2025 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C) | 979-8-3315-0215-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/ACSOS-C66519.2025.00047

Lightweight Temporal Consistency for Grid-Based Obstacle Detection in Edge Devices

Omer Kurkutlu and Arman Roohi
Department of Electrical and Computer Engineering, University of Illinois Chicago, Chicago, IL
E-mails: {okurku2,aroohi}@uic.edu

***Abstract*—We present a lightweight, robust obstacle detection framework designed for deployment on edge devices, including microcontrollers with sub-MB memory. Our approach transforms high-resolution image input into a compact 6×8 binary obstacle grid, referred to as an 'obstacle matrix'. This matrix enables efficient spatial reasoning without requiring bounding box regression or object classification. To address detection noise and instability across frames, we introduce a low-overhead temporal smoothing mechanism that significantly improves consistency without sacrificing real-time performance. The framework is benchmarked on diverse hardware, from GPU-enabled desktops to Raspberry Pi and the Arduino Nano 33 BLE, using three model variants: YOLOv8n, SSD-MobileNet v2, and FOMO. Quantitative results show a 13.38 percentage point increase in detection accuracy with smoothing, while matrix projection and smoothing combined remain under 0.1 ms.**

All source code, trained models, curated dataset, and demo video are available at: https://github.com/UIC-SIRIUS-Lab/Grid-Based-Obstacle-Detection-in-Edge-Devices.

I. Introduction

Obstacle detection is a fundamental capability in autonomous systems [1], which enables robots, drones, and other agents to perceive and interact safely with their environments. Traditionally, this task relies on computationally intensive algorithms that perform object recognition, segmentation, or depth estimation approaches that often assume access to powerful processors or cloud infrastructure [2], [3]. For example, models like YOLOv8 (up to ≈60 M parameters) [4] and SegFormer-B2 [5] can still require hundreds of megabytes of memory and significant compute resources. For instance, SegFormer-B2 requires approximately 47 GFLOPs per inference and consumes over 250 MB of memory during execution. While it can run at 15 to 20 ms per frame on high-end GPUs, inference on edge devices like the Raspberry Pi 4 is severely limited, taking over 1500 ms per frame, making them impractical for deployment on microcontrollers or low-power edge devices. In contrast, many real-world autonomous agents, such as aerial drones or microrobots, must operate within strict energy and memory constraints, often with less than 1 MB of RAM and no GPU acceleration. These limitations necessitate the development of new methods that drastically reduce computational load while maintaining reliable spatial awareness. Recent advances in processing-in-sensor architectures have shown promise for addressing these constraints [6], [7], but challenges remain in maintaining real-time performance while ensuring temporal consistency.

The growing demand for low-cost, low-power autonomous agents, such as microrobots, edge-deployed environmental sensors, and drone swarms, has further emphasized the need for perception systems that operate entirely on-device under tight resource budgets. In such settings, standard object detection pipelines are often infeasible, prompting the need for lightweight, real-time alternatives that still retain critical spatial awareness. Among various sensing modalities, image-based obstacle detection remains highly attractive due to its simplicity, affordability, and wide applicability [8]. Cameras offer dense visual information without the need for specialized hardware like LiDAR or sonar [9]. However, the challenge lies in extracting actionable spatial information from images efficiently, especially on edge devices with limited computing power and no GPU acceleration [10]. Another key challenge in real-world deployments is temporal inconsistency [11], [12]. Frame-to-frame variations due to motion blur, occlusion, lighting changes, or model uncertainty can result in unstable predictions, causing erratic behavior in downstream control systems [13]. This requires lightweight temporal smoothing to enhance prediction reliability without adding latency or exceeding resource limits.

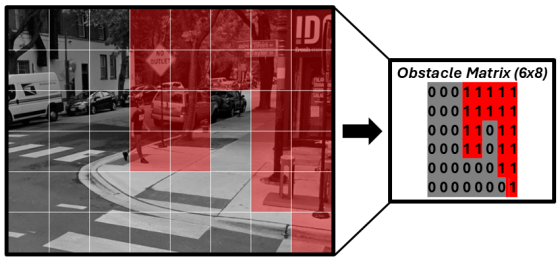

Fig. 1. Example of a lightweight obstacle detection output represented as a binary obstacle matrix. Each cell in the 6×8 grid indicates obstacle presence (1) or absence (0), providing a compact and interpretable spatial abstraction for real-time decision-making.

In this paper, we introduce a minimalist yet effective obstacle detection framework designed for edge intelligence. The system converts visual input into a fixed 6×8 binary obstacle matrix, Fig. 1, enabling fast spatial reasoning. To enhance robustness, we incorporate a memory-efficient temporal smoothing mechanism that improves accuracy in the presence of visual noise or rapid motion. We evaluated our approach across a diverse range of hardware platforms, microcontrollers, edge computers, and desktop-class GPU systems, using three distinct models: FOMO, SSD-MobileNet v2, and YOLOv8n. We thoroughly analyze latency, memory usage, and accuracy, demonstrating our method's adaptability and efficiency across varied environments.

979-8-3315-0215-7/25/$31.00 ©2025 IEEE
DOI 10.1109/ACSOS-C66519.2025.00047

II. Related Works

Lightweight and efficient obstacle detection has received increasing attention with the rise of edge AI and resource-constrained robotic applications. Traditional object detection methods, such as Faster R-CNN [14], SSD [15], and YOLO [16], have demonstrated strong performance on general-purpose GPUs and CPUs. However, their high computational demands make them unsuitable for deployment on microcontrollers and low-power edge devices without significant optimization. Recent advances in TinyML have enabled the deployment of neural networks on embedded systems with sub-MB memory footprints. For example, several methods have been proposed to reduce the computational complexity of object detection and segmentation models. Techniques such as network pruning [17], quantization aware training [18], and neural architecture search (NAS) [19] have shown effectiveness in lowering inference cost without significant accuracy degradation. Frameworks such as TensorFlow Lite for Microcontrollers (TFLM) [20] and Edge Impulse have introduced tools to quantify and compile models for low-power inference. Models like MobileNet [21], and its TFLite variants offer a trade-off between accuracy and efficiency for mobile and edge platforms. Additionally, FOMO (Fast Object Detection for Mobile) [22] eliminates the need for bounding-box regression, making it especially well suited for TinyML deployment. Our work leverages these developments by using FOMO, SSD-MobileNet v2 and YOLOv8n on different hardware profiles.

Temporal consistency in object detection is another area of active research. Techniques like optical flow, Kalman filtering [23], and attention-based temporal models [24] have been proposed to stabilize predictions across frames. However, these methods typically involve additional computation or memory overhead that is incompatible with microcontroller constraints. Lightweight filtering strategies, such as majority voting [25], [26] or temporal smoothing over occupancy grids, have shown promise in maintaining prediction stability under real-world noise. Our approach extends this idea by applying temporal smoothing directly to binary obstacle matrices to enhance robustness without incurring latency or memory penalties. Although several works have explored efficient perception for autonomous agents, few have addressed the combined challenge of real-time image-based obstacle detection and temporal consistency on microcontroller-class devices. This paper aims to bridge that gap by introducing a fully deployable, temporally robust system designed from the ground up for low-power edge intelligence.

III. Proposed Methodology

Our proposed system transforms high-resolution grayscale images into the compact 6×8 binary obstacle matrix. This abstraction enables fast, interpretable spatial reasoning on resource-constrained devices. The methodology comprises four main components: (1) task-specific dataset preparation and annotation tailored for obstacle detection, (2) a model inference pipeline that supports multiple architectures across various hardware tiers, (3) a lightweight matrix projection mechanism to convert detection results into a binary matrix, and (4) a temporal smoothing method designed to improve prediction stability without adding significant overhead. Each stage is optimized to minimize latency and memory usage while preserving the critical spatial context necessary for real-time autonomous decision-making.

A. Dataset Preparation and Annotation

What constitutes an obstacle can vary significantly depending on the application. An object might be considered a target in one context, an obstacle in another, or entirely irrelevant in a third. This distinction is particularly important in edge-based perception, where computational resources are limited and unnecessary detections should be avoided. Most standard data sets, such as COCO [27], Pascal VOC [28], and similar benchmarks, are designed for general-purpose object detection, image classification, or semantic segmentation. While these datasets have advanced general-purpose visual perception, they are not well-suited for practical obstacle detection, where spatial relevance and navigational risk are more critical than mere object presence. For example, a distant column may be correctly identified by object detection models but may not represent an obstacle due to its distance, as shown in Fig. 2. In natural environments like forests, detectors often enclose entire trees within a single bounding box, even when there is navigable space beneath the foliage, resulting in overly conservative or misleading obstacle representations. To improve precision, our dataset annotates relevant structural elements separately, such as trunks, low-hanging branches, and obstructive leaf clusters, focusing only on components that genuinely impede movement, as shown in Fig. 2 (b). While semantic segmentation could offer finer pixel-level localization, it introduces higher computational costs [10], making it less practical for resource-constrained edge platforms.

To address these limitations, this work introduces a task-specific approach to obstacle detection tailored for outdoor aerial and ground-based mobile platforms operating in cluttered environments. We curated a custom dataset comprising 1,120 labeled images with 11 obstacle-relevant classes: barrel, column (metal, concrete, wood), cone, leaf, light (streetlight and traffic light), box (mailbox, recycling bin, etc.), person, pole, sign (traffic or storefront signs), table, and trunk. These categories were selected to capture objects commonly encountered in real-world obstacle detection or navigation tasks. Fig. 2 (a) demonstrates how the same object can or cannot constitute an obstacle depending on its proximity and spatial context relative to the observer.

B. Model Architecture

To support deployment across a wide spectrum of edge computing platforms, we adopted three object detection models, YOLOv8n [29], FOMO [30], and SSD-MobileNet v2 [31], [32], each tailored to a distinct hardware class. *YOLOv8n* is the smallest variant of the YOLOv8 family, which is optimized for GPU-enabled desktop systems and offers a strong balance between accuracy and speed. It is suited for desktop-class applications and embedded GPUs where inference latency is

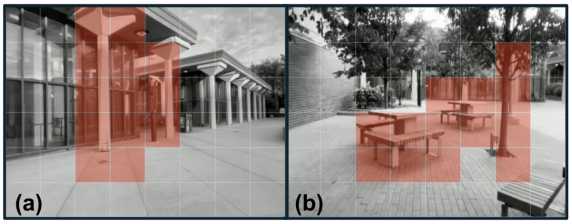

Fig. 2. (a) Obstacle detection with distance awareness: two columns close to the camera are detected as obstacles, while more distant columns are ignored. (b) Tables near the camera are identified as obstacles. Notably, only the trunk of the closest tree is detected, while leaves and distant trees are excluded, demonstrating the system's sensitivity to spatial relevance.

critical. *SSD-MobileNet v2* is converted to TensorFlow Lite (int8), serving as a middle-ground model for devices like the Raspberry Pi. It maintains reasonable detection performance with moderate computational requirements. Finally, *FOMO* is a fully quantized int8 model built on MobileNetV2-0.1, and is designed for ultra-low power microcontrollers. Unlike traditional detectors, it omits bounding-box regression in favor of dense objectness maps, aligning naturally with our grid-based perception. All models were trained on the same custom dataset of 1,120 grayscale images with 11 obstacle-relevant classes. During inference, class labels were discarded, and each model's output was converted into the binary obstacle matrix. This abstraction ensures compatibility across model architectures and input resolutions.

C. Grid-Based Obstacle Representation

To enable compact and interpretable spatial reasoning, we convert each model's output into a binary obstacle matrix of fixed size 6×8 (as shown in Fig. 3). For models like YOLOv8n and SSD-MobileNet, which produce bounding boxes, we mark a cell as 1 if any predicted bounding box overlaps with that region of the frame, and 0 otherwise. For FOMO, which directly predicts object presence on a dense output grid without bounding boxes, we aggregate its class-specific activation map into the corresponding obstacle matrix by thresholding detections within each cell. This unified representation reduces complex visual outputs to a structured occupancy map, regardless of the underlying detection mechanism or object class. This grid-based abstraction builds upon recent work in approximate computing for edge devices [33], [34], which has shown that strategic approximations can significantly reduce computational requirements without sacrificing critical functionality.

Fig. 3. (a) Bounding boxes generated by the object detection model. (b) Corresponding 6×8 obstacle matrix, where red-highlighted cells indicate occupied regions.

The dimension of the obstacle matrix is adjustable to accommodate different input resolutions and aspect ratios. For example, FOMO operates on 96×96 grayscale images, while other models use higher resolutions such as 320×320 or 640×480 with aspect ratios like 4:3 or 16:9. Choosing the matrix resolution involves a trade-off: smaller grids reduce computation but yield coarser localization, while larger grids improve spatial precision at the cost of increased processing overhead.

To determine the optimal configuration, we empirically evaluated several grid sizes. A smaller grid, such as 3×4, resulted in overly coarse localization and limited directional awareness, making it difficult to distinguish obstacle positions. In contrast, a larger grid like 9×12 offered finer spatial granularity, but the added complexity did not lead to meaningful improvements in navigation decisions. Semantic cues such as "obstacle on the left" or "in front" were captured equally well with smaller grids, while the larger matrix introduced additional memory usage and processing time.

We also tested a 6×6 grid, which aligns naturally with FOMO's 96×96 input resolution. However, it reduced horizontal resolution, which is suboptimal for many robotic navigation tasks where lateral awareness (e.g., turning left or right) is more important than vertical detail.

Based on these observations, we selected a 6×8 matrix as the best trade-off. It provides sufficient spatial resolution for reactive navigation while maintaining low computational cost. Additionally, the obstacle matrix is fully configurable, allowing easy reconfiguration across different models and application requirements.

D. Temporal Smoothing Mechanism

Single-frame obstacle detections can be inconsistent due to motion blur, lighting variation, or vibration, especially in fast-moving or unstable environments. For example, an object may be correctly detected in 9 out of 11 frames but missed in the remaining 2 due to transient artifacts, as summarized in Table I. These inconsistencies can lead to unstable or incorrect decisions when using raw detection outputs.

To address this, we implement a very lightweight temporal smoothing mechanism that aggregates binary obstacle grids

TABLE I
TEMPORAL SMOOTHING WITH DYNAMIC THRESHOLD
$T = \lfloor \text{BUFFER\_LENGTH}/2 \rfloor + 1$, BUFFER SIZE = 5.

Frame #	Raw Detection	Buffer Contents (Last N)	Smoothed Output
1	1	[1]	**1** (T=1)
2	1	[1, 1]	**1** (T=2)
3	0 (noise)	[1, 1, 0]	**1** (T=2)
4	1	[1, 1, 0, 1]	**1** (T=3)
5	0 (noise)	[1, 1, 0, 1, 0]	**1** (T=3)
6	1	[1, 0, 1, 0, 1]	**1** (T=3)
7	1	[0, 1, 0, 1, 1]	**1** (T=3)
8	1	[1, 0, 1, 1, 1]	**1** (T=3)
9	1	[0, 1, 1, 1, 1]	**1** (T=3)
10	1	[1, 1, 1, 1, 1]	**1** (T=3)
11	1	[1, 1, 1, 1, 1]	**1** (T=3)
12	0 (missed)	[1, 1, 1, 1, 0]	**1** (T=3)
13	0 (missed)	[1, 1, 1, 0, 0]	**1** (T=3)
14	0 (missed)	[1, 1, 0, 0, 0]	0 (only 2/5)
15	0 (missed)	[1, 0, 0, 0, 0]	0 (only 1/5)

from the most recent N frames (N=5 by default). For each obstacle matrix cell, we count how many times it was marked as occupied within the current buffer. A cell is labeled as an obstacle in the smoothed output if it appears to be occupied at least the threshold $T = \lfloor \frac{N}{2} \rfloor + 1$ of those frames. This majority vote strategy reduces the influence of sporadic false negatives and enhances detection stability. While it introduces a slight delay, bounded by the buffer size and frame rate, the delay is negligible for most edge applications and is adjustable depending on system requirements. Table I illustrates this process over 15 frames. Early inconsistencies in obstacle detection are smoothed out once the buffer accumulates enough supporting evidence, enabling more robust and reliable obstacle awareness under noisy conditions.

IV. Validation and Evaluation

A. Experimental Setup

We evaluated each model on its target hardware to assess runtime, compatibility, and deployability of our obstacle detection framework.

YOLOv8n (Desktop-Class Baseline): YOLOv8n was deployed on a GPU-enabled desktop system and evaluated using 640×480 grayscale input images. The model demonstrated an average end-to-end latency of 44.2 ms per frame, encompassing preprocessing, inference, matrix projection, and temporal smoothing. This configuration establishes a high-performance baseline suitable for latency-sensitive robotics applications requiring rapid visual perception and control. Here, mean Average Precision (mAP) summarizes the model's precision–recall performance across all classes at a given intersection-over-union (IoU) threshold. In terms of accuracy, YOLOv8n achieved an F1 score of 0.51 at a confidence threshold of 0.33 and an overall accuracy of 41.86%, with a mAP of 52%, demonstrating strong detection capability for edge-aligned spatial reasoning.

SSD-MobileNet v2 (Raspberry Pi Tier): To evaluate mid-tier edge deployment, a quantized SSD-MobileNet v2 model (TFLite int8) was tested on Raspberry Pi 2, 3, and 4 with 320×320 grayscale inputs. The model achieved average end-to-end latencies of 397.19 ms, 340.11 ms, and 152.69 ms, respectively, including all processing stages. With a compact model size of 4.7 MB, it provides a strong balance between detection quality and computational efficiency, making it well-suited for moderate-resource platforms without GPU support. Training convergence was indicated by a total loss of 0.5044, comprising a classification loss of 0.1738 and a localization loss of 0.0950. The model achieved an F1 score of 0.758 and a mAP of 47.5%, validating its detection performance for edge-class devices.

FOMO (Microcontroller Tier): FOMO, a fully quantized int8 model designed for TinyML applications, was deployed on the Arduino Nano 33 BLE using 96×96 grayscale inputs. With a model size of just 70.2 KB (flash) and 123.9 KB (RAM), it achieved a total latency of 795 ms per frame, including preprocessing, inference, and matrix projection. Despite its minimal footprint, FOMO attained an F1 score of 0.27 and a mAP of 23.2% on the validation set, demonstrating effective obstacle detection under severe memory and compute constraints, suitable for intermittently powered and ultra-low power platforms.

Cross-Platform Abstraction: Across all devices, model output was consistently projected onto a fixed 6×8 binary obstacle matrix, allowing seamless integration with downstream control pipelines. This abstraction ensures uniform behavior regardless of resolution, input modality, or compute budget, validating the portability and robustness of our system across diverse platforms.

B. Grid-Based Abstraction Performance

Our system encodes obstacle presence in a fixed 6×8 binary matrix, reducing each frame to only 48 bits. This compact format minimizes memory and bandwidth usage, enabling fast, low-overhead processing that is ideal for microcontrollers and real-time control systems. Despite its simplicity, the grid retains sufficient spatial context to support tasks such as obstacle avoidance, trajectory planning, and motion estimation, without relying on object classification or precise bounding boxes. Its fixed size and binary format eliminate the need for dynamic memory allocation and simplify data handling, which is particularly beneficial for resource-limited platforms. As shown in Table II, the matrix projection step takes less than 0.1 ms on all devices, including the Arduino Nano 33 BLE. This negligible overhead enables seamless integration with any inference model, regardless of hardware limitations. Its lightweight design enables efficient transmission, storage, and autonomous control, providing a scalable, reliable link between perception and decision-making.

C. Temporal Smoothing vs Grid

Other temporal smoothing techniques such as optical flow, Kalman filtering, and attention-based temporal models have been widely used to enhance consistency in video object detection. However, these methods typically operate on pixel-level features, bounding box coordinates, or deep feature maps. In contrast, our method applies smoothing directly to the binary obstacle matrix, which represents high-level spatial occupancy.

TABLE II
Cross-Device Latency and Resource Benchmarking of Obstacle Detection Pipeline.

Device	Model	Input Size	Model Size	Preproc. (ms)	Infer. (ms)	Grid (ms)	Smooth (ms)	Total (ms)	mAP
Arduino Nano 33 BLE	FOMO	96×96 (Gray)	∼70.2 KB (flash)	80.00	713.00	∼1	∼1	795.00	23.2%
Raspberry Pi 2	SSD-MobileNet v2	320×320 (Gray)	∼4.7 MB	2.64	393.45	∼1	∼0.1	397.19	47.5%
Raspberry Pi 3	SSD-MobileNet v2	320×320 (Gray)	∼4.7 MB	1.20	337.81	∼1	∼0.1	340.11	47.5%
Raspberry Pi 4	SSD-MobileNet v2	320×320 (Gray)	∼4.7 MB	1.36	150.23	∼1	∼0.1	152.69	47.5%
Desktop-class CPU/GPU	YOLOv8n	640×480 (Gray)	∼6 MB	0.75	43.33	0.11	∼0.01	44.20	52%

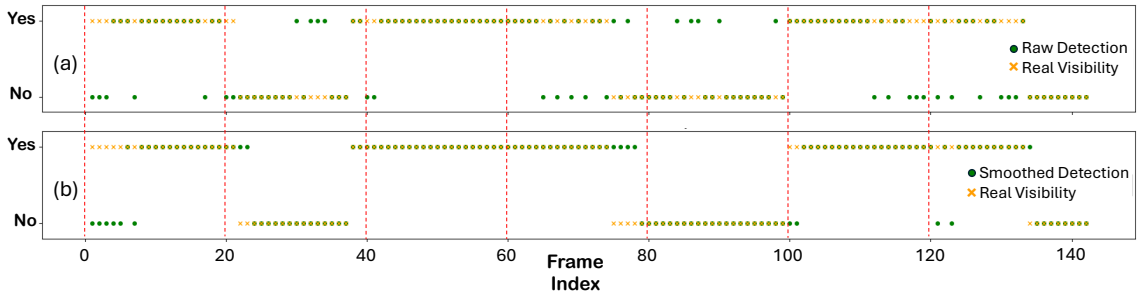

Fig. 4. (a) Raw and (b) smoothed detections versus real visibility. Frame-by-frame comparison between model predictions and manually annotated ground truth across 142 consecutive frames. Top: Raw detection output exhibits inconsistencies due to noise, occlusion, and motion blur, particularly noticeable between frames 22–37 (camera shake) and 75–99 (distant object). Bottom: Temporal smoothing significantly reduces these fluctuations, producing stable and accurate obstacle visibility that closely aligns with the ground truth.

Due to this abstraction layer, our approach is structurally different and not directly comparable to those methods. To evaluate the effectiveness of our temporal smoothing method, we compared raw detections with manually labeled ground truth ("real visibility") over 142 consecutive frames. As shown in Fig. 4(a), the raw detection results exhibit noticeable inconsistencies, particularly during rapid transitions and under noisy conditions, caused by factors such as momentary occlusion, motion blur, or low detection confidence. In contrast, the smoothed detection output as depicted in Fig. 4(b) aligns closely with the ground truth by reducing false negatives and stabilizing predictions over time. This improvement is especially evident in two challenging segments: between frames 22–37, camera shake reduced object clarity and led to intermittent misses in the raw output; and between frames 75–99, the object appeared too distant for consistent detection, resulting in occasional false positives. In both cases, temporal smoothing effectively filtered out these fluctuations, maintaining a more accurate and reliable obstacle awareness.

Importantly, this smoothing process is computationally lightweight, operating on a compact 48-bit obstacle matrix per frame. This enables efficient temporal filtering even on resource-constrained systems without introducing significant latency or memory overhead. Quantitatively, the raw detection approach achieved an accuracy of 74.65%, while temporal smoothing improved it to 88.03%, a gain of 13.38 percentage points. These results demonstrate that our lightweight temporal filtering significantly improves detection stability and reliability in real-world scenarios.

V. Discussion

To evaluate the effectiveness of our temporal smoothing approach, we captured a video sequence and analyzed it frame by frame. Each frame contained both the raw and smoothed detection outputs, allowing direct visual comparison. Because this process requires substantial computational resources, we conducted the evaluation on a GPU-enabled desktop system using the YOLOv8n model; the results illustrated in Fig. 4 are based on this setup. However, performing the same frame-by-frame analysis was not feasible on the Arduino Nano 33 BLE due to its limited processing capabilities and inability to record video. Instead, we verified the Arduino detection accuracy by capturing individual frames and visually comparing the predicted obstacle matrix with the real-world scene at the moment of capture. Since the same matrix projection and smoothing logic are used consistently across all platforms, and the method has been validated under controlled conditions on GPU hardware, we are confident that the system performs reliably on microcontroller-class devices like the Arduino. Furthermore, the lightweight nature of the matrix projection and smoothing mechanism ensures generalizability to other embedded platforms, including the Raspberry Pi.

VI. Conclusion and Future Work

In this work, we presented a lightweight, platform-agnostic obstacle detection framework that transforms high-resolution visual input into a compact 6×8 binary obstacle matrix, enabling efficient spatial reasoning on resource-constrained edge devices. We demonstrated the adaptability of our pipeline across a broad spectrum of hardware, from GPU-enabled desktop systems to Raspberry Pi edge devices and ultra-low power microcontrollers like the Arduino Nano 33 BLE, by deploying three distinct model architectures: YOLOv8n, SSD-MobileNet v2 (TFLite int8), and FOMO (quantized int8). Despite substantial differences in memory, compute, and power availability across these platforms, our approach maintained consistent behavior and latency characteristics, validating its robustness and portability.

We also introduced a lightweight temporal smoothing mechanism that significantly improves prediction stability without incurring additional computational overhead. This method improves detection accuracy under challenging conditions such as rapid motion, occlusion, motion blur, and lighting variations. Quantitative evaluation showed a 13.38 percentage point increase in accuracy when smoothing was applied, making the system more resilient to transient detection noise. The perception pipeline is efficient and suitable for real-time control on microcontroller-class devices with limited memory, owing to a matrix projection time of less than 0.1 ms.

A promising direction for future work is to integrate this grid-based obstacle detection framework with lightweight path planning algorithms directly on edge devices. Although the

binary obstacle matrix does not encode object identities or depth, it provides sufficient spatial context for real-time navigation. By capturing obstacle presence in the grid, it enables directional decision-making (e.g., turn left, move forward) based on occupancy patterns. This abstraction is compatible with efficient control strategies such as rule-based logic or reinforcement learning, which can operate on grid-level inputs without high-resolution perception. Such integration would enable fully autonomous agents like microrobots, swarm drones, or remote monitors to perceive and react to dynamic environments entirely on-device, while operating under stringent energy and memory constraints.

To support such fully autonomous and efficient edge deployments, power consumption must also be carefully evaluated. While this work primarily focuses on latency and memory usage, we acknowledge that power consumption is a critical consideration for resource-constrained devices. The proposed framework is designed with low-power deployment in mind using binary representations, quantized models, and fixed-size processing to minimize energy demands. However, a detailed analysis of energy usage was outside the scope of this study. In future work, we plan to incorporate on-device power profiling using Nordic Power Profiler or Monsoon to quantify energy efficiency across platforms such as the Arduino Nano 33 BLE and Raspberry Pi.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under Grant Nos. 2504839 and 2448133.

REFERENCES

[1] B. Xu and Y. Chen, "A survey of obstacle detection using vision-based sensor for autonomous vehicles," *IEEE Access*, vol. 8, pp. 60 132–60 144, 2020.
[2] L.-C. Chen *et al.*, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *ECCV*, 2018, pp. 801–818.
[3] A. Kendall *et al.*, "End-to-end learning of geometry and context for deep stereo regression," in *ICCV*, 2017, pp. 66–75.
[4] G. Jocher, A. Chaurasia, I. Sabir, J. Borovec, L. Q, A. Stoken *et al.*, "Yolov8 - ultralytics next-generation object detection models," https://github.com/ultralytics/ultralytics, 2023, accessed: August 6, 2025.
[5] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.
[6] S. Angizi, S. Tabrizchi, D. Z. Pan, and A. Roohi, "Pisa: A non-volatile processing-in-sensor accelerator for imaging systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 4, pp. 962–972, 2023.
[7] A. Roohi, S. Tabrizchi, M. Morsali, D. Z. Pan, and S. Angizi, "Pipsim: A behavior-level modeling tool for cnn processing-in-pixel accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 1, pp. 141–150, 2023.
[8] T. Kim *et al.*, "Vision-based obstacle detection and avoidance systems for autonomous ground vehicles: A review," *Journal of Field Robotics*, vol. 37, no. 1, pp. 124–144, 2020.
[9] Z. Zhou *et al.*, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2021.
[10] H. Hu *et al.*, "A survey on visual perception for autonomous driving: Datasets, tasks, and challenges," *CVIU*, vol. 215, p. 103341, 2022.
[11] J. Song *et al.*, "Temporal knowledge distillation for efficient video object detection," in *CVPR*, 2022, pp. 9543–9552.
[12] T. He *et al.*, "Bag of tricks for image classification with convolutional neural networks," in *CVPR*, 2019, pp. 558–567.
[13] Y. Zou *et al.*, "Robust perception for autonomous driving: Datasets, challenges, and beyond," *arXiv preprint arXiv:2004.08566*, 2020.
[14] S. Ren *et al.*, "Faster r-cnn: Towards real-time object detection with region proposal networks," *NeurIPS*, 2015.
[15] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *ECCV*, 2016.
[16] J. Redmon *et al.*, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.
[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.
[18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
[19] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
[20] R. David and et al., "Tensorflow lite micro: Embedded machine learning on tinyml systems," 2020.
[21] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," in *arXiv preprint arXiv:1704.04861*, 2017.
[22] "Edge impulse fomo," 2022, https://docs.edgeimpulse.com/docs/edge-impulse-studio/processing-blocks/image/fomo.
[23] A. Bewley *et al.*, "Simple online and realtime tracking," in *ICIP*, 2016.
[24] P. Zhao, Y. Wang, J. Liu, X. Zhang, P. Li, and Y. Song, "Attention-based temporal–spatial convolutional network for ultra-short-term load forecasting," *Applied Energy*, vol. 307, p. 118231, 2022.
[25] M. Dallel, V. Havard, Y. Dupuis, and D. Baudry, "A sliding window based approach with majority voting for online human action recognition using spatial temporal graph convolutional neural networks," in *Proceedings of the 2022 7th International Conference on Machine Learning Technologies*, ser. ICMLT '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 155–163. [Online]. Available: https://doi.org/10.1145/3529399.3529425
[26] A. Zehtabian and H. Ghassemian, "An adaptive framework for spectral-spatial classification based on a combination of pixel-based and object-based scenarios," *Earth Science Informatics*, vol. 10, pp. 1–12, 09 2017.
[27] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," *ECCV*, 2014.
[28] M. Everingham *et al.*, "The pascal visual object classes (voc) challenge," *IJCV*, vol. 88, no. 2, pp. 303–338, 2010.
[29] G. Jocher *et al.*, "Ultralytics yolov8," *Zenodo*, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
[30] Q. Zhu *et al.*, "Fomo: Lightweight object detection for tinyml," *arXiv preprint arXiv:2210.02247*, 2022.
[31] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *CVPR*, 2017, pp. 7310–7311.
[32] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems." Mountain View, CA: Tensorflow, 2015.
[33] S. Tabrizchi, R. Gaire, M. Morsali, M. Liehr, N. Cady, S. Angizi, and A. Roohi, "Apris: Approximate processing reram in-sensor architecture enabling artificial-intelligence-powered edge," *IEEE Transactions on Emerging Topics in Computing*, 2024.
[34] S. Tabrizchi, A. Nezhadi, S. Angizi, and A. Roohi, "Appcip: Energy-efficient approximate convolution-in-pixel scheme for neural network acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, no. 1, pp. 225–236, 2023.