UDC 629.7.014-519.058.6:004.932 **doi: 10.32620/reks.2024.1.09**

Bohdan KARAPET, Roman SAVITSKYI, Tetiana VAKALIUK

Zhytomyr Polytechnic State University, Zhytomyr, Ukraine

METHOD OF COMPARING AND TRANSFORMING IMAGES OBTAINED USING UAV

*The **subject matter** of this article involves reviewing and developing methods for the comparison and transformation of images obtained using UAV via Computer Vision tools. The **goal** is to improve methods for image comparison and transformation. Various image-processing methods were employed to achieve the goal of this study,–thereby contributing to the development of practical algorithms and approaches for image analysis and comparison. The **tasks** can be described as follows: 1) development of image comparison methods: design tools for the comparison of images from UAV that efficiently detect differences using algorithms such as cv2.absdiff and the PIL module; 2) Image transformation: implement transformation methods for images from UAV, including perspective transformation and thresholding, to enhance the quality and accuracy of image analysis. The **methods** used were algorithm development, image transformation methods, statistical analysis, experimental testing, and performance evaluation. **The metrics** used in this article are response time and accuracy. Algorithms for image comparison have also been refined, particularly those transformed through Global Threshold Value, Adaptive Mean Thresholding, and Adaptive Gaussian Thresholding. A novel change filtering method was introduced to enhance the precision of image comparison by filtering out insignificant alterations following image transformation. Comprehensive investigation of image comparison involving edge detection methods has been systematically presented. The **results** contain the development of practical algorithms and approaches for image analysis and comparison applicable in diverse areas such as military, security, and agriculture. Possibilities of applying our methods and algorithms in the context of drones were also considered, which is particularly relevant in tasks related to computer vision in unmanned aerial vehicles, where limited resources and the need for real-time processing of a large volume of data create unique challenges. **Conclusions**. The results contain OpenCV and PIL image comparison methods. OpenCV pixel-by-pixel comparison algorithm showed a better response time with the same accuracy. OpenCV method has 92,46% response time improvement compared with PIL and is 276ms. As for image thresholding with comparison, a method based on Global Threshold Value showed the shortest response time (266ms) and the lowest accuracy. The highest accuracy and response time (366ms) were obtained using the Adaptive Gaussian Thresholding method.*

***Keywords:** UAV; computer vision; image comparison; image transformation; image processing.*

1. Introduction

1.1. Motivation

Nowadays, a crucial area of study revolves around developing innovative solutions in computer vision and unmanned aerial vehicle (UAV) technologies~~.~~ This research domain involves an analysis of image comparison and transformation tools in Computer Vision, their practical applications, and the resolution of various theoretical and practical challenges faced by modern scientific communities.

Drones with image analysis hardware can effectively analyze data and provide operators with valuable real-time information.

Computer vision has significantly evolved with technological advancements, particularly the increase in computational power and the improvement of algorithms. Following this, pivotal methodologies have emerged, including but not limited to image processing and object recognition. With the introduction of neural networks and deep learning, significant strides have been made in the field of computer vision.

This advancement empowers developers to craft sophisticated systems capable of precisely parsing images captured by drones, ensuring accurate identification and categorization of objects within both images and videos [1].

The initial methods included image filtering and enhancement, edge detection, and essential shape recognition. With the development of machine learning algorithms, object classification and localization methods have been introduced, allowing for more accurate identification and analysis of visual data.

Computer Vision plays a crucial role in many modern technologies, providing essential technological support from automated quality control in manufacturing to facial recognition in smartphones and from security systems to autonomous transportation [2].

Despite significant advancements in this area, there are situations where computer vision systems may still make errors, especially in complex lighting conditions or when detecting beautiful details.

Another crucial challenge is the speed of image processing. Processing speed is critically important for many applications, particularly those requiring real-time capabilities such as automated driving or video surveillance systems [3].

1.2. State of the Art

Over the past few years, drones have become integral components of computer vision and the Internet of Things (IoT).

Researchers such as Volodymyr Rebrov and Vladimir Lukin focus on the post-processing of compressed noisy images using BM3D filters. Researchers explored the use of the BM3D filter to improve the quality of noisy, compressed images, finding that it surpasses traditional options and offers valuable parameter setting recommendations [4].

Another important application area for drones is video stream brightness stabilization. The work of Vladyslav Bilozerskyi, Kostyantyn Dergachov, Leonid Krasnov, Anatolii Zymovin, and Anatoliy Popov tackles video brightness fluctuations caused by lighting and noise. This study proposes a novel brightness stability method using the "average frame brightness" indicator and digital filtering algorithms, achieving promising results in real-world scenarios [5].

The adaptation of FPGA architecture for accelerated image preprocessing, presented in the work of Olesia Barkovska, Inna Filippenko, Ivan Semenenko, Valentyn Korniienko, Peter Sedlaček, shown how to improve slow image processing by building a reconfigurable FPGA system, demonstrating a 60x speed-up compared to software solutions. The resilient design offers real-time reconfiguration and high throughput, making it ideal for embedded tasks requiring fast image processing [6].

Zhaolong Ning, H. Hu, and Xiaojie Wang explore the use of edge computing and machine learning in the Internet of Things (IoT) for Unmanned Aerial Vehicles (UAVs). This study examines the potential use of UAV resources for distributed data processing and real-time decision-making. The research emphasizes the significance of leveraging edge computing and machine learning to enhance efficiency and automate UAV operations in various sectors, including monitoring and rescue operations [7].

Research conducted by Arsen Petrosian, Ruslan Petrosian, Ihor Pilkevych, and Maryna Graf is focused on advancing and refining Unmanned Aerial Vehicle (UAV) control systems. The primary emphasis of this study lies in optimizing algorithms and using cloud and distributed computing for efficient management and processing of the substantial amount of data generated by drones. The research also considers the growing complexity of the drone flight environment, particularly the presence of moving obstacles [8].

Ming-You Ma, Shang-En Shen, and Yi-Cheng Huang [9], has studied areas of object detection ability and finding the emergent landing platform and for future reconnaissance.

Lijia Cao, Pinde Song, Yongchao Wang, Yang Yang, and Baoyu Peng [10] studied algorithms for image recognition of unmanned aerial vehicles for military countermeasures and disaster search and rescue.

In research, provided by Sergio Bemposta Rosende, Sergio Ghisler, Javier Fernández-Andrés, and Javier Sánchez-Soriano, different computer vision models and reduced-board (and small-power) hardware were developed, evaluated, and compared to optimize traffic management in these scenarios [11].

All of these studies emphasize the importance of using drones in various applications because they facilitate real-time data collection and analysis, which is crucial.

1.3. Objectives and Approach

This study **aims** to develop and implement methods for image comparison and transformation in the field of Computer Vision to enhance the efficiency of visual data analysis from UAVs and across various application domains.

The **tasks** can be described as follows:

1) development of image comparison methods: design tools for image comparison that efficiently detect differences using algorithms such as cv2.absdiff and the PIL module;

2) image transformation: implement image transformation methods, including perspective transformation and thresholding, to enhance the quality and accuracy of image analysis.

The assistance of Copilot was used to refactor the code and provide insights into identifying and rectifying errors encountered during the development process. ChatGPT was used to help with the article translation and to check parts of the text to avoid misinterpretations.

The approach to this research included the analysis of various image comparison methods, designing algorithms and architecture of image comparison, and improving existing algorithms.

Research contained the next stages:

- analysis of the existing image comparison methods;

- implementation PoC of image comparison algorithms;
- architecture design and solution validation;
- development;
- testing and measuring;
- improvement of existing image comparison algorithms.

In this study, we will use the following metrics: response time in milliseconds, response time compared to other methods, and accuracy.

Response time (in milliseconds) shows the amount of time from sending the request with images that we must compare to achieve the result.

Response time will also be compared between methods in this research to show differences as a percentage.

Accuracy in terms of this research is a boolean value which indicates were differences found or not. Additional details will be provided for this value.

Current research includes only methods that can be automated and do not require human interaction or routine.

2. Materials and methods of research

2.1. Pixel-wise comparison using cv2.absdiff with difference highlighting

Pixel-wise image comparison is a method that allows detection of the difference between two images at the most minor level. OpenCV (or cv2 in Python) is a popular library for implementing such a comparison. This library's cv2.absdiff() function enables the quick obtaining of an image representing the difference between two other images.

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine-learning library. Licensed under the Apache 2 license, OpenCV enables companies to easily use and modify the code [12].

The cv2.absdiff() function takes two input images and returns a new image representing their absolute difference. For each pixel in the input images, the corresponding pixels are subtracted, and the result is taken as an absolute value, forming the output image's pixels.

To make the difference more noticeable in the resulting image, you can apply the following specific highlighting methods:

- binarisation: after obtaining the difference image, you can use thresholding (cv2.threshold()) to highlight changed areas;
- colour map: to enhance visibility, you can use a colour map (cv2.applyColorMap()) that makes the difference more pronounced with contrasting colours;
- dilation: to widen and make the difference more noticeable, you can apply dilation (cv2.dilate()).

Pixel-by-pixel comparison using cv2.absdiff() is an effective method for detecting differences between two images, especially when the shooting conditions are almost 100% similar (stationary). When photos were taken at different angles, changed shooting locations, or under altered conditions, the results may be inaccurate.

2.2. Overlaying two BGR2GRAY photos

Overlaying photos, whether in colour or monochrome format, can create a visual effect useful for analyzing and demonstrating changes. Converting images from the BGR (Blue-Green-Red) format to the GRAY (monochrome) format helps emphasize brightness and contrast by discarding colour information.

When referring to overlaying two images, it implies creating a composite image where one image is superimposed on another. This can be done using various merging methods such as simple addition, averaging, and multiplication [13].

To overlay two grayscale (GRAY) images, follow these steps [14]:

1. Load images and convert them to grayscale.
2. Overlaying images. One method uses the mean squared error.
3. Displaying or saving the result.

Overlaying grayscale images can be used in various scenarios:

- difference analysis: if you have two similar photos taken at different times, overlaying them can help identify changes between them;
- information merging: for example, if you have a topographic map and a rainfall map, you can combine them to obtain an image that shows both sets of information together.

Comparing two BGR2GRAY drone photos using overlaying may show a less effective result than comparison using cv2.absdiff().

Converting to grayscale somewhat mitigates the impact of changes in the shooting angle and, to some extent, reduces the influence of lighting and weather conditions at the time of shooting.

Therefore, overlaying grayscale images helps to combine, analyze, and create new visual compositions.

2.3. Comparison using the PIL module

The Python Imaging Library (PIL) is a library for image processing in Python. It provides tools for loading, processing, and saving images in various formats. One of its key features is the ability to compare images at different levels [15].

The main comparison methods include:

- Pixel-by-pixel: this straightforward method involves checking each pixel of one image against the corresponding pixel of another image. Although this method is linear, it may not always be efficient, as minor changes in the image can result in several pixel differences.
- Histograms: comparing images using their histograms allows assessing the distribution of brightness and colour components in the image. This can be useful when finding general differences between two images rather than specific pixel discrepancies;
- shape and features: instead of directly comparing pixels or histograms, more complex algorithms can be used to detect shapes or features in an image and then compare these features between images.

PIL is well-suited for simple image comparison tasks. For example, it can be useful for detecting duplicates or checking for changes on a website.

However, it is recommended to use specialised libraries such as OpenCV for more complex tasks, such as finding differences in drone images or comparing faces,.

2.4. Image Transformation with Subsequent Comparison

Perspective Transformation

OpenCV, or Open Source Computer Vision Library, is one of the most popular libraries for computer vision. It provides a range of tools for image processing, including perspective transformation. The method cv2.warpPerspective allows altering the perspective of an image in OpenCV, which can be useful for correcting distortions or changing the viewpoint.

Key concepts [16]:

1. Defining points: to perform a perspective transformation, it is necessary to define four points on the input image and four corresponding points on the output image. These points are used to create a prospective matrix.
2. Creating a prospective matrix: the cv2.getPerspectiveTransform function takes two arrays of four points each for the input and output images and returns the prospective matrix.
3. Applying the transformation: the cv2.warpPerspective function uses the prospective matrix to change the image’s perspective.

Perspective transformation can be applied in the following cases:

- distortion correction: if a photo was taken at an angle to a flat surface, perspective transformation can be used to obtain a "top-down" view;
- changing the viewpoint: perspective transformation can be used to simulate a change in the observer's position;
- augmented reality: integrating virtual world objects into the real world requires changing the perspective to make them look natural.

Let us consider the perspective transformation on a drone image. For example, to create a map of the terrain or, more accurately, determine the coordinates of a point on the terrain by referencing landmarks, it is necessary to transform an image taken at an angle into an image with a "top-down" view [17].

The limitations of perspective transformation include the choice of key points, data loss, and computational complexity. Proper selection of corresponding points is crucial for obtaining an accurate prospective matrix. Correct choices can lead to proper transformation. Significant perspective changes may result in the loss of parts of the image or low-quality resulting images.

While OpenCV is optimized for image processing, high-resolution perspective transformation can be resource-intensive.

2.5. Global Threshold Value

Thresholding, or image binarization, is one of the most common image processing methods. This technique allows the highlighting of specific areas of an image by converting it into a binary (black and white) representation. A global threshold value means setting a single threshold value for the entire image.

The same threshold value is applied to the entire image when referring to a global threshold value.

This means that all pixels with brightness values lower than the threshold value will be converted to black and all others to white (Figure 1).

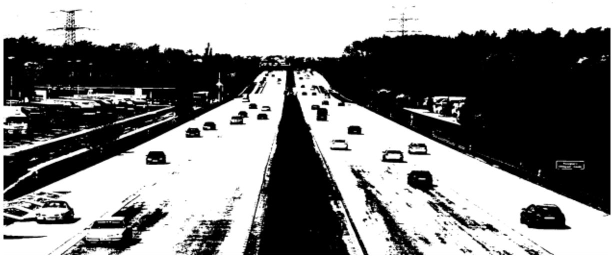

Fig. 1. Image Transformation Using a Global Threshold Value

The thresholding process consists of:

1. Loading the image: load the input image that requires binarisation.

2. Conversion to grayscale: typically, images are converted to grayscale (grey levels) to work only with brightness rather than three colour channels.

3. Applying the threshold value: all pixels with brightness below the threshold value are converted to black and others to white (Figure 2).

Advantages of global thresholding include simplicity - global thresholding is a straightforward method that does not require complex calculations or parameters. Because of its simplicity, this method is speedy and efficient for processing large images.

Limitations include suboptimal performance for imperfect images - global thresholding may need to be more efficient for images with varying lighting or low contrast.

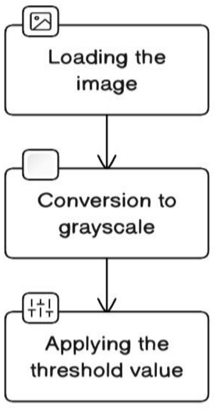

Fig. 2. Thresholding process

Another limitation is the loss of details – some details may be lost after binarisation, especially in images with low contrast between the object and background.

Therefore, image thresholding is a fundamental tool in image processing that can be used in various applications, from object highlighting to preparing data for machine learning. Global thresholding is a simple and fast method for obtaining binary images. However, its effectiveness may depend on the quality of the input image and the tasks' specific requirements.

2.6. Adaptive Mean Thresholding

Image thresholding allows the highlighting of important areas by converting them into a binary representation. Adaptive thresholding attempts to calculate an adaptive threshold for different regions of the image.

Adaptive mean thresholding, for instance, determines the threshold for each pixel by calculating the average brightness value in its neighbouring pixels (Figure 3).

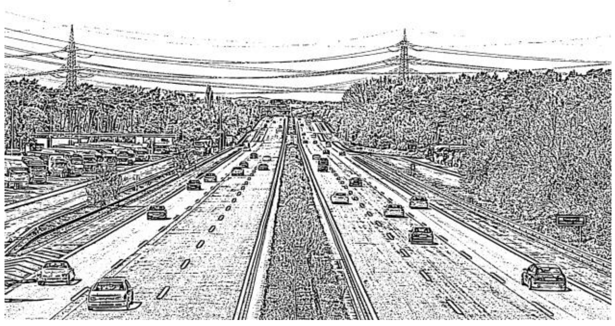

Fig. 3. Example of Applying the Adaptive Mean Thresholding

Process of adaptive mean thresholding:

1. Loading and preparing the image: load it and convert it to grayscale.

2. Choosing the window size: select the window size in pixels (e.g., 11x11) used to determine the threshold for each pixel.

3. Calculating the threshold: for each pixel in the image, calculate the average brightness value from all pixels in the selected window and use this average value as the threshold.

4. Binarisation: using the calculated threshold, convert each pixel to black or white depending on its brightness (Figure 4).

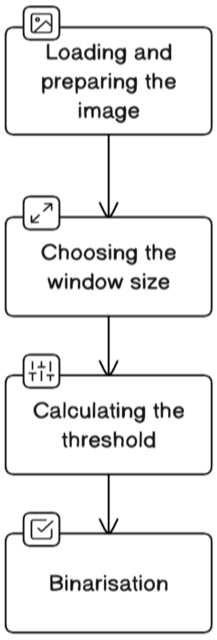

Fig. 4. Adaptive Mean Thresholding process

The advantages of the algorithm include handling uneven illumination and preserving details. Because of the local threshold calculation, this method effectively works with images with uneven lighting. In addition, it often preserves details that may be lost with global thresholding.

Limitations include computational complexity because adaptive thresholding requires more computations than global thresholding. The choice of window size is crucial, and an incorrect selection may lead to the loss of details or unwanted noise in the image.

Therefore, adaptive mean thresholding is an effective method for thresholding images with uneven illumination or contrast. Determining the threshold helps achieve more accurate binarisation than global methods. However, the correct selection of parameters, such as window size, is crucial for obtaining the best results.

2.7. Adaptive Gaussian Threshold Value

Threshold image adjustment converts images into binary (black-and-white) representations. However, not all images have a uniform background or lighting. In such cases, a global threshold value may not provide optimal results. Adaptive thresholding comes to the rescue, and one of its variants is Adaptive Gaussian thresholding (Figure 5).

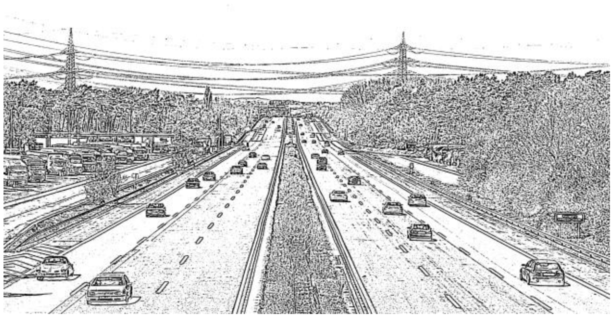

Fig. 5. Example of applying Adaptive Gaussian Thresholding

With Adaptive Gaussian thresholding, the threshold for each pixel in the image is calculated on the basis of the weighted sum of pixel intensities in the window around that pixel. A Gaussian function gives the weights, so close pixels will significantly influence the threshold value more than those farther away [18].

The process of Adaptive Gaussian Thresholding involves:

1. Loading and preparing the image: load it and convert it to grayscale.
2. Choosing the window size: select the local window size used to calculate the threshold value.
3. Calculating the threshold: for each pixel in the image, calculate the threshold based on the weighted sum of pixel intensities in the corresponding window using the Gaussian function.
4. Binarisation: apply the calculated threshold to each pixel, converting it to black or white, depending on its intensity (Figure 6).

Advantages of Adaptive Gaussian Thresholding include:

- accuracy in unevenly illuminated areas: because of the use of weighting coefficients, the method is more sensitive to local features of the image, such as shadows or highlights;
- preservation of details: simple binarisation often leads to detail loss, but the Gaussian method can retain more details because of its adaptability.

The limitations of this algorithm include higher computational complexity and the need to choose appropriate parameters. Adaptive Gaussian thresholding is computationally more demanding than other thresholding methods.

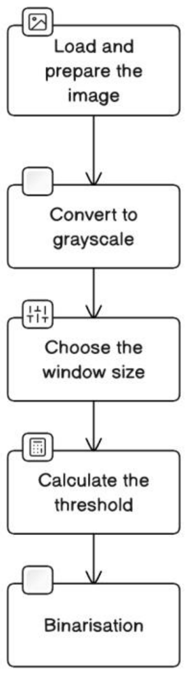

Fig. 6. Adaptive Gaussian Thresholding process

Therefore, Adaptive Gaussian Thresholding is a powerful tool for thresholding images with uneven backgrounds or lighting. It considers local image information and applies a Gaussian weighting scheme to obtain more accurate threshold values. If properly tuned, this method can provide clear, detailed binary images.

2.8. Edge Detection

Edge detection is one of the fundamental operations in image processing used to identify areas where there is a change in the intensity of brightness or colour in an image. This change in intensity may indicate an edge, which in turn can suggest the boundary between two objects or parts of an object.

Methods of edge detection [19]:

1. Gradient-based methods: using gradient operations, you can determine the direction in which the intensity of the image changes most rapidly. Key methods include:

- sobel method: uses two 3x3 kernels to determine horizontal and vertical intensity changes;
- prewitt method: similar to Sobel but with a different kernel;
- scharr's method uses larger kernels for increased accuracy.

2. Laplace method: Applies the second derivative to detect edges, which can identify edges where the image gradient passes through zero.
3. The canny method is one of the most popular and involves several steps, including smoothing, gradient detection, non-maximum suppression, and double thresholding (Figure 7).

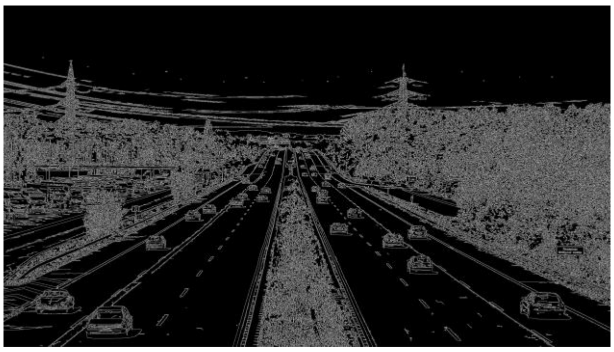

Fig. 7. Example of applying the edge detection algorithm.

Challenges of this algorithm include noise in the image, which can detect incorrect edges. Often, before edge detection, images undergo smoothing to reduce the impact of noise.

Another challenge is the discontinuous edges and thickness of the edges. Ideally, the edge should be a one-pixel-wide line. However, some methods may result in wide or blurry edges.

The algorithm is applied in various fields:

- medical imaging: for detecting pathological changes such as tumours;
- automatic object recognition: to determine the object boundaries in an image;
- video surveillance: for detecting moving objects.

Edge detection is a crucial component in image processing. It helps highlight important information in an image and can be used in various applications, from medical diagnostics to object recognition.

3. Results and Discussion

3.1. Pixel-wise Comparison Using Computer Vision

Overlaying two BGR2GRAY images and subsequent comparison.

Developing server applications to handle requests is integral to many information technologies. In this context, one of the popular libraries for creating web servers in the Python programming language is Flask.

Flask is a lightweight and flexible framework designed for rapid web application development. Using Flask, you can create a server that handles HTTP requests and sends HTTP responses [20]. Flask is based on the principles of web application development using the "routing-function" model. A URL to which HTTP requests can be sent, associated with a specific processing function, represents the «route». Processing functions define the logic of responding to requests and may include operations such as data processing and interaction with a database, and more. Creating a server with Flask involves importing the Flask class and defining routes using decorators.

In many cases, image processing on the server is an important task. Using Flask, you can create a server that accepts images as requests and returns the results of their processing.

A "compare_gray" endpoint (Listing 1) compares two images in grayscale mode. One approach to implementing this is to create the "compare_gray_image_pixels" function, which takes two images in PNG format and compares them using pixels. Subsequently, the "compare_gray" endpoint, which handles POST requests, accepts files and calls the specified function, was created.

Listing 1

Implementation of the server with the «compare_gray» endpoint

```
from flask import Flask, request,
send_file
from compare_gray_image_pixels import
compare_gray_image_pixels
app = Flask(__name__)

@app.route('/compare_gray',
methods=['POST'])
def compare_gray():
  if 'file1' is not requested.files
or 'file2' is not in request.files:
    Return 'Files are not uploaded',
400
```

```
  file1 = request.files['file1']
  file2 = request.files['file2']
  fileMimeType = file1.mime-type
  if
file1.mimetype.startswith('image/')
and
file2.mimetype.startswith('image/'):
    # Save the files to disk or
process them in memory

file1.save('./compare/pixel_compare_i
mg_1.png')

file2.save('./compare/pixel_compare_i
mg_2.png')
    # Compare the images

compare_gray_image_pixels('./compare/
pixel_compare_img_1.png',
'./compare/pixel_compare_img_2.png')
    result =
open('./compare/pixel_compare_result.
png', 'rb')
    return send_file(result,
mimetype=fileMimeType)
  else:
    return 'Invalid file type', 400
```

Let us take a closer look at the "compare_gray_image_pixels" function (Listing 2):

1. The function should include importing the cv2 (OpenCV) and imutils libraries. OpenCV is used for image processing operations [12], while imutils provides several convenient functions for working with images.
2. It takes the paths to the original and new images as arguments.
3. The original and new images are read using OpenCV and resized using the imutils.resize function.
4. A copy of the original image is created, and the absolute difference between the original and new images is computed.
5. The comparison result is converted to grayscale [13], and dilation highlights the differences.
6. Binarisation is used to determine the contours, which are then used to highlight differences in the new image [21].
7. A rectangle is drawn for each contour on the new image, and the result is saved in the "pixel_compare_result.png" file [14].
8. Adding function export using __all__.

Postman is used to test the web server endpoint. Postman simplifies the testing process, allowing efficient verification of the correctness of endpoint operations and confident implementation of changes in web applications.

The choice and use of Postman in the project were driven by its high functionality and convenience for testing and interacting with APIs.

Listing 2

Implementation of «compare_gray_image_pixels»

import cv2

```
import imutils

def
compare_gray_image_pixels(original_pa
th: str, new_path: str):
  original =
cv2.imread(original_path)
  new = cv2.imread(new_path)

  original = imutils.resize(original,
height = 600)
  new = imutils.resize(new, height =
600)

  diff = original.copy()
  cv2.absdiff(original, new, diff)

  gray = cv2.cvtColor(diff,
cv2.COLOR_BGR2GRAY)

  for i in range(0, 3):
    dilated = cv2.dilate(gray.copy(),
None, iterations= i + 1)

  (T, thresh) =
cv2.threshold(dilated, 3, 255,
cv2.THRESH_BINARY)

  cnts = cv2.findContours(thresh,
cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
  cnts = imutils.grab_contours(cnts)

  for c in cnts:
    (x, y, w, h) =
cv2.boundingRect(c)
    cv2.rectangle(new, (x, y), (x +
w, y + h), (0, 255, 0), 2)


cv2.imwrite("./compare_gray/pixel_com
pare_result.png", new)

__all__ =
['compare_gray_image_pixels']
```

The images for comparison are provided below (Figure 8). The obtained result is shown in Figure 9.

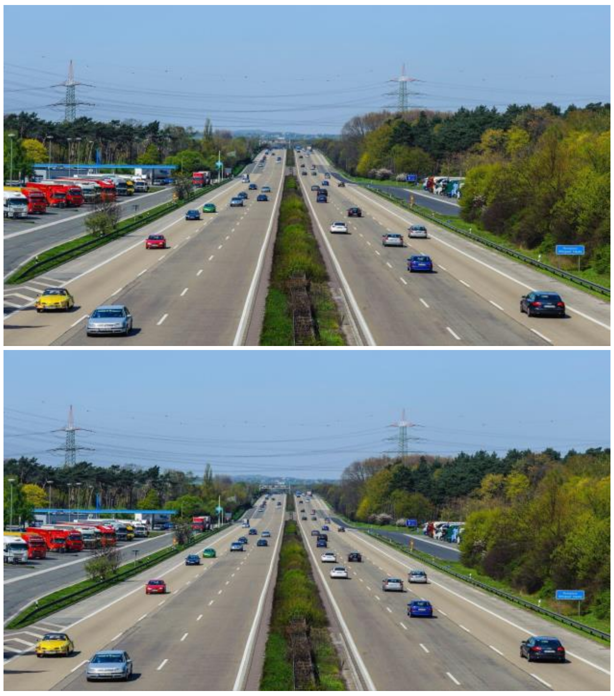
Fig. 8. Images for Comparison

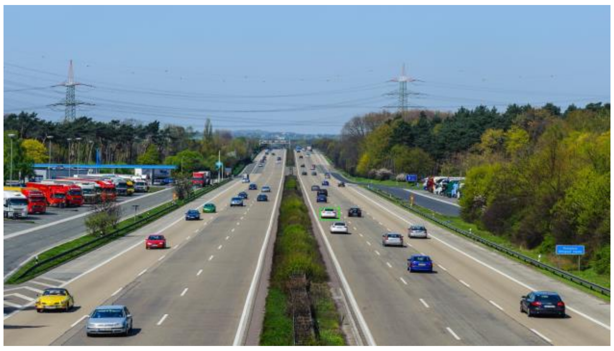
Fig. 9. Comparison Result of «compare_gray»

The average response time for images with resolution 2400x1597 pixels is 276ms. All images in this research contain one object as a difference to make measurements more accurate and to be able to find average values.

This method was chosen as a comparison point and the relative response time for it is 100%.

This method is accurate because it found the difference between two compared images.

3.2. Comparison using the PIL module

Continuing the functionality already implemented in the 'compare_gray' endpoint, a new endpoint 'compare_pillow' has been added. This new endpoint further expands the image comparison capabilities in the Flask-based web application.

The 'compare_pillow' endpoint takes two PNG format files for comparison and invokes the 'compare_pillow' function (Listing 3), which implements the photo comparison algorithm using the Pillow library. This algorithm allows for a more detailed comparison and detection of differences in various aspects of the images.

Listing 3

Implementation of the «compare_pillow» function
from PIL import Image, ImageDraw

```
import numpy as np

def compare_pillow(image1, image2):
  #Opening images and converting to
B&W
  original = Image.open(image1)
  new = Image.open(image2)

  original_converted =
original.convert("L") # I and L have
the same result
  new_converted = new.convert("L") #
I and L have the same result
  original_data =
original_converted.getdata()
  new_data = new_converted.getdata()

  #Subtracting pixels
  diff_pix =
np.subtract(original_data, new_data)

  #Creating a new image with only the
different pixels
  img_final = Image.new("L",
original.size)
  img_final.putdata(diff_pix)

  #Calculating box coordinates
  threshold = 25
  box_width = 50
  Drawer = ImageDraw.Draw(new)
  for y in range(original.size[1]):
    for x in range(original.size[0]):
      i = y * original.size[0] + x
      if abs(diff_pix[i]) >
threshold:
        Drawer.rectangle((x, y,
x+box_width, y+box_width),
outline="red", width=3)

  #Saving the image with box
```

```
new.save('./compare_pil/pillow_compar
e_result.png')

__all__ = ['compare_pillow']
```

The main idea of 'compare_pillow' is as follows:

1. Two images are read and converted to grayscale.
2. Pixel values of one image are subtracted from the other.
3. A new image is created, where each pixel represents the difference between the corresponding pixels of the original image.
4. Coordinates of rectangles (boxes) are determined to highlight the differences outlined in red on the new image.
5. The image with highlighted differences is saved in the specified directory.

For comparison, the images shown in Figure 9 were used. The obtained result is depicted in the figure below (Figure 10).

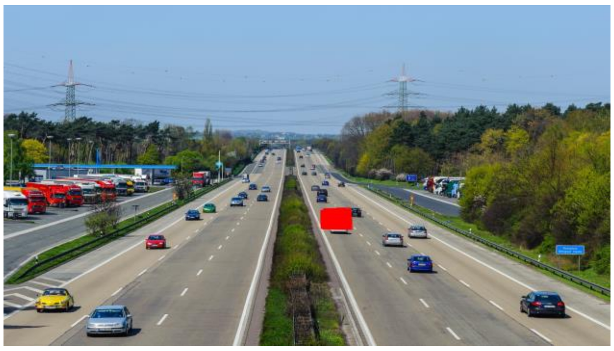

Fig. 10. Result
of the «compare_pillow» comparison

The "compare_gray" and "compare_pillow" endpoints represent two different approaches to image comparison in a Flask-based web application.

The "compare_gray" endpoint uses the OpenCV library to compare images in grayscale. This method allows for quick detection of differences, but it is sensitive to changes in lighting and colour.

On the other hand, the "compare_pillow" endpoint uses the Pillow library and presents a more flexible approach. It utilizes black-and-white versions of the images and identifies differences in pixel intensity. This method is less sensitive to colour variations and allows for more accurate highlighting of differences in images.

The average response time for image comparison via PIL is 3230ms. This algorithm showed a 1070% increase in response time compared with an algorithm based on OpenCV.

Similar to OpenCV, the current algorithm is accurate as it could find a difference between 2 images.

3.3. Image Transformation with Subsequent Perspective Comparison

Perspective transformation in the OpenCV library is a crucial aspect of image processing that allows changing and adapting the perspective of an image by considering its geometric characteristics.

We create the 'perspective_transform' endpoint, which takes a PNG file and known coordinates to perform a perspective transformation. For convenience, the coordinates are specified in percentages.

Afterwards, the 'perspective_transform' function is called, which executes the transformation of the image, considering the specified parameters and transformation settings [16].

We will not provide a listing of the 'perspective_transform' endpoint. The code for creating a new endpoint is analogous to the abovementioned ones.

Let us delve into the details of the 'perspective_transform' function (Listing 4), which is called in the endpoint.

Listing 4

Implementation
of the «perspective_transform» function.

```
import cv2
import numpy as np

def perspective_transform(img_path,
inputPoints):
  img = cv2.imread(img_path)
  assert img is not None, "file could
not be read, check with
os.path.exists()"
  height, width, channel = img.shape
  pixelCoords = [((point[0] / 100) *
width, (point[1] / 100) * height) for
point in inputPoints]
  resultPoints = np.float32([[0,
0],[width, 0],[0, height],[width,
height]])
  M =
cv2.getPerspectiveTransform(np.float3
2(pixelCoords), resultPoints)
  dst = cv2.warpPerspective(img, M,
(width, height))

cv2.imwrite('./transformation/transfo
rm_result.png', dst)
__all__ = ['perspective_transform']
```

Let us take a closer look at how it works:

1) img_path is the path to the input image that

will be transformed;

2) inputPoints is a list of coordinates on the input image that define the region for the perspective transformation. Coordinates are specified as percentages of the image width and height;

3) cv2.imread(img_path) reads the input image from the specified path;

4) assert img is not None checks if the image was successfully read;

5) pixelCoords translates percentage coordinates into pixel coordinates to determine the perspective transformation points;

6) resultPoints are the final coordinates for the transformation specified as a quadrilateral;

7) cv2.getPerspectiveTransform(np.float32 (pixelCoords),resultPoints) obtains the perspective transformation matrix based on the input and output coordinates;

8) cv2.warpPerspective(img, M, (width, height)) applies the transformation to the input image;

9) cv2.imwrite('./transformation/transform_ result.png',dst) saves the transformed image to the specified path.

The photo for the transformation and the results are shown in Figure 11.

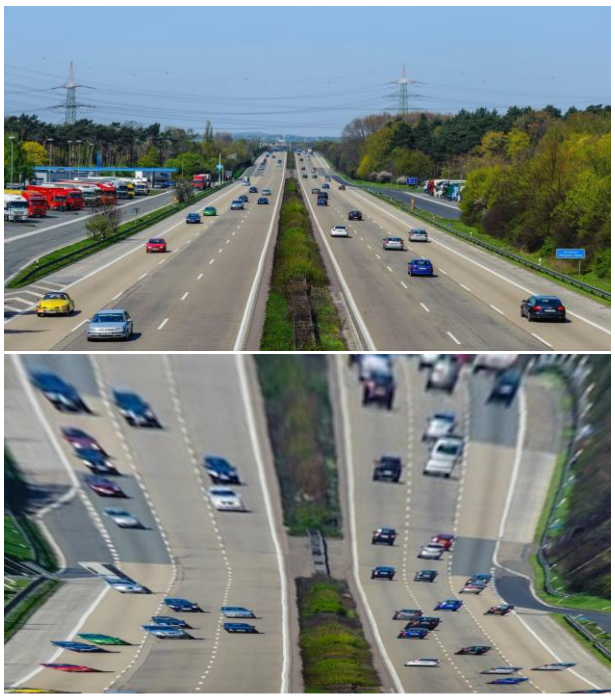

Fig. 11. Perspective Transformation Result

3.4. Global threshold value (V=127)

The next endpoints focus on image thresholding and subsequent comparison. In particular, the global_thresholding endpoint will receive two images in PNG format and invoke the global_thresholding function. In this function, a global threshold is applied to each pixel in the image, changing its value based on the threshold value. The modified images represent the result of threshold processing [22].

After obtaining the modified images, the global_thresholding endpoint invokes the compare_thresholded_image method, comparing the corresponding pixels on the two images and highlighting areas where changes occurred due to threshold processing.

This approach efficiently identifies differences between two photos resulting from global threshold value changes in pixel values. Let us now provide the code implementation for applying a global threshold value (Listing 5).

Listing 5

Implementation
of the «global_thresholding» function

```
import cv2

def global_thresholding(image_path,
result_path):
  img = cv2.imread(image_path,
cv2.IMREAD_GRAYSCALE)
  assert img is not None, "file could
not be read, check with
os.path.exists()"
  img = cv2.medianBlur(img, 5)
  thresholdedImage =
cv2.threshold(img, 127, 255,
cv2.THRESH_BINARY)[1]
  cv2.imwrite(result_path,
thresholdedImage)

__all__ = ['global_thresholding']
```

Next, we carefully examine the code of the "compare_thresholded_image" function (Listing 6). This function plays a crucial role in comparing images that have undergone global thresholding. We will analyze each step of this code, revealing how the comparison and determination of differences between images occur.

Listing 6

Implementation of the «compare_thresholded_image» function

```
import cv2

def
compare_thresholded_image(original_im
g_path: str, new_img_path: str,
result_img_path: str,
min_contour_area: int = 100):
  original =
cv2.imread(original_img_path)
  new = cv2.imread(new_img_path)

  result = new.copy()
  diff = cv2.absdiff(original, new)

  diff_gray = cv2.cvtColor(diff,
cv2.COLOR_BGR2GRAY)

  _, thresh =
cv2.threshold(diff_gray, 25, 255,
cv2.THRESH_BINARY)

  contours, _ =
cv2.findContours(thresh,
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

  contours = [cnt for cnt in contours
if cv2.contourArea(cnt) >
min_contour_area]
  for contour in contours:
    (x, y, w, h) =
cv2.boundingRect(contour)
    cv2.rectangle(result, (x, y), (x
+ w, y + h), (0, 255, 0), 2)

  cv2.imwrite(result_img_path,
result)

__all__ =
['compare_thresholded_image']
```

First, the function loads the original and new images. Then, it calculates the difference between them using the cv2.absdiff() function, which subtracts the value of each pixel in one image from the corresponding pixel in the other.

The obtained difference is transformed into a black-and-white image, and thresholding is applied to highlight areas where changes have occurred. The contours in this image help identify the boundaries of the altered areas.

The function uses the min_contour_area parameter to filter out small contours, which can be useful for ignoring minor changes or noise in the image.

Finally, the contours are displayed on the new image with green rectangles highlighting the changed areas. The comparison result is saved at the specified result_img_path.

It is worth noting that the function also saves the original and new images in the compare_thresholded_image.original and compare_thresholded_image.new properties, respectively.

Photos for comparison are shown in Figure 9. Below are the results of applying a global threshold, comparing two photos, and highlighting the differences (Figures 12 and 13).

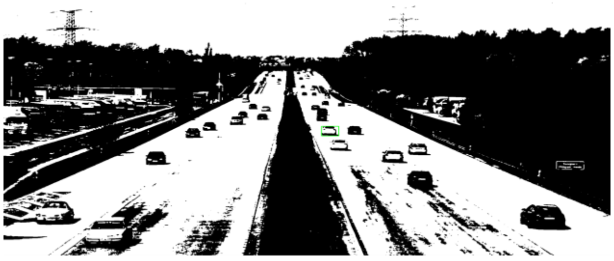

Fig. 12. Comparison of the results of photos with an applied global threshold value and noise filtering

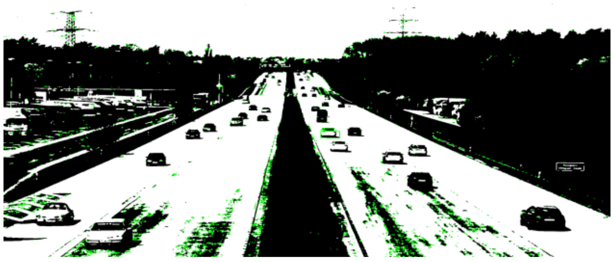

Fig. 13. Comparison of the results of photos with the applied global threshold value without noise filtering

The average response time for transforming images via Global threshold value and subsequent comparison is 266ms. Transforming images via Global threshold value was chosen as a comparison point and relative response time for it is 100%.

This algorithm is based on OpenCV image comparison and shows accurate results. The difference was found and the response time was improved.

3.5. Adaptive mean threshold value

Similar to comparing images using a global threshold value, we are introducing the "mean_thresholding" endpoint (Listing 7). This endpoint identifies the changed areas in photos by applying the mean thresholding method.

Listing 7
Implementation of the «mean_thresholding» function

```
import cv2
def mean_thresholding(image_path, 
result_path):
 img = cv2.imread(image_path, 
cv2.IMREAD_GRAYSCALE)
 assert img is not None, "file could 
not be read, check with 
os.path.exists()"
 img = cv2.medianBlur(img,5)
 thresholdedImage = 
cv2.adaptiveThreshold(img, 255, 
cv2.ADAPTIVE_THRESH_MEAN_C,
   cv2.THRESH_BINARY, 11, 2)
 cv2.imwrite(result_path, 
thresholdedImage)

__all__ = ['mean_thresholding']
```

After obtaining the modified images using this method, their comparison is conducted using the "compare_thresholded_image" function. Using the images from Figure 8, we obtain the result in the image below (Figure 14).

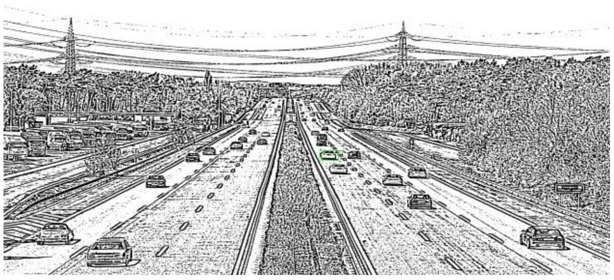

Fig. 14. Result of comparing images using adaptive thresholding

The average response time for transforming images via Adaptive mean threshold value is 350ms. This method shows a 32% increase compared to Global threshold value.

This algorithm is also based on OpenCV image comparison and shows accurate results.

The difference between the previous and current algorithms is that Adaptive mean threshold shows more detailed results. For example, a car was shown as a single difference. In the Global threshold value algorithm, the car was shown as two objects.

3.6. Adaptive Gaussian thresholding

The last example of thresholding presented in this study is adaptive Gaussian thresholding. The 'gaussian_thresholding' endpoint (Listing 8) invokes the corresponding function to perform adaptive Gaussian thresholding and compares the resulting transformed images [21].

The request in Postman is similar to the previous endpoints and contains two images.

The comparison result is shown in the figure below (Figure 15).

Listing 8
Implementation
of the «gaussian_thresholding» function.

```
import cv2

def gaussian_thresholding(image_path, 
result_path):
 img = cv2.imread(image_path, 
cv2.IMREAD_GRAYSCALE)
 assert img is not None, "file could 
not be read, check with 
os.path.exists()"
 img = cv2.medianBlur(img,5)
 thresholdedImage = 
cv2.adaptiveThreshold(img,255,cv2.ADA
PTIVE_THRESH_GAUSSIAN_C,\
   cv2.THRESH_BINARY,11,2)
 cv2.imwrite(result_path, 
thresholdedImage)
__all__ = ['gaussian_thresholding']
```

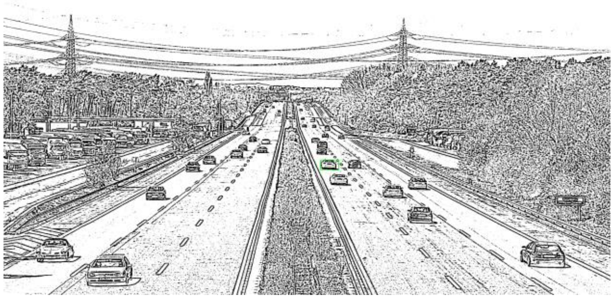

Fig. 15. The result of comparing the images with the application of adaptive Gaussian thresholding

Table 1 presents the response time analysis for various image processing methods, including OpenCV, PIL (Python Imaging Library), Global Thresholding, Adaptive Thresholding, and Gaussian Thresholding. The response time was measured in milliseconds, and the percentage values were calculated relative to the response time of the OpenCV method, which served as the baseline (100%).

The analysis of the presented table (see Table 1) reveals distinct response time characteristics for different image processing methods.

Notably, the OpenCV method exhibited a response time of 276 ms, serving as the baseline with a relative percentage of 100%. In contrast, the PIL method demonstrated a considerably higher response time of 3,230 ms, representing an increase of approximately 1170% compared with OpenCV. The Global threshold value method showed a response time of 266 ms, aligning closely with the OpenCV baseline. However, the Adaptive mean threshold value and Adaptive Gaussian thresholding methods displayed response times of 350 ms and 366 ms, respectively, corresponding to a percentage increase of approximately 132% for both.

Table 1

Response time analysis for various image-processing methods

Method	Response time, ms	Response time, %
OpenCV	276	100
PIL	3 230	1170
Global threshold value	266	100
Adaptive mean threshold value	350	132
Adaptive Gaussian thresholding	366	132

Variation in response times underscores the importance of selecting an appropriate image processing method based on the specific requirements and constraints of the application.

Based on the results of the research, we can definitely say that OpenCV provides a more effective pixel-by-pixel comparison. OpenCV consumes fewer resources and provides faster results.

The decision that images thresholding algorithm to choose can be made based on requirements to accuracy and available resources.

The use of global threshold values consumes fewer resources and provides faster results. However, the results provide worse accuracy compared to other methods.

Adaptive mean threshold values and adaptive Gaussian thresholding provide results that are more accurate but consume more resources.

With additional configuration, we can increase the number of pixels (window around pixel) that will affect pixel transformation. Such configuration increases the accuracy of image comparison, but requires more resources.

Based on available computer capacity and requirements for accuracy, the decision on which thresholding method should be made. For example, for devices with low computer capacity and low demand for comparison accuracy, it is suggested to use a global threshold value.

3.7. Edge Detection

The edge detection method, represented by the "edge_detection" endpoint, proved interesting for further image comparison. This method highlights the prominent edges of objects in the image, allowing identification of their shapes and positions. However, its use is broader than just detecting the contours of objects.

The "edge_detection" endpoint (Listing 9) can be an effective tool for identifying changes in images, as changes often accompany alterations in the contours of objects. This approach enables highlighting areas where changes have occurred, which can be useful for analyzing the dynamics of changes in an image.

The edge detection and comparison results are shown in the figure below (Figure 16).

Listing 9

Implementation of the «detect_edges» function

```
import cv2
import numpy as np

def detect_edges(image_path,
result_path):
  img = cv2.imread(image_path)
  gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY)
  edges = cv2.Canny(gray, 50, 150,
apertureSize = 3)

  cv2.imwrite(result_path, edges)

__all__ = ['detect_edges']
```

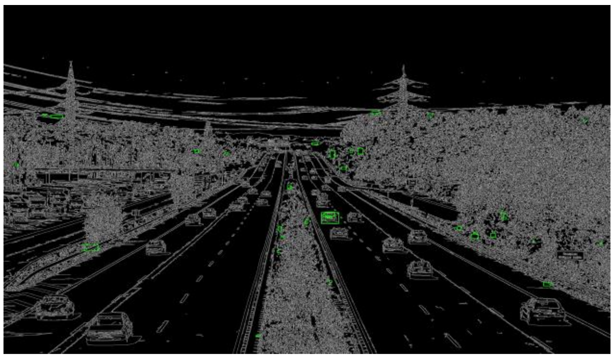

Fig. 16. Result of comparing the images with detected edges

4. Conclusions

Efficient image comparison algorithms were developed and improved, enabling the detection of minor differences and highlighting similarities with high precision. These algorithms served as a basis for further research and ensured high accuracy in detecting changes in images.

In addition, image transformation methods were optimized to enhance quality and prepare them for further analysis. This approach improved working with visual data and provided more accurate results in various tasks.

Measurements collected during the research showed that the OpenCV method had a response time of 276 ms (100%), whereas PIL had a significantly higher response time of 3,230 ms (~1170% increase). The Global threshold value method closely matched the OpenCV baseline at 266 ms. However, the Adaptive mean threshold value and Adaptive Gaussian thresholding methods had response times of 350 ms and 366 ms, respectively, representing approximately a 132% increase for both.

For future scientific research, it is recommended that efforts be focused on improving algorithms to reduce computational complexity and increase processing speed. Exploring opportunities for integrating the developed methods into other computer systems and software complexes is also essential. Further development of deep learning algorithms could be a crucial direction, providing even greater accuracy and efficiency when working with visual data.

Author Contributions

Conceptualization, methodology – **Roman Savitskyi**; formulation of tasks, analysis – **Bohdan Karapet**, **Roman Savitskyi**; development of model, software, verification – **Bohdan Karapet**; analysis of results, visualization – **Bohdan Karapet**, **Tetiana Vakaliuk**; writing – original draft preparation, writing – review and editing – **Tetiana Vakaliuk**.

Financing

This study was conducted without financial support.

Conflicts of Interest

The authors declare no conflict of interest.

Data availability

Data will be made available upon reasonable request.

Use of Artificial Intelligence

The authors have used artificial intelligence technologies within acceptable limits to provide their own verified data, as described in the research methodology section.

Acknowledgments

The authors would like to thank the scientific and teaching staff of Zhytomyr Polytechnic State University for their support.

All the authors have read and agreed to the published version of this manuscript.

References

1. Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., & Ghayvat, H. CNN variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, 2021, vol. 10, iss. 20, article no. 2470. DOI: 10.3390/ electronics10202470.
2. Davies, E. R. *Computer vision: principles, algorithms, applications, learning*. Academic Press Publ., 2018. DOI: 10.1016/C2015-0-05563-0.
3. Janai, J., Güney, F., Behl, A., & Geiger, A., 2020. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 2020, vol. 12, iss. 1-3, pp. 1-308. DOI: 10.1561/0600000079.
4. Rebrov, V., & Lukin, V. Post-processing of compressed noisy images using BM3D filter. *Radioelectronic and Computer Systems*, 2023, no. 4, pp. 100-111. DOI: 10.32620/reks.2023.4.09.
5. Bilozerskyi, V., Dergachov, K., Krasnov, L., Zymovin, A., & Popov, A. New method for video stream brightness stabilization: algorithms and performance evaluation. *Radioelectronic and Computer Systems*, 2023, no. 3, pp. 125-135. DOI: 10.32620/reks.2023.3.10.
6. Barkovska, O., Filippenko, I., Semenenko, I., Korniienko, V., & Sedlaček, P. Adaptation of FPGA architecture for accelerated image preprocessing. *Radioelectronic and Computer Systems*, 2023, no. 2, pp. 94-106. DOI: 10.32620/reks.2023.2.08.
7. Ning, Z., Hu, H., Wang, X., Guo, L., Guo, S., Wang, G., & Gao, X. Mobile edge computing and machine learning in the internet of unmanned aerial vehicles: A survey. *ACM Computing Surveys*, 2023, vol. 56, iss. 1, article no. 13, pp. 1-31. DOI: 10.1145/3604933.
8. Petrosian, A. R., Petrosyan, R. V., Pilkevych, I. A., & Graf, M. S. Efficient model of PID controller of unmanned aerial vehicle. *Journal of Edge Computing*,

2023, vol. 2, iss. 2, pp. 104–124. DOI: 10.55056/jec.593.

9. Ma, M.-Y., Shen, S.-E., & Huang, Y.-C. Enhancing UAV Visual Landing Recognition with YOLO's Object Detection by Onboard Edge Computing. *Sensors*, 2023, vol. 23, iss. 21, article no. 8999. DOI: 10.3390/s23218999.

10. Cao, L., Song, P., Wang, Y., Yang, Y., & Peng, B. An Improved Lightweight Real-Time Detection Algorithm Based on the Edge Computing Platform for UAV Images. *Electronics*, 2023, vol. 12, iss. 10, article no. 2274. DOI: 10.3390/ electronics12102274.

11. Bemposta Rosende, S., Ghisler, S., Fernández-Andrés, J., & Sánchez-Soriano, J. Implementation of an Edge-Computing Vision System on Reduced-Board Computers Embedded in UAVs for Intelligent Traffic Management. *Drones*, 2023, vol. 7, iss. 11, article no. 682. DOI: 10.3390/drones7110682.

12. Gollapudi, S. OpenCV with Python. In: *Learn Computer Vision Using OpenCV*, Apress, Berkeley, CA, 2019, pp. 31-50. DOI: 10.1007/978-1-4842-4261-2_2.

13. Yulina, S. Implementation of Haar Cascade Classifier for Face Detection and Grayscale Image Transformation Using OpenCV. *Jurnal Komputer Terapan*, 2021, vol. 7, iss. 1, pp. 100-109. DOI: 10.35143/jkt.v7i1.3411.

14. Chadha, A., Kashyap, S., Gupta, M., & Kumar, V. License plate recognition system using OpenCV & PyTesseract. *CSI Journal of Computing*, 2020, vol. 3, iss. 3, pp. 31-35. Available at: https://www.researchgate.net/profile/Jyoti-Deone/publication/348232599_CSI_Journal_of_COMPUTING/links/5ff44163299bf14088707fa8/CSI-Journal-of-COMPUTING.pdf?_sg[0]=started_experiment_milestone&origin=journalDetail&_rtd=e30%3D (Accessed 17 Jan. 2024).

15. Lindblad, T., & Kinser, J. M. NumPy, SciPy and Python Image Library. *Image Processing using Pulse-Coupled Neural Networks. Biological and Medical Physics, Biomedical Engineering*. Springer, Berlin, Heidelberg, 2013, pp. 35-56. DOI: 10.1007/978-3-642-36877-6_3.

16. Zhang, Ju., Zhang, Ji., Chen, B., Gao, J., Ji, S., Zhang, X., & Wang, Z. A perspective transformation method based on computer vision. *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, Dalian, China, 2020, pp. 765-768. DOI: 10.1109/ICAICA50127.2020.9182641.

17. Batubara, M. A., Alam, S., Nisa, K., Utami, R. W., Fitriawan, H., & Ulvan, A. Estimating the number of trees in Margasari mangrove forests of Lampung through aerial images using adaptive thresholding and contour extraction methods. *Proceedings of the International Conference on Sustainable Biomass (ICSB 2019)*, Atlantis Press Publ., 2021, vol. 202, pp. 131-136. DOI: 10.2991/aer.k.210603.022.

18. Rehman, N. A., & Haroon, F. Adaptive Gaussian and double thresholding for contour detection and character recognition of two-dimensional area using computer vision. *Engineering Proceedings*, 2023, vol. 32, iss. 1, article no. 23. DOI: 10.3390/ engproc2023032023.

19. Sultana, H., Kamal, A. H. M., Apon, T. S., & Alam, M. G. R. Increasing embedding capacity of stego images by exploiting edge pixels in prediction error space. *Cyber Security and Applications*, 2024, vol. 2, article no. 100028. DOI: 10.1016/j.csa.2023.100028.

20. Grinberg, M. *Flask web development: developing web applications with Python*. 2nd edition. O'Reilly Media, Inc., 2018. 312 p. ISBN: 978-1491991732.

21. Orban, C. *How to track objects with stationary background?* Available at: https://www.authentise.com/post/how-to-track-objects-with-stationary-background (Accessed 17 Jan. 2024).

22. Devi, R. Geometric transformations and thresholding of images using Opencv-Python. *GRD Journal for Engineering*, 2017, vol. 2, iss. 11, pp. 49-52. Available at: https://www.grdjournals.com/uploads/article/GRDJE/V02/I11/0020/GRDJEV02I110020.pdf (Accessed 17 Jan. 2024).

Received 15.01.2024, Accepted 20.02.2024

МЕТОД ПОРІВНЯННЯ ТА ТРАНСФОРМУВАННЯ ЗОБРАЖЕНЬ, ОТРИМАНИХ ЗА ДОПОМОГОЮ БПЛА

Богдан Карапет, Роман Савіцький, Тетяна Вакалюк

Предметом цього дослідження є перегляд і розробка методів порівняння та трансформації зображень, отриманих за допомогою БПЛА, засобами комп'ютерного зору. **Мета** полягає в удосконаленні методів порівняння та трансформації зображень. Для досягнення цієї мети використовувалися різні методи обробки зображень, включаючи порівняння та трансформацію зображень, що сприяло розробці практичних алгоритмів і підходів для аналізу та порівняння зображень. **Завдання** можна сформулювати наступним чином: 1) розробка методів порівняння зображень: створення інструментів для порівняння зображень, отриманих за допомогою БПЛА, які ефективно виявляють відмінності за допомогою алгоритмів, таких як cv2.absdiff та модуль PIL; 2) трансформація зображень: впровадження методів трансформації зображень з БПЛА, включаючи перспективну трансформацію та порогову зміну, для покращення якості та точності аналізу зображень. Використані **методи** включають розробку алгоритмів, методи трансформації зображень, статистичний аналіз, експериментальне тестування та оцінку продуктивності. **Метрики**, які були використані в цій статті, - це час відгуку, точність та ефективність. Дослідження та вдосконалення порівняння зображень за допомогою OpenCV і PIL є важливим аспектом цього дослідження. Також було покращено алгоритми порівняння зображень, які були трансформовані за допомогою Global Threshold Value, Adaptive Mean Thresholding та Adaptive Gaussian Thresholding. Було введено новий метод фільтрації змін, щоб підвищити точність порівняння зображень шляхом фільтрації незначних змін після трансформації зображення. Крім того, було систематично представлено комплексне дослідження та пояснення порівняння зображень із застосуванням методів виявлення країв. Отримані **результати** включають розробку ефективних алгоритмів та підходів для аналізу та порівняння зображень, що можуть бути використані у різних областях, таких як військова, безпекова, сільськогосподарська тощо. Зважаючи на зростаючий інтерес до БПЛА, також було враховано можливості застосування наших методів та алгоритмів у контексті зображень, отриманих за допомогою БПЛА. Що особливо актуально в задачах, пов'язаних з комп'ютерним зором у безпілотних літальних апаратів, де обмежені ресурси та необхідність обробки великого обсягу даних в реальному часі створюють унікальні виклики. Результати містять методи порівняння зображень OpenCV та PIL. Алгоритм порівняння пікселів OpenCV показав кращий час відгуку з такою ж точністю. Метод OpenCV показав поліпшення часу відгуку на 92,46% у порівнянні з PIL та становить 276 мс. Щодо порівняння зображень з використанням порогової обробки, метод, заснований на глобальному значенні порогу, показав найкоротший час відгуку (266 мс) та найнижчу точність. Найвищу точність та час відгуку (366 мс) показав метод адаптивної гаусівської порогової обробки.

Ключові слова: БПЛА; Computer Vision; порівняння зображень; трансформація зображень; обробка зображень.

Карапет Богдан Васильович – здобувач вищої освіти каф. інженерії програмного забезпечення, Державний університет «Житомирська політехніка», Житомир, Україна.

Савіцький Роман Святославович – старш. викл. каф. інженерії програмного забезпечення, Державний університет «Житомирська політехніка», Житомир, Україна.

Вакалюк Тетяна Анатоліївна – д-р пед. наук, проф., зав. каф. інженерії програмного забезпечення, Державний університет «Житомирська політехніка», Житомир, Україна.

Bohdan Karapet – Student of the Department of Software Engineering, Zhytomyr Polytechnic State University, Zhytomyr, Ukraine,
e-mail: bogdankarapet@gmail.com, ORCID: 0009-0009-4104-8362.

Roman Savitskyi – Senior Lecturer of the Department of Software Engineering, Zhytomyr Polytechnic State University, Zhytomyr, Ukraine,
e-mail: roman.savitskyi@gmail.com, ORCID: 0000-0001-9804-3604.

Tetiana Vakaliuk – Doctor of Pedagogical Sciences, Professor, Head of the Department of Software Engineering, Zhytomyr Polytechnic State University, Zhytomyr, Ukraine,
e-mail: tetianavakaliuk@gmail.com, ORCID: 0000-0001-6825-4697, Scopus Author ID: 57211133927.