- [5] S. C. Gaddam, N. V. K. Ramesh, H. Dhanekula, Face recognition based attendance management system with raspberry pi 2 using eigen faces algorithm, 2016. URL: <https://api.semanticscholar.org/CorpusID:31248117>.
- [6] I. B. Mustaffa, S. F. B. M. Khairul, Identification of fruit size and maturity through fruit images using opencv-python and raspberry pi, in: 2017 International Conference on Robotics, Automation and Sciences (ICORAS), IEEE, Melaka, 2017, p. 1–3. doi:10.1109/ICORAS.2017.8308068.
- [7] V. Kaverinskiy, I. Chaikovskiy, A. Mnevets, T. Ryzhenko, M. Bocharov, K. Malakhov, Scalable clustering of complex ecg health data: Big data clustering analysis with umap and hdbscan, *Computation* 13 (2025) 144. doi:10.3390/computation13060144.
- [8] A. A. Litvin, V. Y. Velychko, V. V. Kaverinskiy, Method of information obtaining from ontology on the basis of a natural language phrase analysis, in: CEUR Workshop Proceedings, volume 2866, CEUR-WS, Kyiv, Ukraine, 2020, p. 323–330. URL: https://ceur-ws.org/Vol-2866/ceur_322_330_litvin_velichko.pdf.
- [9] A. A. Litvin, V. Y. Velychko, V. V. Kaverinskiy, A new approach to automatic ontology generation from the natural language texts with complex inflection structures in the dialogue systems development, in: CEUR Workshop Proceedings, volume 3501, CEUR-WS, Kyiv, Ukraine, 2022, p. 172–185. URL: <https://ceur-ws.org/Vol-3501/s16.pdf>.
- [10] A. Litvin, O. Palagin, V. V. Kaverinskiy, K. S. Malakhov, Ontology-driven development of dialogue systems, *South African Computer Journal* 35 (2023) 37–62. doi:10.18489/sacj.v35i1.1233.
- [11] O. V. Palagin, M. G. Petrenko, V. V. Kaverinskiy, K. S. Malakhov, A method for enhancing the efficiency of rdf/xml-structure processing in the apache jena semantic web framework, *Cybernetics and Systems Analysis* 61 (2025) 469–486. doi:10.1007/s10559-025-00784-w.
- [12] O. V. Palagin, M. G. Petrenko, K. S. Malakhov, Challenges and role of ontology engineering in creating the knowledge industry: a research-related design perspective, *Cybernetics and Systems Analysis* 60 (2024) 633–645. doi:10.1007/s10559-024-00702-6.
- [13] V. Kaverinskiy, G. Bagliuk, Z. Sukhenko, D. Verbylo, Micro-scale fem study of inclusion-matrix interactions in al–si composites under compression, *Modelling and Simulation in Materials Science and Engineering* 34 (2025) 035003. doi:10.1088/1361-651X/ae4f04.
- [14] G. Bagliuk, V. Kaverinskiy, Numerical simulation of the powder briqueted modifiers dissolution at in-mold modification of cast iron, *International Journal of Metalcasting* 20 (2026) 282–292. doi:10.1007/s40962-025-01615-6.
- [15] ITU-R, Enhanced Wide-Screen PAL TV Transmission System (the PALplus System), Recommendation ITU-R BT.1197-1, 1998. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1197-1-199802-W!!PDF-E.pdf.
- [16] O.V.Bahatskyi, Software realization of intellectual video camera, *Visnyk of Vinnytsia Politechnical Institute* 140 (2018) 60–66. doi:10.31649/1997-9266-2018-140-5-60-66.
- [17] V. Popovic, K. Seyid, O. Cogal, A. Akin, Y. Leblebici, Design and Implementation of Real-Time Multi-Sensor Vision Systems, Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-59057-8.

- [18] J. Zhang, D. Tao, Empowering things with intelligence: A survey of the progress, challenges, and opportunities in artificial intelligence of things, *IEEE Internet of Things Journal* 8 (2021) 7789–7817. doi:10.1109/JIOT.2020.3039359.
- [19] M. H. Schimek, B. Dirks, H. Verkuil, M. Rubli, Video for linux two api specification: Revision 0.24, 2024, PDF document. Accessed: 2026-04-16. URL: <https://api.eestar.com/article/share/2023/06/730a5202306251916539796.pdf>.
- [20] X. Qi, Design pattern improvement of chain of responsibility, *Computer Engineering* 36 (2010) 56–57. doi:10.3969/j.issn.1000-3428.2010.10.018.
- [21] D. Puyda, Scheduling: A fast c++ thread pool implementation capable of executing task graphs, *Computer systems and information technologies* (2025) 73–78. doi:10.31891/csit-2025-1-9.
- [22] E. Reinhard, G. Ward, S. Pattanaik, P. Debevec, High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting, Morgan Kaufmann, 2006. doi:10.1016/B978-0-12-585263-0. X5000-3, cited pages: 233–234.
- [23] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, IEEE Comput. Soc, Kauai, HI, USA, 2001, pp. I–511–I–518. doi:10.1109/CVPR.2001.990517.
- [24] D. G. Lowe, Distinctive image features from scale-invariant keypoints, *International Journal of Computer Vision* 60 (2004) 91–110. doi:10.1023/B:VISI.0000029664.99615.94.
- [25] Xunlong Software Co., Ltd., Orange Pi Zero 2W User Manual, 2023. Version 1.0, 2023-09-14; online; accessed 2026-04-16. URL: https://www.orangepi.org/orangepiwiki/index.php/Orange_Pi_Zero_2W.
- [26] Logitech, Getting started with Logitech Webcam C170, 2012. Logitech Support Documentation; online; accessed 2026-04-16. URL: <https://www.logitech.com/assets/46920/webcam-c170-quickstart-guide.pdf>.
- [27] ISO/IEC, ISO/IEC 14882:2014 Information technology – Programming languages – C++, 2014. International Standard, Edition 4. URL: <https://www.iso.org/standard/64029.html>.