Palindrome.js: 3D monitoring for distributed systems visual analysis

Jonathan Rivalan, Long H Ngo, Etienne Leclercq, Anastasios Zafeiropoulos

Palindrome.js: 3D monitoring for distributed systems visual analysis

Jonathan Rivalan
SMILE, France
jonathan.rivalan@smile.fr

Long H. Ngo
SMILE, France
long.ngo@smile.fr

Etienne Leclercq
SMILE, France
etienne.leclercq@smile.fr

Anastasios Zafeiropoulos
School of Electrical and Computer Engineering,
National Technical University of Athens, Greece

***Abstract*—While the Cloud-Edge continuum scheduling domain is well studied, and proposes numerous approaches, from high-level modeling to dynamic scheduling, monitoring means are rarely debated. Solving the ever-growing number of metrics, which can exist in different units and update frequencies, support various system scales and topologies, relies upon classical 2D features (made of charts, gauges, maps and lists) without proposing visual abstractions directed towards multi-layered architectures and distributed infrastructures. Moreover, the need for monitoring in modern Edge systems and applications exceeds system metrics, as other informational dimensions can be considered, such as the ones collected through web services or IoT devices and sensors in the user space.**

We propose with Palindrome.js a 3D monitoring probe, which enables multidimensional visual modeling and analysis. Through sets of metrics called layers, Palindrome.js builds in 3D a structured visual abstraction for any problem modeled with metrics or KPIs. We present in this article the solution initial background, the related works, its design principles, discuss experimental results, and conclude with our observations and future outcomes.

***Index Terms*—3D monitoring, compute continuum, visual analysis**

I. Introduction

The rise of new virtualization technologies, known as containers, the development of IoT devices infrastructures and the Cloud-native trend, a recent approach to telecom appliances execution, deployment, and management, had for effect a dramatic increase in metrics production. Legacy monolithic applications became sets of containers [1], where each container is a metrics producer that can spawn or stop at any given point in time [2].

DevOps answer to metrics collection, visualization, and analysis, is made of three main components. Specialized databases, known as time series databases (TSDBs) [3], dashboards, and alerting systems.

Dashboards present metrics and can be configured manually or programmatically to provide views made of graphs. When programmed, these views support dynamicity through manually or randomly selectable options to either drill down in architecture informational levels or consult different tenants. Sometimes the switch between views or dashboards can be automated, either following a time-based rule (as within a timer) or when connected to alerting systems.

With this approach, information is scattered over different views and graphs and requires manual settings, specialized programming and constant interaction to retrieve high and low level metrics values or states.

Considering multi-dimensional representation, the 2D nature of monitoring dashboards and alerting tools limit them into providing a homogeneous visualization of multiple informational dimensions, such as cloud and edge nodes, metrics with different units, systems with different metrics or states made of sub-states.

Also and last, as monitoring concerns evolve towards a mix of system metrics (which can be distributed over different servers, different providers or in different infrastructure layers), and higher level KPIs such as price and energy consumption, the ability to visually access all information dimensions at once becomes critical.

Goal. Our goal in conceiving and developing Palindrome.js falls into two main objectives. First, to allow a way for multidimensional display and analysis. As systems and applications gain in distribution complexity, monitoring tenants instantly while preserving a metric grain level resides on the ability of the sub-system to expose global information as a multidimensional visual representation. Second, to provide a pre-arranged visual representation enabling computer vision analysis. Detecting differences in behaviors between heterogeneous metrics sets can be tedious at large scale and in real-time in 2D. Using computer vision may help in dynamically discovering features, given that the sub-system exposes a pre-arranged 3D visual output.

Approach. We explored with Palindrome.js the ability to reach these goals, by 1. supporting a minimal metrics description schema, embedding a three levels (min/med/max) range parameter for each metric grouped as layers, and by 2. proposing a versatile 3D visualization engine, which allows through configuration to model generic or specific use cases, for human or machine oriented visual outputs. The description interface is implemented in JSON, the logic in Javascript, and the 3D engine in Three.js[1].

The rest of the paper is organized as follows: Section II introduces the related works; Section III presents the design principles; Section IV shows the demonstration-related experiments; Section V concludes the paper while listing the contributions and future works.

II. Related works

The notion of multidimensional monitoring appears in the literature in the early 2000s [4], [5], as a means to reduce maintenance costs and address anomaly detection.

In [6] and [7] authors consider multidimensional information from a single computer system, such as system metrics, system reports, and logs. A time-sensitive analysis

[1] https://threejs.org/

is then performed to detect possible upcoming computer failures (e.g. disk errors). While this study introduces the concept of multidimensional definition, it does not delve into potential applications or explore possibilities associated with 3D visualization.

A generic modern approach for classical monitoring is based on the Prometheus[2], Loki[3], and Grafana[4] stack [8], [9]. Prometheus provides the metrics retrieval, Loki handles the logs retrieval, and Grafana displays the combined 2D visualization for metrics and logs. It differs from our approach by the need to define both display configuration and queries in specialized DSLs (promQL[5] for Prometheus and Grafonnet[6] for Grafana). On the other hand, Palindrome.js lacks support for metrics collection compared to Prometheus.

Some works extended the Prometheus, Loki, and Grafana stack to support automated information retrieval. In [10], the authors propose a complete architecture for alerts retrieval based on metrics and logs analysis. In regards, Palindrome.js supports only part of the metric level use case as a single javascript-based client and could be seen as a possible extension to the proposed glass dashboard.

Supervision-related works provide few examples of 3D monitoring applications, mostly oriented towards either a specific use case or a specific domain [11]–[14]. A notable example is network topology visualization [11], which supports network hierarchy visualization with dynamic updates through a Node.js[7] server.

The use of OpenCV[8] [15]–[18] in monitoring automation follows the same trend of use-case-specific implementations, with a focus in the literature on observing real-life objects (persons, vehicles) to trigger alerts or custom instructions [19]–[21]. In [22], authors propose a more generic approach to automated visual monitoring by coupling real-life meters to OpenCV. Each number of the meter LCD screen is transformed as a digital value, later usable in monitoring and alerting systems. With Palindrome.js we orient the OpenCV use towards color detection, rather than characters or shapes, to enable a seamless interaction between human operators, who can visually supervise, and services, who can benefit from the digital output offered by computer vision.

III. Design Principles

We develop in this section the main design principles for our solution. It depicts the relation to metrics and how they are used to build a 3D representation, the color-based output core feature, the support for a modular, extensive, and configurable architecture, the library accessibility through multiple testing environments and lastly its integration within the DevOps ecosystem.

Metrics first orientation. In Palindrome.js we intend to expose a minimalist API so users can easily implement new use cases. This API exists through a JSON structure enabling metrics and metrics layers descriptions. Metrics layers are sets of metrics; use cases can be implemented through one or more layers depending on their complexity or scale. In a layer, each metric is described as a data point with its ranges (min, med, max). This structure is meant to discover states made of sub-states: as each metric shares its position within its ranges, so does the layer hosting it (Eq.1, page 2), by displaying a range-based color for all its children metrics (Eq.2, page 2). The metrics set (layer) $L(i)$ calculated at Tn is defined by the following equation:

$$L(i)Value(Tn) = \sum_{m \in L(i)} m(val)(Tn)/m(max)(Tn) \quad (1)$$

The current range of a metrics set (layer) is then defined as follows:

$$L(i)\text{Range}(Tn) = \begin{cases} \text{low} & \text{if } L(i)m(\text{min/max})(Tn) \leq L(i)\text{Value}(Tn) \leq L(i)m(\text{med/max})(Tn) \\ \text{high} & \text{if } L(i)m(\text{med/max})(Tn) \leq L(i)\text{Value}(Tn) \leq L(i)m(\text{max/max})(Tn) \end{cases} \quad (2)$$

Layers can then reflect the real-time state of one or more metric sets, without this state being described previously (Fig. 1, page 2). The function illustrated in Fig. 1 provides sample data for Palindrome.js showcasing how metrics can be organized and presented in a 3D visual representation. Specifically, it returns a configuration object for 3 system metrics in one layer, i.e. CPU, RAM, and storage, each specified with labels, units, minimum, medium, maximum, and current values.

```
export function demoConfig() {
    return {
        "systemMetrics": {
            "metrics": {
                "cpu": {
                    "label": "Available CPU",
                    "unit": "cycle / seconds",
                    "min": 100,
                    "med": 500,
                    "max": 1000,
                    "current": 390
                },
                "ram": {
                    "label": "Available RAM",
                    "unit": "GB",
                    "min": 2,
                    "med": 64,
                    "max": 128,
                    "current": 78
                },
                "storage": {
                    "label": "Available Storage",
                    "unit": "GB",
                    "min": 102,
                    "med": 512,
                    "max": 1024,
                    "current": 450
                }
            },
            "layer": {
                "systemMetrics-layer": {
                    "label": "System metrics"
                }
            }
        }
    }
};
```

Fig. 1: Example of a Palindrome.js use case JSON data structure, describing three metrics within a single layer. (Javascript)

Color based output. As metrics values evolve between the low and high ranges, Palindrome.js outputs a color-based

[2]https://prometheus.io
[3]https://github.com/grafana/loki
[4]https://grafana.github.io
[5]https://prometheus.io/docs/prometheus/latest/querying/basics
[6]https://grafana.github.io/grafonnet
[7]https://nodejs.org/
[8]https://opencv.org/

```
import {demoConfig} from '../data/demoConfig';
export const Conf = createPalindrome.bind({});
Conf.args = {
    statusColorLow: '#e8e49f',yan
    statusColorMed: '#8400ff',
    statusColorHigh: '#FF0000',
    data: demoConfig()
};
```

Fig. 2: Example of a Palindrome.js use case being instantiated with custom configuration options. (Javascript)

abstraction for these ranges. The layer value is transformed in a color code belonging to a predefined color scale (made of either 2 or 3 colors depending on the user configuration). As such, the library can visually represent ranges not previously expressed. Horizontally between metrics in a set, vertically between sets of metrics.

Modular configuration. Since metrics values are not expressive of their quality level (*e.g.* where the *0* value can be either good or bad) or their behavior (e.g. binary or non-binary, ascending or descending, being discrete or having free values), Palindrome.js provides a rich API for visual configuration. Each configurable parameter is made of a JavaScript object that can be reused over environments. Visual configuration can then be embedded directly into a use case, when it is being instantiated, or configured in real-time through a web-based interface (Fig. 2, page 3). In Fig. 2, this code creates a configuration object to instantiate a Palindrome.js instance, i.e. colors for different status levels (low, medium, high).

Accessibility. We provide two web-based environments for development and testing purposes: the ”dev” environment and the ”Storybook” environment (Fig. 3, page 3). While the former exposes a limited set of configuration options, the latter provides the full Palindrome.js API as interactive options within a graphical user interface (GUI). Users and developers can set and test their configurations prior to production use. The code is also made available as a docker image, enabling headless execution. The library, along with the testing environments, is provided publicly as an open-source project under the Apache 2.0 license [23].

DevOps ecosystem. The DevOps methodology is strongly oriented towards proposing and supporting open source tools, to allow the developers community to participate in their design, extension and customisation towards users needs or customers use cases. To cope with this trend, and provide Palindrome.js as a *de facto* DevOps component we chose to implement it as an extension to the well-known Grafana dashboard solution [24]. In this extension, Palindrome.js configuration parameters are either editable, for the visual aspects, or generated from the TSDBs queries, to provide the data structure. While users define their queries in the relevant Grafana plugin panel, the Palindrome.js data structure is built automatically and the resulting 3D model displayed instantly (Fig. 4, page 3). This is made possible through an additional domain specific language (DSL) we introduced, which enables passing extra parameters to typical queries making Palindrome.js attributes explicit, such as layers names and metrics ranges (Fig. 5, page 4).

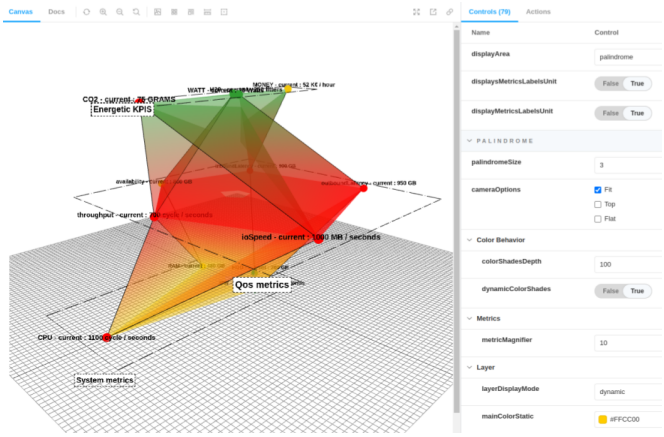


Fig. 3: Example of a Palindrome.js configuration session in Storybook. API parameters are made available as interactive HTML inputs (on the right) to ease configuration setup for custom use cases.

(a)

(b)

Fig. 4: Example of 3D monitoring within a Grafana dashboard. The 3D model of observed metrics can be zoomed out (a) for an overall view, or zoomed in (b) to grasp additional details.

IV. Experiments

We study within the below experiment the ability to support synergetic Edge to Cloud orchestration by coupling Palindrome.js, serving as a 3D visualization probe, and OpenCV, for the computer vision effort. Synergetic scheduling [25] aims at providing unified scheduling decisions out of single system states or multiple systems of systems states (e.g. such as Edge-Cloud, CPSoS or compute continuum architectures). While Palindrome.js can leverage the display for states comprised of multidimensional information, synergetic scheduling expects an automated state detection to assert

```
disk_io{device="nvme"} #layer: SSD, ranges: [0, 50, 100]
```

Fig. 5: Example of a PromQL query with custom Palindrome.js parameters. The resulting *disk_io* metric will be displayed within the *SSD* layer.

with dynamic scheduling decisions in layered and distributed contexts.

We use OpenCV to explore the possibility of automating visual states retrieval from Palindrome.js. The OpenCV library aims to support the identification of colors and their respective volumes. Extracted information from side or top views serves as an input for taking further informed decisions. By understanding the colors and their volumes, we can extract meaningful insights and potentially apply them to various applications and contexts. Figure 6 illustrates the general architecture schema of our approach combining 3D monitoring and computer vision to achieve multidimensional scheduling decisions.

Our OpenCV API algorithm is made of two parts:

1) colors and system states definition (Algorithm 1).
2) colors based states detection (Algorithm 2).

More specifically, Algorithm 1 presents two dictionaries definitions related to colors ranges and respective system states. Algorithm 2 displays the color volume calculation as well as conditions to indicate system states in different scenarios. In this case, encountering the "red" color, will determine an "Exit" state (which excludes other states), while encountering "green" or "yellow" colors will determine either "Running" or "Idle" states (which are compatible).

Algorithm 1: Colors ranges and states definitions

Define $color\_bounds$ dictionary holding HSV color format range tuples for observed colors `# Green, Yellow, Red`
Define $color\_states$ dictionary holding states descriptions for observed colors `# Running, Idle, Exit`

A. Single state detection, top view.

In this scenario, we consider a single state identification for Palindrome.js representation. For this particular case, we set within one layer, five metrics with three ranges: low, medium, and high. Corresponding colors can be assigned to the layer based on its metrics values range. Green represents low values, yellow represents medium values, and red represents the high ones. Each color can describe a predefined state of the system. OpenCV library is used to detect the color and its outline within the representation. Once the color has been identified, system state information is collected (*e.g.* idle or busy). This information can then be used to make decisions about whether the system can host more services or not.

B. Multi-state detection without threshold, side view

OpenCV is used to execute outline detection and color recognition on various frames of Palindrome.js, captured

Algorithm 2: Colors volumes based states detection

Input: $color\_bounds$ and $color\_states$ dictionaries
Output: $volume\_state$ list, used for visual decoration
Data: Palindrome.js Tn pixels matrix

Assign the current Palindrome.js pixels matrix to the $total\_colors\_pixels$ variable
for $name, bounds$ *in* $color\_bounds.items()$ **do**
    Calculate the $name$ color $bounds$ percentage from $total\_colors\_pixels$, and assign it to the $color\_volume[\$name]$ dictionary entry
if $color\_volume["green"] == 100$ **then**
    $flag \leftarrow 1$ `# Running`
else if $color\_volume["yellow"] > 1$ *and* $color\_volume["red"] == 0$ **then**
    $flag \leftarrow 2$ `# Idle`
else if $color\_volume["red"] > 1$ **then**
    $flag \leftarrow 3$ `# Exit`
for $name, volume$ *in* $color\_volume.items()$ **do**
    if $flag == 1$ *or* $flag == 2$ **then**
        Append $current\_time$, $name$, $volume$ and $color\_states[\$name]$ variables to the $volume\_state$ list
    else if $flag == 3$ **then**
        Append $current\_time$, $name$, $volume$ and $color\_states["red"]$ variables to the $volume\_state$ list

from different viewpoints. Our approach involves applying OpenCV processing capabilities to detect the outlines of the frame and identify the colors composing it. This technique allows for analyzing Palindrome.js from multiple angles and perspectives, providing a comprehensive understanding of its properties and characteristics (Fig. 7 and 8, page 5). Fig. 7 and 8 depict an example of colors detection on a Palindrome.js model with according system's states.

One can see in these figures, 4 distinct colors, representing 4 different states, of the given multivariate problem. Each layer of the 3D object corresponds to its own unique state. Consequently, this observation poses a significant challenge for the local scheduler. We consider the following approaches to select the whole system's state:

- Predominant color: In this approach, we identify the color that appears the most frequently or with the biggest volume among the layers, and use it as the overall state of the system.
- State hierarchy: In some cases, the different states represented by the colors may have a hierarchical relationship, where one state is considered more severe than another. In this approach, we use the most severe state among the layers as the overall state of the system.

C. Multi-state detection with threshold

We conduct simulations for multiple systems states detection within a use case supporting 3 distinct scenarios: (a) Ready to go : in this scenario, a system is considered

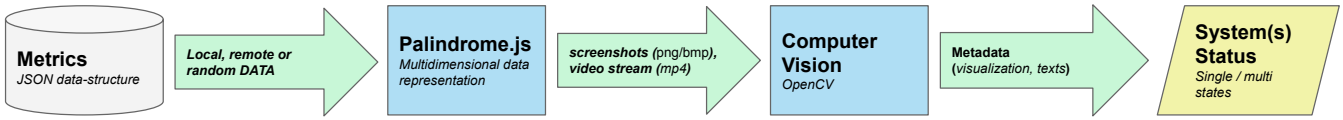


Fig. 6: Event-driven architecture combining 3D monitoring and computer vision to automate scheduling decisions.

	Color	Volume (%)	State
0	red	29	System is locked down
1	orange	17	Full services are running
2	yellow	46	Service has been deployed but is not running
3	green	8	System is ready to host services

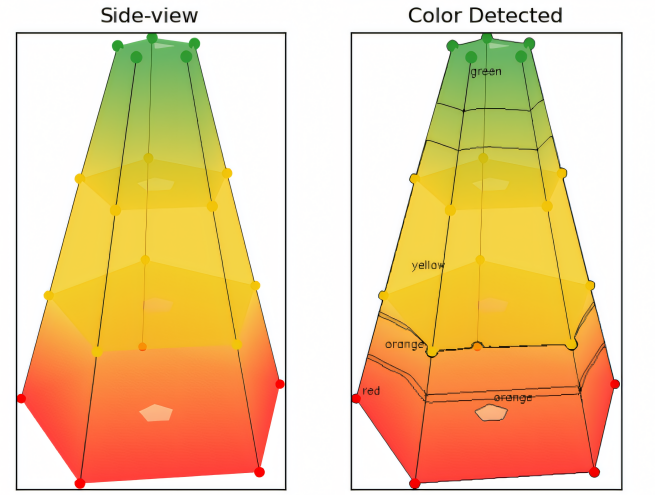


Fig. 7: Colors identification from the *side* view of Palindrome.js alongside the according volumes and states.

	Color	Volume (%)	State
0	red	24	System is locked down
1	orange	16	Full services are running
2	yellow	56	Service has been deployed but is not running
3	green	3	System is ready to host services

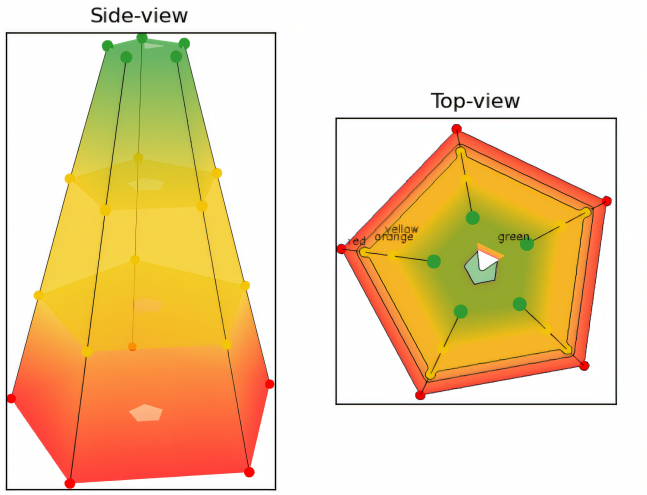


Fig. 8: Colors identification from the *top* view of Palindrome.js alongside the according volumes and states.

online only when its metrics values surpass the computed medium threshold. If the entirety of the systems are online, they are thereby in a working state; (b) Not good to go : in this scenario, the operational status of one or more systems transitions from an online state to a sleeping state due to their metrics values falling below the medium threshold. In such conditions, they are considered in a non-blocking error state; (c) Exit (Fig. 9, page 6): within this context, should the metrics values fall below low thresholds, the whole platform is considered in a blocking error state.

Multiple systems undergoing such offline states will result in the exit of all these systems. illustrates our use cases from the ”Ready to go” scenario to the ”Not good to go” and vice versa. Fig. 9 describes the ”exit” scenario, when any given system falls to red, all the systems exit.

In conclusion, our experiments aimed to explore the automation of visual state retrieval from Palindrome.js using OpenCV to support synergetic scheduling demonstration. We leveraged OpenCV to identify colors and their respective volumes from Palindrome.js representation, showcasing our ability to make informed decisions based on the extracted visual information. Through various scenarios and use cases, we demonstrated the effectiveness of our approach in detecting and categorizing system states.

This integration provides a promising foundation for unified scheduling decisions, enabling comprehensive analysis and monitoring in decentralized, distributed or both contexts, such as the ones comprising the compute continuum.

V. Conclusion and Future Works

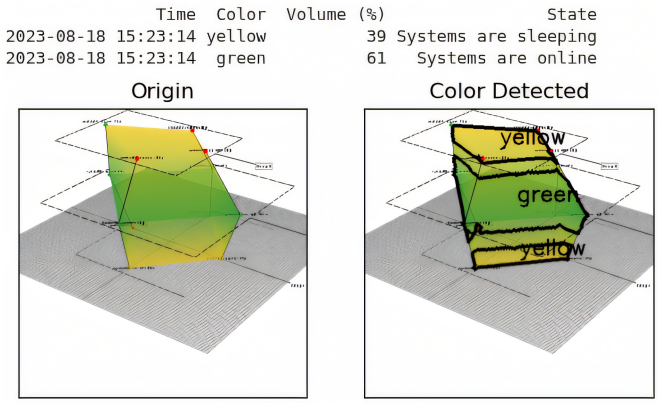
In this article, we introduced our open-source 3D monitoring library Palindrome.js, presented its background and design principles and discussed experimental implementations coupling our library with OpenCV.

We demonstrated that we could, through multidimensional monitoring and computer vision, automatically solve system states detection problems with a limited understanding and definition of the overall constraints. We went up to define a naive color-based threshold to detect a good system of systems state from a bad one, along with its intermediary states.

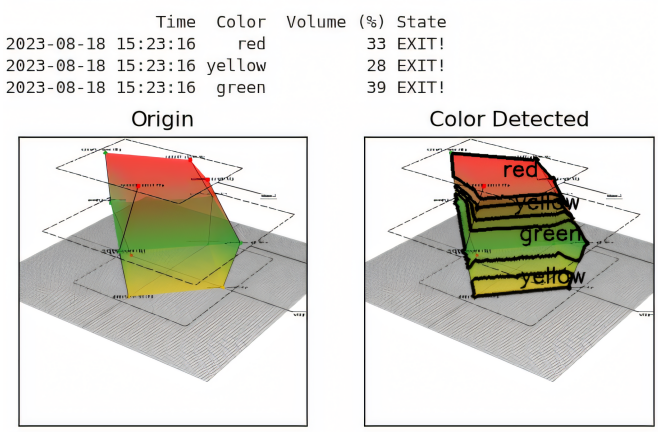
We believe these initial experiments demonstrate the interest and possible usage for 3D color-based information modeling and automated visual analysis.

Further research is needed to assert the probe’s ability to scale up: as 3D representations will grow in complexity, colors mixes may need some additional attention to provide relevant informational outputs from analytics modules. In our observations, using predefined configurations could be considered to limit the solution space and improve the computer vision assisted system(s) states retrieval.

Future integration of the library with the Web of Things (WoT) standards is considered to provide a generic support



(a)



(b)

Fig. 9: Colors identification for the ”exit” use case. The initial T_n situation (a) where the distributed system is online; the T_{n+1} one (b), where an anomaly is visually detected.

for Internet of Things (IoT) 3D monitoring, enabling seamless integration of systems, devices and sensors metrics.

ACKNOWLEDGMENT

This work has been funded by the European Union’s Horizon Europe research and innovation program under grant agreement No. 101070487 (NEPHELE).

REFERENCES

[1] S. Eski and F. Buzluca, “An automatic extraction approach: Transition to microservices architecture from monolithic application,” in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, 2018, pp. 1–6.

[2] T. Watts, R. G. Benton, W. B. Glisson, and J. Shropshire, “Insight from a docker container introspection,” *Hawaii International Conference on System Sciences 2019*. [Online]. Available: https://par.nsf.gov/biblio/10093193

[3] N. Sabharwal, P. Pandey, N. Sabharwal, and P. Pandey, “Working with prometheus query language (promql),” *Monitoring Microservices and Containerized Applications: Deployment, Configuration, and Best Practices for Prometheus and Alert Manager*, pp. 141–167, 2020.

[4] B. Bajic, “Multidimensional monitors for hydroelectric power plants,” in *Proceedings of the Hydro 2002 Conference, Kiris, Turkey*, 2002, pp. 4–7.

[5] V. Gudkov and J. E. Johnson, “Multidimensional network monitoring for intrusion detection,” in *Unifying Themes in Complex Systems IV: Proceedings of the Fourth International Conference on Complex Systems*. Springer, 2008, pp. 291–302.

[6] M. Krol and J. Sosnowski, “Multidimensional monitoring of computer systems,” in *2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*. IEEE, 2009, pp. 68–74.

[7] J. Sosnowski, “Tracing anomalies in computer systems,” in *Proc. 41st International Conference on Dependable Systems and Networks (DSN)(Ed. Muppala JK), IEEE Computer Society, ISBN*, 2011, pp. 978–1.

[8] M. Chakraborty and A. P. Kundan, “Grafana,” in *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software*. Springer, 2021, pp. 187–240.

[9] Simili, Emanuele, Stewart, Gordon, Roy, Gareth, Skipsey, Samuel, and Britton, David, “A hybrid system for monitoring and automated recovery at the glasgow tier-2 cluster,” *EPJ Web Conf.*, vol. 251, p. 02047, 2021. [Online]. Available: https://doi.org/10.1051/epjconf/202125102047

[10] E. Bautista, N. Sukhija, and S. Deng, “Shasta log aggregation, monitoring and alerting in hpc environments with grafana loki and servicenow,” in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, 2022, pp. 602–610.

[11] Y. Wang, Y. Li, Q. Zhang, J. Zheng *et al.*, “Large scale network topology visualization system based on three. js,” in *2016 International Conference on Artificial Intelligence: Technologies and Applications*. Atlantis Press, 2016, pp. 152–155.

[12] Y. Zhang, P. Teng, M. Aono, Y. Shimizu, F. Hosoi, and K. Omasa, “3d monitoring for plant growth parameters in field with a single camera by multi-view approach,” *Journal of agricultural meteorology*, vol. 74, no. 4, pp. 129–139, 2018.

[13] W. Shan, W. Wan, A. Chen, L. Ren, J. Sun, M. Fang, and Y. Zhan, “3d warehousing: Enabling intelligent warehousing visualization based on three. js,” in *International Conference On Signal And Information Processing, Networking And Computers*. Springer, 2022, pp. 63–71.

[14] B. Wu, S. Zhang, W. Tian, and H. Wang, “Application and development prospect of monitoring screen based on three. js unit equipment control system,” in *2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. IEEE, 2022, pp. 347–351.

[15] G. Bradski, A. Kaehler *et al.*, “Opencv,” *Dr. Dobb’s journal of software tools*, vol. 3, no. 2, 2000.

[16] G. Bradski, “The opencv library.” *Dr. Dobb’s Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[17] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. ” O’Reilly Media, Inc.”, 2008.

[18] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, “A brief introduction to opencv,” in *2012 proceedings of the 35th international convention MIPRO*. IEEE, 2012, pp. 1725–1730.

[19] S. Wawage, V. Shinde, S. Indurkar, and G. Sorte, “Augmented-reality computer-vision and iot control platform,” 2019.

[20] F. H. Shubho, F. Iftekhar, E. Hossain, and S. Siddique, “Real-time traffic monitoring and traffic offense detection using yolov4 and opencv dnn,” in *TENCON 2021-2021 IEEE Region 10 Conference (TENCON)*. IEEE, 2021, pp. 46–51.

[21] D. Bandopadhyay, V. Jha, A. Bandyopadhyay, P. Roy, R. Halder, and S. Majhi, “Automated people monitoring system using opencv and raspberry pi,” in *ICT Analysis and Applications*. Springer, 2022, pp. 905–913.

[22] Y. Abdelrahman, M. El-Salamony, and M. Khalifa, “Visual data acquisition for measuring devices using deep learning,” in *2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. IEEE, 2021, pp. 301–304.

[23] J. Rivalan, “Palindrome.js,” May 2019. [Online]. Available: https://github.com/Smile-SA/palindrome.js

[24] ——, “Palindrome.js-grafana-plugin,” Apr. 2024. [Online]. Available: https://github.com/Smile-SA/palindrome.js-grafana-plugin

[25] W. Zhu, “Synergetic scheduling energy and reserve of wind farms for power systems with high-share wind power,” *Journal of Energy Engineering*, vol. 149, no. 2, p. 04023003, 2023.