Research on Object Recognition Approaches for Mobile Platforms with Limited Resources[1]

Dmytro Dovhal[1,†], Natalia Myronova[1,*,†] and Anzhelika Parkhomenko[1,2,†]

[1] National University Zaporizhzhia Polytechnic, 64, Zhukovskogo str., Zaporizhzhia, 69063, Ukraine
[2] University of Applied Sciences and Arts, 23, Otto-Hahn str., Dortmund, 44227, Germany

Abstract

This paper presents the results of a study of approaches to object detection for mobile platforms with limited computing resources. The research focuses on the development of software for real-time object identification using computer vision technologies optimised for embedded systems. The selected hardware platform is the ESP32-CAM, a low-power microcontroller with a built-in camera that allows for efficient video stream processing. The proposed approach involves the use of lightweight image processing methods and deep neural networks, in particular YOLO, adapted to work in resource-limited environments. Experiments confirm that the system can be implemented for real-world applications such as automated monitoring, security, and autonomous navigation.

Keywords

object detection, ESP32-CAM, computer vision, machine learning, OpenCV, Python

1. Introduction

Nowadays, the development of software for object detection systems for mobile platforms with limited resources is driven by the growing demand for autonomous robotic systems in various industries, such as security [1, 2, 3, 4], logistics [5, 6], agriculture [7, 8, 9, 10], etc. In addition, such systems have a dual purpose, being used both in civilian and military applications [11].

In civilian purposes, they can be used for emergency response, agricultural automation and intelligent surveillance. From a military perspective, autonomous platforms with limited resources play an important role in reconnaissance, target detection and situational awareness on the battlefield, where real-time data processing is critical [12]. Such mobile platforms equipped with cameras and sensors are increasingly used to perform complex tasks such as obstacle detection, mapping and environmental analysis [13, 14]. As these platforms often have limited resources, especially in terms of computing power and energy consumption, it is important to develop software solutions that allow for a sufficient level of object identification and recognition at a minimal cost.

The use of computer vision algorithms together with optimised software solutions ensures fast and accurate data processing in real time [15]. This approach enables mobile platforms with limited hardware resources to perform complex tasks such as object detection, obstacle recognition, and navigation [16]. Software optimization for low-power devices allows for efficient image processing, reducing CPU load and minimising power consumption.

This enables autonomous systems to operate in challenging conditions, increasing the accuracy and speed of performing various tasks. Therefore, the development of object detection software is a pressing task.

1SMARTINDUSTRY-2025: International Conference on Smart Automation & Robotics for Future Industry, April 3 - 5, 2025, Lviv, Ukraine.
* Corresponding author.
† These authors contributed equally.
dovgal.dima.86@gmail.com(D. Dovhal); natali.myronova@gmail.com (N.Myronova); parkhomenko.anzhelika@gmail.com (A. Parkhomenko);
0009-0002-6712-5468 (D. Dovhal); 0000-0001-7484-559X (N.Myronova); 0000-0002-6008-1610 (A. Parkhomenko);

The goal of the work is to research and implement object recognition algorithms for mobile platforms with limited resources, allowing for real-time video stream processing with minimal computing resources and a high degree of reliability.

2. Materials and methods

2.1. Software architecture for mobile platforms

According to the conducted research, the software architecture for mobile platforms (e.g., wheeled robots, tracked or other autonomous vehicles) is usually built on a modular principle and consists of the following main components [17]:

- Hardware level that provides the interface with sensors, actuators and other physical components of the platform (controllers, power modules, servo motors, etc.).
- The system level or middleware is responsible for the integration of hardware and software, providing standardized interfaces and data transfer (real-time operating system FreeRTOS, Zephyr, RTEMS, robotics frameworks ROS/ROS2 [18, 19]).
- Control level responsible for decision-making and platform control (PID controllers for movement, programming of the motion trajectory and obstacle avoidance).
- Localisation and navigation level, which provides platform location and trajectory planning (localisation algorithms, Simultaneous Localization and Mapping (SLAM) and navigation, such as algorithms A Star [20] and Dijkstra for route planning).
- Sensory data processing level - image processing using computer vision software solutions, data filtering using the Kalman filter [21], object recognition.
- Interaction level - development of interfaces for monitoring and managing the platform (web or mobile applications, use of remote access interfaces, data exchange via MQTT protocols) [22]).

The generalised software architecture for mobile platforms is shown in Figure 1. This architecture can be adapted to a specific mobile platform (ground drone, quadcopter, etc.).

The main software components for mobile platforms (in particular, ground drones), necessary for the implementation of autonomous navigation, detection and identification of objects, as well as real-time data processing, are the following [17]:

- The motion control system is responsible for controlling the platform's motor functions, including speed, direction and manoeuvring control, namely: trajectory planning is the calculation of the optimal route based on set goals or received data; motion control is the control of manoeuvres using feedback from sensor.
- The image processing module is used to collect and analyse visual information from cameras installed on the platform in order to identify objects and detect possible obstacles.
- Sensor integration module, which includes processing of data from various sensors such as GPS, LiDAR, ultrasonic and inertial modules, to monitor and map the environment.
- User interaction module that allows operators to remotely control the platform or configure autonomous modes and is implemented through web interfaces, mobile applications or specialised consoles.

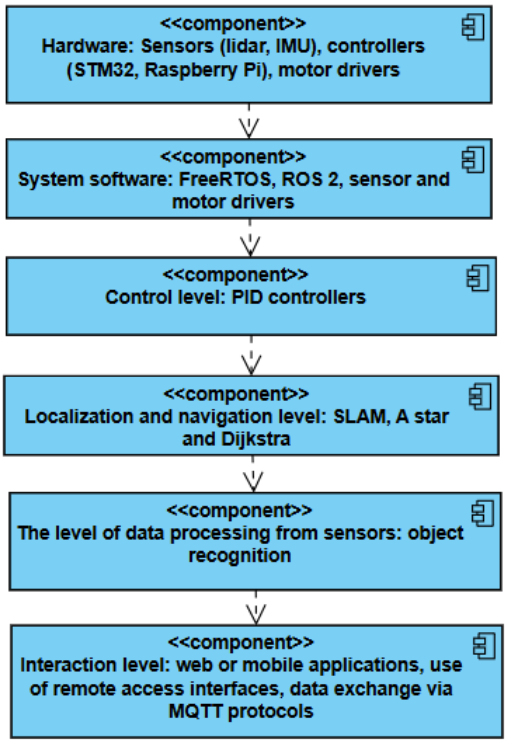

Figure 1: Generalised software architecture for mobile platforms

2.2. Research on hardware aspects for mobile platforms with limited resources

Mobile platforms with limited resources are the basis for building compact and affordable computer vision systems. The comparative analysis of the most common platforms: ESP32-Cam, Raspberry Pi 4, Arduino Nano RP2040, and NVIDIA Jetson Nano [23, 24, 25, 26] with limited resources is shown in Table 1.

Table 1
The comparative analysis of popular platforms with limited resources

Platform	CPU	Memory	Connection	Availability of camera	Price
ESP32-Cam	Tensilica LX6	520 KB RAM	Wi-Fi, Bluetooth	OV2640 (1600×1200)	$10-15
Raspberry Pi 4	ARM Cortex-A72	2/4/8 GB LPDDR4	Wi-Fi, Bluetooth, GPIO	Raspberry Pi Camera (up to 12 Mp)	$35-70
Arduino Nano RP2040	Dual Cortex-M0+	264 GB SRAM	USB, GPIO	Via additional modules	$10-20
NVIDIA Jetson Nano	Cortex-A57, 128 CUDA	4 GB LPDDR4	Ethernet, GPIO	MIPI CSI-2	$99

The analysis showed that additional modules are needed to connect the camera to the Arduino Nano RP2040 platform, so to simplify the task it is advisable to consider the ESP32-Cam, Raspberry

Pi 4 and NVIDIA Jetson Nano platforms for further research. The comparison of the characteristics of the selected platforms [23, 24, 25, 26] with limited resources for implementing computer vision is given in Table 2. According to the criterion "TensorFlow, PyTorch, OpenCV support" from Table 2, the following conclusions can be made:

- ESP32-Cam has limited support as TensorFlow Lite Micro only works with simplified models while OpenCV is used for basic frame processing. This limitation is due to low computing power and limited memory.
- Raspberry Pi 4 fully supports TensorFlow, PyTorch and OpenCV, enabling the implementation of complex computer vision algorithms, including both training and inference.
- NVIDIA Jetson Nano is the optimal platform for TensorFlow, PyTorch, and OpenCV with its high pcomputing power and GPU hardware support for accelerating neural networks.

Table 2
The comparative characteristics of selected platforms with limited resources for computer vision implementation

Platform	Video streaming support	Energy consumption	TensorFlow, PyTorch, OpenCV support	Application areas
ESP32-Cam	160×120 - 1600×1200, up to 30 FPS	0.6-0.9 W	TensorFlow Lite Micro, OpenCV (limited)	IoT systems, surveillance, portable monitoring systems
Raspberry Pi 4	640×480 - 4K, up to 60 FPS	5-8 W	TensorFlow, PyTorch, OpenCV	Robotics, home automation, multimedia systems
NVIDIA Jetson Nano	1080p - 4K, up to 30 FPS	10-15 W	TensorFlow, PyTorch, OpenCV	Computer vision, deep learning, complex robotics systems

As is known, the choice of platform depends on the project budget and requirements for support machine learning algorithms.

In terms of price, the ESP32-Cam is the cheapest platform, making it an attractive choice for low-budget projects. The Raspberry Pi 4 is in the middle price range, offering flexibility and the ability to implement more complex projects. The NVIDIA Jetson Nano is at the higher end of the price range, but its high power justifies the cost for compute-intensive projects.

Thus, ESP32-Cam is suitable for low-cost projects that require minimal image processing and remote control. Raspberry Pi is suitable for more complex tasks that require high computing power and the ability to work with large cameras. Jetson Nano is suitable for machine learning projects that require high performance and real-time video processing.

Therefore, the ESP32-Cam platform was chosen for the study, since it was necessary to implement an object detection system with limited use of hardware resources. The ESP32-Cam provides the required functionality through a compact design, energy efficiency and sufficient computing power to perform basic computer vision tasks. In addition, its low cost and availability make it an optimal choice for developing systems on a budget. The platform supports video stream processing using the built-in OV2640 camera module and data transmission via Wi-Fi. This allows integration with other systems, transmittion of video stream to the server for further processing and implementation tasks that requiring minimal energy consumption.

2.3. Research of the features of integration of machine learning technologies for mobile platforms with limited resources

At the research stage, the following technologies and frameworks were considered to solve the project tasks: ESP-WHO (ESP-IDF), TensorFlow Lite, OpenCV.

ESP-WHO (ESP-IDF) is a specialised computer vision framework developed by Espressif for use on the ESP32 [27]. It is part of the ESP-IDF (Espressif IoT Development Framework) ecosystem and provides integration with camera modules such as the OV2640 for basic image processing tasks.

TensorFlow Lite is a simplified version of the TensorFlow framework optimized to run on devices with limited resources such as mobile devices [28]. For microcontrollers, including ESP32, the TensorFlow Lite for Microcontrollers version is used.

OpenCV (Open Source Computer Vision Library) is a popular computer vision library that supports a wide range of image and video operations. It is widely used for processing video streams, object recognition and working with machine learning models.

The results of the comparison of machine learning technologies and frameworks for mobile platforms with limited resources are presented in Table 3.

Table 3
The comparison of machine learning technologies and frameworks for mobile platforms with limited resources

Technology (framework)	ESP-WHO (ESP-IDF)	TensorFlow Lite	OpenCV
Advantages	ESP32-specific, face recognition support	Optimised for low power consumption devices, using compact models	The universal framework for computer vision, the rich set of functions for image processing
Disadvantages	Limited set of functions for complex tasks; difficult to integrate third-party machine learning models	ESP32's limited memory does not allow loading even minimal TensorFlow models, as well as poor performance in video processing	More powerful hardware is required for real-time video processing, which requires an external server
Support ESP32-CAM	Limited support for complex tasks	Integration problems on ESP32-CAM	Support for basic tasks
Purpose and application	face recognition, basic computer vision tasks	Support for machine learning models on microcontrollers	Image and video processing, real-time object detection

3. Related works

The use of image processing approaches for object detection system for mobile platforms is an important component, especially considering the limited resources of mobile platforms. The main tasks include data preparation, pre-processing to improve image quality, and the use of object detection algorithms.

Data preparation for the machine learning model consists of frame size preparation and image normalisation [29]. For most machine learning models to function properly, input images must be of a fixed size. This is achieved by scaling the image, which allows to adjust the frames to the required dimensions without losing importand information. Image normalization involves bringing

pixel values into the range [0, 1]. This helps prevent large discrepancies between pixel values and ensures stable performance of the machine learning model.

Pre-processing to improve image quality includes the use of noise filtering and lighting correction.

Noise filtering reduces noise that can reduce recognition accuracy. The following filters can be distinguished [30]:

- The Gaussian filter is used to smooth images and reduce fine noise, workss based on a Gaussian distribution, which determines the values of pixel depending on their neighbors.
- Median filter is used to remove image noise while preserving sharp edges, and is especially effective for images with point noise.

Lighting correction involves adjusting the brightness and contrast of an image and is performed using histogram equalization, a technique that increases contrast by evenly distributing pixel intensities, thereby improving image quality.

According to the conducted research, the most common algorithms for detecting objects in images that can be implemented for mobile platforms are: Haar cascades [31, 32], Histogram of Oriented Gradients (HOG), You Only Look Once (YOLO) [33], Single Shot Multibox Detector (SSD) та Region-based Convolutional Neural Networks (Faster R-CNN).

Haar cascades are a classical object recognition method [31, 32], that uses Haar-like features to detect objects such as faces, cars, etc. The algorithm uses a cascade of simple classifiers that sequentially filter the frame, quickly discarding unnecessary parts of the image.

HOG is a method that detects objects by analysing histograms of oriented gradients in local areas of an image. It is particularly effective at detecting people in the frame by recognizing characteristic shape patterns.

YOLO is one of the most popular object detection methods [33]. Unlike traditional methods such as Haar cascades, YOLO analyses the image as a whole, dividing it into a grid and detecting several objects simultaneously. This approach allows to quickly processing large images and work in real time.

SSD is another fast algorithm for real-time object detection. It is similar to YOLO, but uses different sizes of “anchors” to detect objects. SSD is faster than most methods on medium and low resolution images.

Faster R-CNN is a more complicated model for object detection that uses regional proposals to find potential objects. Compared to other methods, it has high accuracy, but requires significantly more resources.

The results of the comparison of object detection algorithms in images are presented in Table 4.

To implement the object detection system on the ESP32-CAM platform, it was decided to choose the YOLO algorithm due to its real-time processing speed. One of the main advantages of YOLO is its ability to perform object detection in a single pass through the network, providing high speed and real-time capabilities. This is extremely important for the ESP32-Cam, as the platform has limited computing resources but requires the ability to process video streams with minimal delays. Unlike other models such as Faster R-CNN that rely on pre-generated region proposals to detect object, YOLO processes the image simultaneously without performing complex pre-processing steps, speeding up the detection process, which is essential for running on the ESP32-Cam. Despite hardware limitations (520 KB of RAM, 4 MB of flash memory), ESP32-Cam can use optimized versions of YOLO models such as Tiny YOLO or YOLOv4-tiny, which have a reduced number of parameters and require less memory and processing power. One of the reasons for choosing YOLO for the ESP32-Cam is the ability to use TensorFlow Lite Micro, which supports simplified models including YOLO. This allows inference to be performed on microcontrollers with limited resources. YOLO is also capable of detecting multiple objects simultaneously in a single frame, which is important for many real-world applications such as monitoring multiple objects at

the same time. This allows ESP32-Cam to be used for tasks where it is necessary to simultaneously identify different types of objects in a video stream.

YOLO offers a wide range of tools and libraries for implementation on various platforms, including support for libraries such as OpenCV, allowing for easy integration with platforms such as the ESP32-Cam.

Table 4
The comparison of object detection algorithms in images

Algorithm	Advantages	Disadvantages	ESP32-CAM support	Purpose
Haar cascades	Fast, with low resource requirements	Low accuracy for complex objects	Limited	Simple detection of objects, for example, a person's face
HOG	High accuracy for people detection	Requires more resources	Limited	People detection
YOLO	High speed and accuracy	Reduced accuracy for small objects	Supported	Real-time object detection
SSD	Processing speed on medium-sized images	less accurate than YOLO on large images	Supported	Detection of several objects at the same time
Faster R-CNN	Very accurate	Requires significant resources	Limited	Detection of complex scenes

4. Development of an algorithm based on the YOLO computer vision model for ESP32-Cam

Detection and recognition of objects in a real-time video stream using the YOLO computer vision model was implemented with the ESP32-Cam module, with subsequent display of results on the screen. The algorithm detects objects in the image, classifies them into categories and displays the results as rectangles around the detected objects with corresponding labels. Below is a description of the algorithm.

Step 1. Importing and connecting libraries to a video stream.

```
import cv2
import numpy as np
```

The OpenCV library cv2 is used to work with images and videos. The numpy library is required to work with numeric arrays, which is necessary for processing data by a neural network.

```
url = "http://192.168.0.102:81/stream"
```

Next, the URL for connecting to the ESP32-Cam video stream is configured.

Step 2. Loading the YOLO model.

The pre-trained YOLO model is loaded, including the yolov3.weights file and the yolov3.cfg configuration file. If the download is successful, a confirmation message is displayed, otherwise, an error message is displayed on the screen.

```
net = cv2.dnn.readNet(r"C:\Users\Dovgal
Dima\Desktop\esp32cam_video_stream_on_web_server\CameraWebServer\yolov3.weights",
```

r"C:\Users\Dovgal Dima\Desktop\esp32cam_video_stream_on_web_server\CameraWebServer\yolov3.cfg")

Step 3. Getting layer names.

The names of all network layers are obtained and the output layers that will be used to process the results are identified.

```
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
```

Step 4. Loading object classes.

The list of object classes is loaded from coco.names. Each class corresponds to a type of object that the model can recognize (e.g., person, car, etc.).

```
with open(r"C:\Users\Dovgal Dima\Desktop\esp32cam_video_stream_on_web_server\CameraWebServer\coco.names", "r") as f:
classes = [line.strip() for line in f.readlines()]
```

Step 5. Opening the video stream.

The connection is established with the video stream at the specified URL. The success of opening the stream is verified.

```
cap = cv2.VideoCapture(url)
```

Step 6. Reading and processing frames.

While the stream is open, frames are continuously read from the video stream. If a frame cannot be retrieved, the loop terminates.

```
while cap.isOpened():
 ret, frame = cap.read()
```

Step 7. Frame preprocessing.

The frame is converted into a format suitable for the neural network input. It is resized to 416x416 pixels and normalized.

```
blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
```

Step 8. Object prediction.

The input data is fed to the network, which returns the prediction of each frame.

```
net.setInput(blob)
outs = net.forward(output_layers)
```

Step 9. Processing predictions.

The output data is processed to calculate the coordinates and dimensions of detected objects.

```
for out in outs:
for detection in out:
```

Step 10. Filtering and displaying results.

Non-Maximum Suppression (NMS) is applied to remove redundant bounding boxes.

```
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
for i in indexes:
 x, y, w, h = boxes[i]
 label = str(classes[class_ids[i]])
 cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
 cv2.putText(frame, f"{label} {int(confidence * 100)}%", (x, y - 10), font, 1, color, 2)
```

Bounding boxes with labels indicating class names and recognition accuracy are drawn around detected objects.

Step 11. Displaying frames on the screen.

The processed frame is displayed on the screen. Pressing 'q' will exit the loop.

```
cv2.imshow("ESP32-CAM Object Detection", frame)
```

Step 12. Releasing computational resources.

Upon termination, resources are released, the video stream is closed, and the display windows are closed.

```
cap.release()
cv2.destroyAllWindows()
```

Thus, an algorithm has been developed consisting of the following main stages: loading the model, processing the video stream and displaying the results. Based on this algorithm, a system for recognizing real-time object on a video stream was implemented using the YOLO model.

Additionally, a web application was developed using the Django framework to display the video stream from the ESP32-CAM camera on a web page [34].

5. Experiments and results

The study of computing resources usage was conducted based on the following parameters:

- Frame processing time: during the test, the average time of one frame processing was measured for two video stream resolutions (320×240 and 640×480).
- Memory usage: the RAM and flash memory usage of the ESP32-CAM was monitored.
- ESP32-CAM CPU load: to evaluate the performance, the CPU load was monitored for two video stream resolutions and the average CPU load was determined (320×240 and 640×480).
- Power consumption: the power consumption was measured using a multimeter during system operation, and the experiment showed that the average power consumption of the system was 0.65W (at 5V power supply).

To evaluate the accuracy of recognition of objects of a given class by the developed system, experiments were conducted on data sets with various objects: simple objects (people) and complex objects (small items, low-contrast objects). For simple objects, the recognition accuracy was 94%, while for complex objects, the accuracy dropped to 77%, indicating the limitations of the model when processing more complex types of data.

Additional experiments were conducted to assess the impact of lighting conditions on the recognition accuracy of both simple and complex objects. These experiments showed that as illumination levels decreased, the accuracy of object recognition decreased.

The results of the experimental research of the object detection system for mobile platforms based on the ESP32-CAM are presented in Table 5. They confirmed that the system is capable of performing basic functions with high accuracy and stability under normal operating conditions. All core functions of the system, such as video stream capture, data transmission to the server, object detection, and result visualization, work correctly and meet the specified requirements.

Testing has shown that the system can recognize objects in stable lighting conditions at the proper level, but in challenging conditions such as low light or noise, recognition accuracy decreases.

Testing was also conducted to assess the stability of the results, which confirmed that the system demonstrates minimal deviations in results upon repeated testing (less than 1%). This demonstrates a high level of stability and reliability of the software under the same input data conditions.

Testing the system's performance under weak Wi-Fi conditions showed that at reduced data transfer rate (less than 5 Mbps), the system experiences frame transfer delays and periodic loss of images, which reduces the efficiency of object recognition. This indicates that the system depends on a stable Internet connection.

Experimental studies of computing resources usage, have shown that ESP32-CAM has limitations in terms of computing power. The system is capable of processing a video stream with a resolution of 320×240, however, when working with higher resolutions (640×480), there is a significant increase in CPU load and frame processing time. This indicates the need to optimise the algorithm to make more efficient use of limited resources.

The results of power consumption experiments showed that the average power consumption of the system is 0.65W, which is low enough for autonomous operation. However, for long work sessions, it is important to reduce power consumption through additional optimisation measures.

Table 5
Results of the experimental research system

Criterion	Description	Results
Frame processing time	Processing time measurement for different resolutions	320×240: 107 ms 640×480: 179 ms
Memory usage	Monitoring of RAM and flash memory usage on the ESP32-CAM	RAM: 165 KB, Flash: 3.1 MB.
CPU load of ESP32-CAM	Evaluation of CPU load during frame processing	320×240: 47.5 % 640×480: 48.54 %
Power consumption	Measurement of power consumption during system operation	Average consumption: 0.65 W
Accuracy of simple object recognition	Evaluation of object identification in the conditions of real data and comparison of results with reference data	94 %
Accuracy of complex object recognition	Evaluation of object identification in the conditions of real data and comparison of results with reference data	77 %

6. Conclusions

As a result of the work, a prototype of an object detection system was developed that allows real-time object identification using computer vision algorithms for mobile platforms with limited resources.

An analysis of mobile platforms with limited resources, image processing methods and machine learning technologies was conducted. Based on a comparative study of popular platforms, ESP32-Cam was chosen as the optimal option for developing an object detection system with limited hardware resources, taking into account its low cost, compactness and energy efficiency.

An application has been developed for the ESP32-Cam module that provides remote control of a ground drone and real-time environmental monitoring. The system enables video streaming via Wi-Fi, providing high image quality through the use of the OV2640 camera.

The developed algorithm consists of the following main stages: loading the model, processing the video stream and displaying the results is developed. Based on this algorithm, a system for real-time object recognition was implemented using the YOLO model. A web application has been developed using the Django framework that displays a video stream from the ESP32-CAM camera on a web page.

Testing and experimental study of the developed object detection system for mobile platforms based on ESP32-CAM was carried out. It has been confirmed that the developed system has a sufficient level of performance for use in stable conditions, however, further improvement is required for operation in more challenging conditions (poor lighting, low data transfer rate).

The scientific novelty of the work lies in the fact that using the YOLO model, an algorithm for recognizing objects in real-time in a video stream with high recognition reliability has been developed.

The practical value of the results of the work lies in the fact that the developed software can be used as a tracking and object recognition system for real-time environmental monitoring for mobile platforms with limited resources.

Future research will focus on further improving and expanding the system's functionality under limited computing resources, optimizing resource usage, increasing recognition accuracy in challenging conditions, and ensuring stable operation at low Wi-Fi signal levels.

Acknowledgements

This work is partly carried out with the support of Erasmus + KA2 project WORK4CE “Cross-domain competences for healthy and safe work in the 21st century” (619034-EPP-1-2020-1-UA-EPPKA2-CBHE-JP) and DAAD project ViMUk.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] U. Sharma, U.S. Medasetti, T. Deemyad, M. Mashal, V. Yadav, Mobile robot for security applications in remotely operated advanced reactors, Applied Sciences 14(6) (2024) 1-33. doi:10.3390/app14062552.

[2] T.-N. Pham, D.-T. Mai, A mobile robot design for home security systems, Engineering, Technology & Applied Science Research 14(4) (2024) 14882–14887. doi:10.48084/etasr.7336.

[3] R. Prykhodchenko, R. P. Rocha, M. S. Couceiro, People detection by mobile robots doing automatic guard patrols, in: Proceedings of the 2020 IEEE International conference on Autonomous robot systems and competitions (ICARSC), Ponta Delgada, Portugal, 2020, pp. 300-305. doi:10.1109/ICARSC49921.2020.9096147.

[4] A. Kurniawan, A. A. S. Gunawan, B. Hartanto, A. Mili, W. Budiharto, Swarm of mobile robots for security surveillance based on Android smartphone and Firebase, International Journal of Intelligent Systems and Applications in Engineering 11(4) (2023) 810–815. https://ijisae.org/index.php/IJISAE/article/view/3614.

[5] G. Fragapane, H.H. Hvolby, F. Sgarbossa, J. O. Strandhagen, Autonomous mobile robots in hospital logistics, in: Proceedings of the IFIP International conference on Advances in production management systems (APMS), Novi Sad, Serbia, 2020, pp. 672-679. doi:10.1007/978-3-030-57993-7_76.

[6] T. Lackner, J. Hermann, C. Kuhn, D. Palm, Review of autonomous mobile robots in intralogistics: state-of-the-art, limitations and research gaps, Procedia CIRP 130 (2024) 930-935. doi:10.1016/j.procir.2024.10.187.

[7] R. R. Shamshiri, I. A. Hameed, Mobile robots for digital farming, CRC Press, 2024, 208 p. doi:10.1201/9781003306283.

[8] X. Ren, B. Huang, H Yin, A review of the large-scale application of autonomous mobility of agricultural platform, Computers and Electronics in Agriculture 206 (2023) 107628. doi:10.1016/j.compag.2023.107628.

[9] M. Satoh, A. Bhat, T. Usami., K. Tatsumi, A multi-purpose autonomous mobile robot as a part of agricultural decision support systems, in: Proceedings of the 2023 IEEE 19th International conference on Automation science and engineering (CASE), Auckland, New Zealand, 2023, pp. 1-7. doi:10.1109/CASE56687.2023.10260483.

[10] D. F. Yépez Ponce, J. V. Salcedo, P.D. Rosero-Montalvo, J. Sanchis, Mobile robotics in smart farming: current trends and applications. Frontiers in Artificial Intelligence 6 (2023) 1-13. doi:10.3389/frai.2023.1213330.

[11] V. Koval, O. Semenenko, S. Baranov, S. Ostrovskyi, T. Akinina, O. Siechenev, The role and place of robotic systems in modern wars and armed conflicts: theoretical aspect, Social Development and Security 13(5) (2023) 256-276. doi:10.33445/sds.2023.13.5.24.

[12] J. Gong, J. Yan, D. Kong, D. Li, Introduction to drone detection radar with emphasis on automatic target recognition (ATR) technology, Electrical Engineering and Systems Science: Signal Processing (2023) 1-17. doi:10.48550/arXiv.2307.10326.
[13] L. Nobile, M Randazzo., M. Colledanchise, L Monorchio., W. Villa, F. Puja, L. Natale, Active exploration for obstacle detection on a mobile humanoid robot, Actuators 10(9) (2021) 1-20. doi:10.3390/act10090205.
[14] L. Mochurad, Y. Hladun, R. Tkachenko, An obstacle-finding approach for autonomous mobile robots using 2D LiDAR data, Big Data and Cognitive Computing 7(43) (2023) 1-16. doi:10.3390/bdcc7010043.
[15] C. Zaharia, V. Popescu, F. Sandu, Hardware–software partitioning for real-time object detection using dynamic parameter optimization, Sensors (Basel), 23(10) (2023) 1-25. doi:10.3390/s23104894.
[16] A. Loganathan, N. S. Ahmad, A systematic review on recent advances in autonomous mobile robot navigation, Engineering Science and Technology, an International Journal, 40 (2023) 1-26. doi:10.1016/j.jestch.2023.101343.
[17] H. Andreasson, G. Grisetti, T. Stoyanov, A. Pretto, Software architectures for mobile robots, in: Ang, M.H., Khatib, O., Siciliano, B. (Eds.) Encyclopedia of Robotics, Springer, Berlin, Heidelberg, 2023, pp.1-11. doi: 10.1007/978-3-642-41610-1_160-1.
[18] A. Eliasz, Zephyr RTOS embedded C programming, Apress Berkeley, CA, 2024, 677 p. doi:10.1007/979-8-8688-0107-5.
[19] G. Bloom, J. Sherrill, T. Hu, I. C. Bertolotti, Real-time systems development with RTEMS and multicore processors, CRC Press, Boca Raton, 2020, 534 p. doi:10.1201/9781351255790.
[20] Y. Yan, Research on the A Star algorithm for finding shortest path, Highlights in Science Engineering and Technology 46 (2023) 154-161. doi:10.54097/hset.v46i.7697.
[21] D. Kalita, P. Lyakhov, Moving object detection based on a combination of Kalman filter and median filtering, Big Data and Cognitive Computing, 6(4) (2022) 1-13. doi:10.3390/bdcc6040142.
[22] A. Aleesha, C. A. Laseena, MQTT protocol for resource constrained IoT applications: a review, in: Proceedings of the International conference on Systems, energy and environment (ICSEE 2022), Kannur, India, 2022, pp. 1–7. doi:10.2139/ssrn.4299372
[23] H. Dietz, D. Abney, P. Eberhart, N. Santini, W. Davis, E. Wilson, M. McKenzie, ESP32-CAM as a programmable camera research platform, Electronic Imaging 34 (2022) 232-1–232-6. doi:10.2352/EI.2022.34.7.ISS-232.
[24] R. Chin, Spy camera DIY wireless using ESP32 CAM and Android, Independently published, 2022, 54 p.
[25] S. Smith, RP2040 Assembly language programming, Apress Berkeley, CA, 2022, 320 p. doi:10.1007/978-1-4842-7753-9.
[26] G. Parthasarathi, U. Prethashree, A. Harish, R. Moulieshwaran, M. Shunmugadinesh, Envision – an object detection system using Jetson Nano, in: Proceedings of the 2nd International conference on Inventive computing and informatics (ICICI), Bangalore, India, 2024, pp. 542-545. doi:10.1109/ICICI62254.2024.00094.
[27] H. Fairhead, Programming the ESP32 in C using the Espressif IDF, I/O Press, 2024, 445 p.
[28] S. E. Adi, A. J. Casson, Design and optimization of a TensorFlow Lite deep learning neural network for human activity recognition on a smartphone, in: Proceedings of the 43rd Annual International conference of the IEEE Engineering in Medicine & Biology Society (EMBC), Mexico, 2021, pp. 7028-7031. doi:10.1109/EMBC46164.2021.9629549.
[29] J. Howse, Learning OpenCV 4 computer vision with Python 3, Packt Publishing, 2020, 372 p.
[30] K. Shi, Comparison of image enhancement algorithms based on denoising and edge detection, Applied and Computational Engineering, 133 (1) 2025, 174-184. doi:10.54254/2755-2721/2025.20700.
[31] L. Arreola, G. Gudiño, G. Flores, Object recognition and tracking using Haar-like features cascade classifiers: application to a quad-rotor UAV, in: Proceedings of the 8th International

conference on Control, decision and information technologies (CoDIT), Istanbul, Turkey, 2022, pp. 45-50. doi:10.1109/CoDIT55151.2022.9803981.
[32] S. Gharge, A. Patil, S. Patel, V. Shetty, N. Mundhada, Real-time object detection using Haar cascade classifier for robot cars, in: Proceedings of the 4th International conference on Electronics and sustainable communication systems (ICESC), Coimbatore, India, 2023, pp. 64-70. doi:10.1109/ICESC57686.2023.10193401.
[33] Z. Guan, Real time object recognition based on YOLO model, Theoretical and Natural Science 28(1) (2023) 137-143. DOI: 10.54254/2753-8818/28/20230450
[34] D. Dovhal, Object Detection, 2024. URL: https://github.com/Dmitriy-1986/object_detection/tree/main.