Kazakh-British Technical University

Introduction to Computer Vision

Laboratory Work #1

Image Preprocessing with OpenCV (Variant 2)

Full Name: Sekenova Balym

ID: 23B031433

Almaty

January 27, 2026

Contents

1 Introduction

The purpose of this laboratory work is to study the main image preprocessing techniques used in computer vision. During this work, an image was loaded, processed, and analyzed using OpenCV and Matplotlib. The main goal was to understand how different preprocessing techniques affect pixel values, image structure, and visual quality.

2 Part 1: Environment Setup and Image Import

A Google Colab environment was used for this laboratory work. The OpenCV, NumPy, and Matplotlib libraries were imported. An image was uploaded and loaded using OpenCV.

The `image.shape` command shows the height, width, and number of channels of the image. At first, the concept of image channels was unclear, so additional research was done. Image channels refer to individual components of color in an image. In RGB images, each channel corresponds to red, green, or blue intensity values.

In my own words, the number of channels indicates that the image is a color image (three channels). The data type `uint8` means that pixel values are stored in the range from 0 to 255.

3 Part 2: Color Space Processing

In this task, the image was converted from the BGR color space to RGB, and then from RGB to HSV. RGB is intuitive: R stands for red, G for green, and B for blue.

Initially, HSV was unfamiliar. After studying it, I understood that HSV separates color information from brightness. HSV stands for Hue, Saturation, and Value. Hue represents the color, Saturation represents color intensity, and Value represents brightness. This separation makes HSV useful for image segmentation and illumination-invariant analysis.

Each HSV channel was visualized separately to better understand the role of each component.

4 Part 3: Image Transformation

In this task, several transformations were applied to the image. First, the image was resized to a fixed size of 256×256 pixels. Then, the image was rotated by 45 degrees around its center using an affine transformation.

Translation was also applied. At first, the concept of translation was unclear, but it can be understood as dragging the image on the screen. The image was shifted horizontally and vertically without changing its shape.

All transformed images were displayed and compared with the original image.

Image transformations such as resizing, rotation, and translation are commonly used in computer vision for data augmentation and geometric normalization, especially before applying machine learning models.

5 Part 4: Contrast Enhancement and Normalization

To perform histogram operations, the image was converted to grayscale. A color image consists of three channels, but grayscale uses only one channel, meaning each pixel is represented by a single intensity value.

Histogram equalization was applied to enhance contrast. An image can be understood as a matrix of numbers where each pixel has a value between 0 and 255. A histogram shows how often each brightness value appears. Histogram equalization redistributes pixel intensities across the full range, making dark areas darker and bright areas brighter, thus increasing contrast.

The `ravel()` function was used to convert the 2D image matrix into a 1D array, which is required for histogram plotting. Pixel values were also normalized to the range [0, 1] to make the data suitable for further processing.

Contrast enhancement is particularly important in real-world images where lighting conditions are uneven, as it improves visibility of details in both dark and bright regions.

6 Part 5: Noise Filtering and Edge Detection

Gaussian blur was applied to reduce noise in the image. Gaussian blur works by replacing each pixel with a weighted average of its neighboring pixels. Pixels closer to the center have higher weight, following a Gaussian distribution. The kernel size (5, 5) controls the strength of the blur.

After blurring, Sobel edge detection was applied. First, horizontal gradients (x-direction) were computed, which detect vertical edges. Then, vertical gradients (y-direction) were computed, which detect horizontal edges. These gradients were combined using the gradient magnitude formula:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

Blurring was performed before edge detection because noise can appear as false edges. Blurring makes edges smoother and more meaningful.

Canny edge detection was also applied. Canny is a multi-step algorithm, not a single filter. The lower and upper thresholds (100 and 200) control which edges are kept. Strong edges are preserved, weak edges are discarded, and medium edges are kept only if they are connected to strong ones. This results in cleaner and more meaningful edges.

7 Part 6: Binary Image and Object Extraction

Thresholding was used to convert the image into a binary image, where pixel values are either 0 (black) or 255 (white). Otsu's method was used to select the threshold automatically. This method analyzes the grayscale histogram and finds the threshold that best separates background and foreground pixels.

Binary images make object shapes very clear, which is necessary for contour extraction. A contour can be understood as a sequence of points that form the boundary of an object, similar to tracing the outline with a pen.

Only external contours were extracted using `cv2.RETR_EXTERNAL`, which is useful for object counting. The `cv2.CHAIN_APPROX_SIMPLE` method was used to remove redundant points and store only important contour points.

Contours were drawn on a copy of the original image. From contours, it is possible to compute object shape, area, perimeter, and object count. Contours are commonly used in segmentation and object tracking.

Because contours describe object boundaries mathematically, they are widely used in applications such as object detection, measurement, and tracking.

8 Results

The applied preprocessing techniques successfully enhanced image quality and highlighted important structures. Histogram equalization improved contrast, edge detection revealed object boundaries, and contour extraction allowed object shape analysis.

Overall, the Canny edge detector produced cleaner and more continuous edges compared to the Sobel operator. While Sobel highlights edge intensity and direction, it is more sensitive to noise. Canny, due to its multi-stage processing and thresholding strategy, better preserves meaningful object boundaries and suppresses noise.

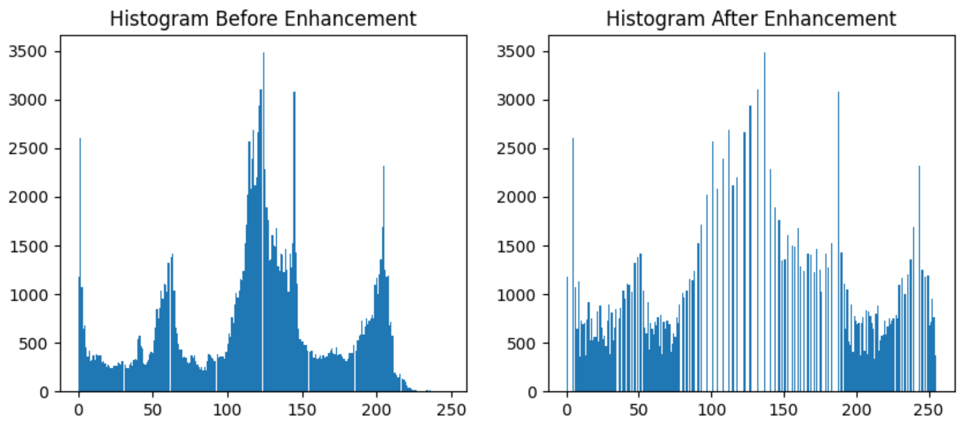


Figure 1: Histogram before and after contrast enhancement. Task 4

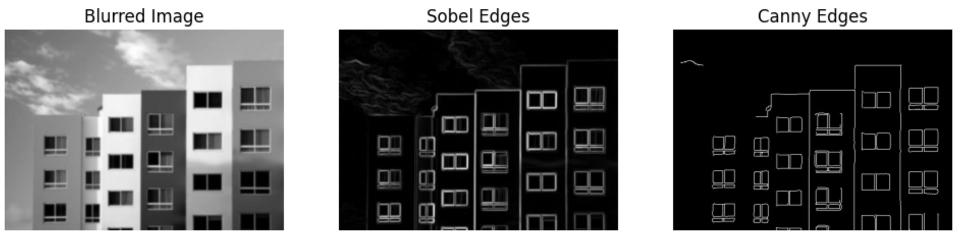


Figure 2: Comparison of Gaussian blurred image, Sobel edge detection, and Canny edge detection. Task 5

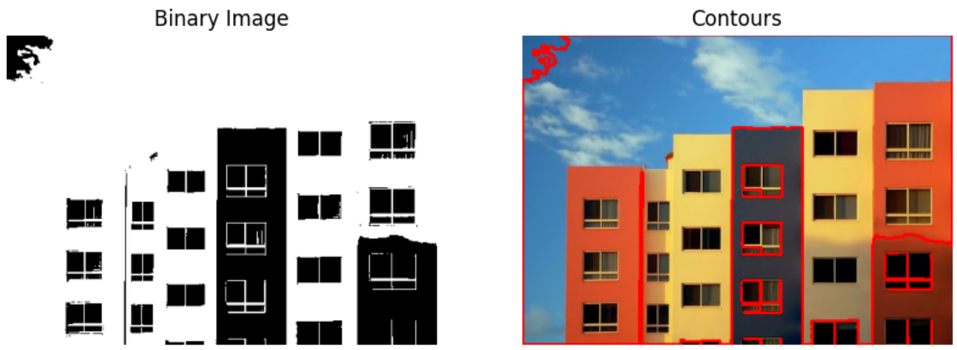


Figure 3: Binary image obtained using thresholding and extracted object contours. Task 6

9 Conclusion

In this laboratory work, fundamental image preprocessing techniques were studied and applied. Through practical implementation, a deeper understanding of image representation, contrast enhancement, noise filtering, edge detection, and object extraction was achieved. The results demonstrate that preprocessing significantly improves image quality and plays a crucial role in preparing data for further computer vision and machine learning applications.

10 References

- OpenCV Documentation: `https://docs.opencv.org`
- Computer Vision lectures by Koishiyeva Dina
- Google Colab Notebook: `https://colab.research.google.com/drive/14Yh9q4ZWlgP7nu5LY` `usp=sharing`
- Google Drive as repository: `https://drive.google.com/drive/folders/1xHxL3V-KNcoS46o` `usp=sharing`

A Appendix

Additional screenshots, processed images, and outputs demonstrating the work results are provided in this section.

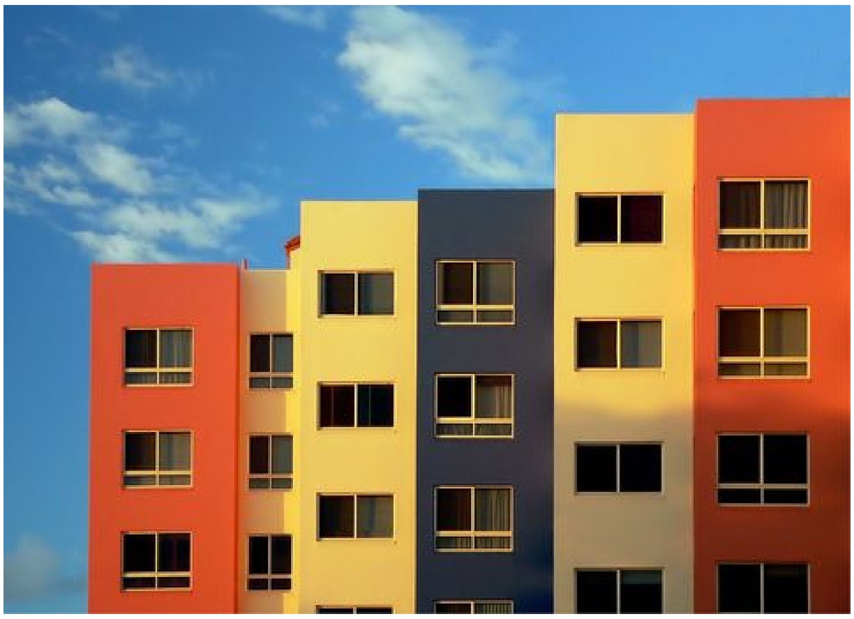


Figure 4: Original image used in the laboratory work

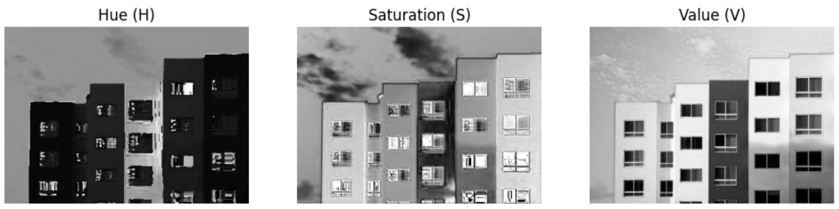


Figure 5: Visualization of Hue (H), Saturation (S), and Value (V) channels. Task 2

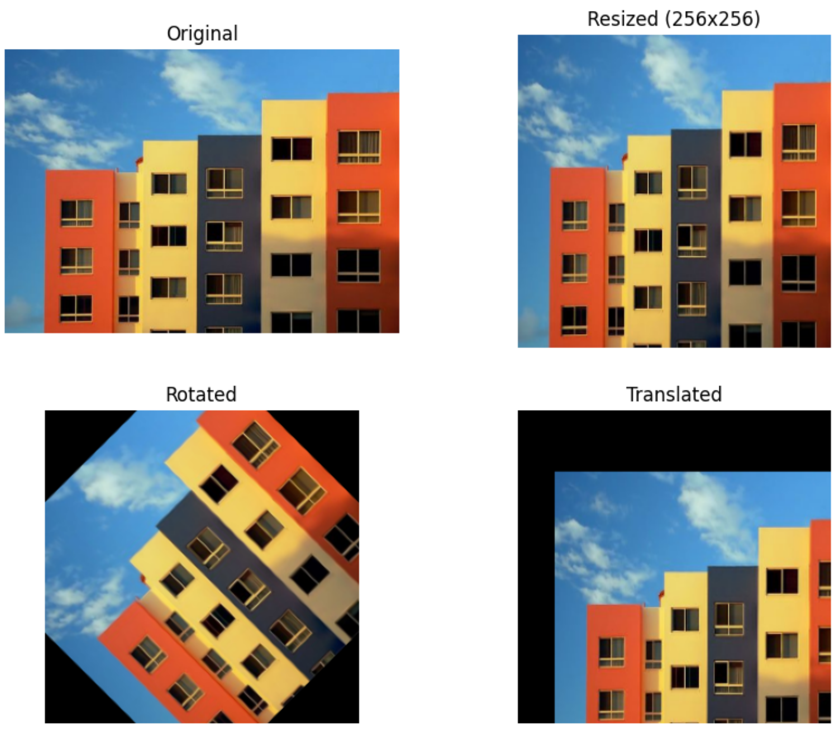


Figure 6: Image transformations: original, resized, rotated, and translated images. Task 3

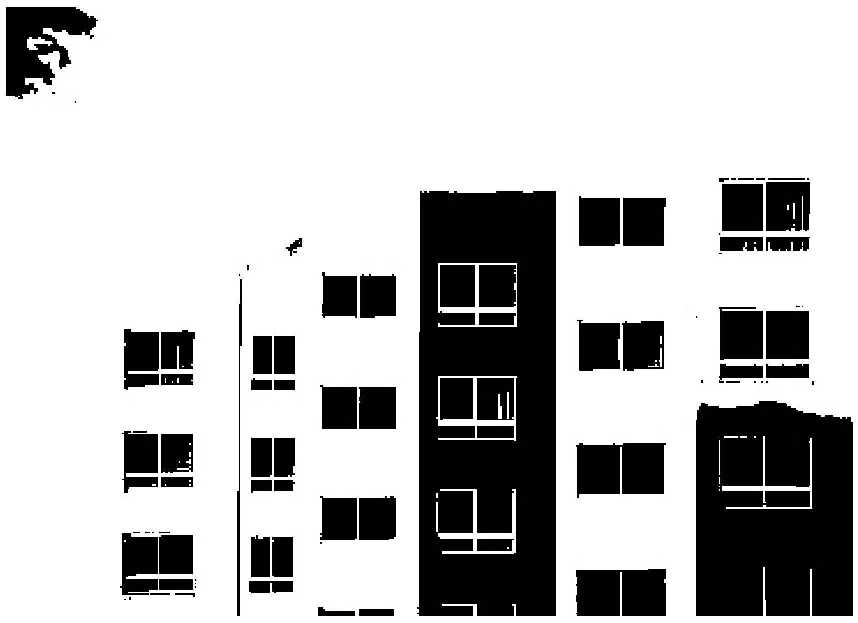


Figure 7: Binary image obtained using thresholding (Otsu's method). Task 6

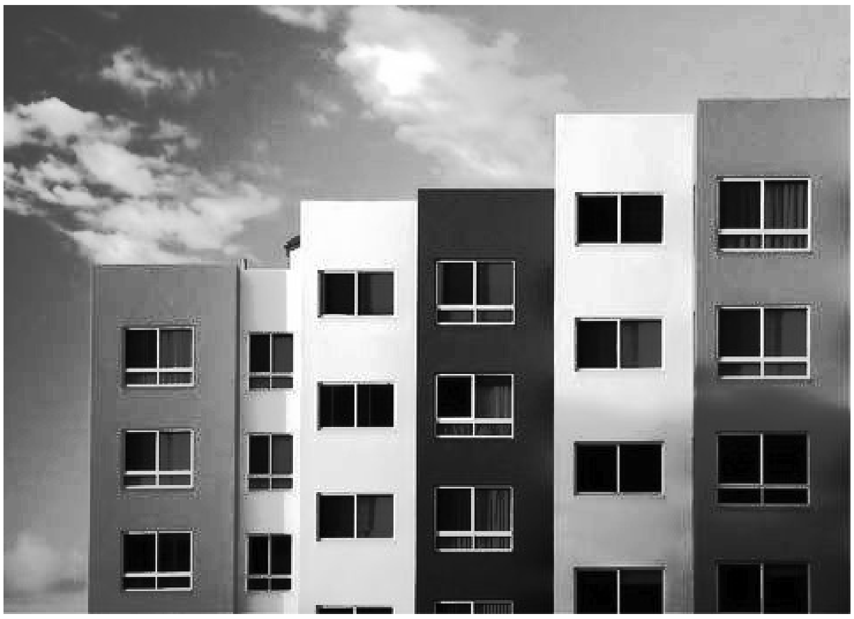


Figure 8: Image after contrast enhancement using histogram equalization. Task 6

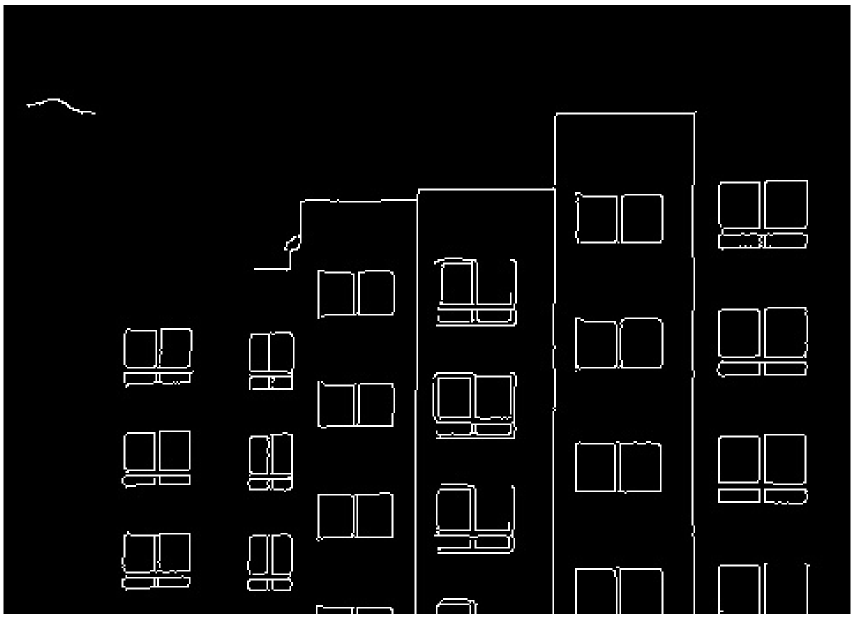
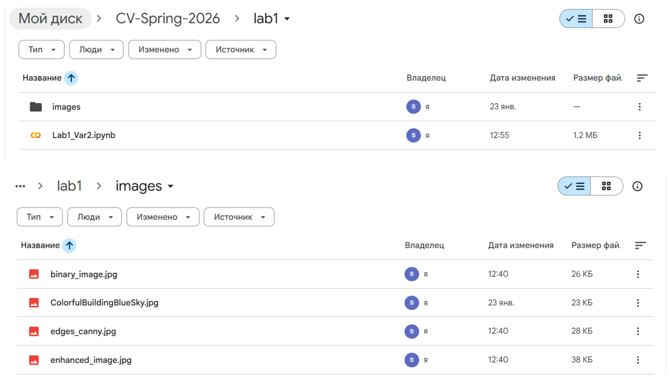


Figure 9: Edges detected using the Canny edge detection algorithm. Task 5



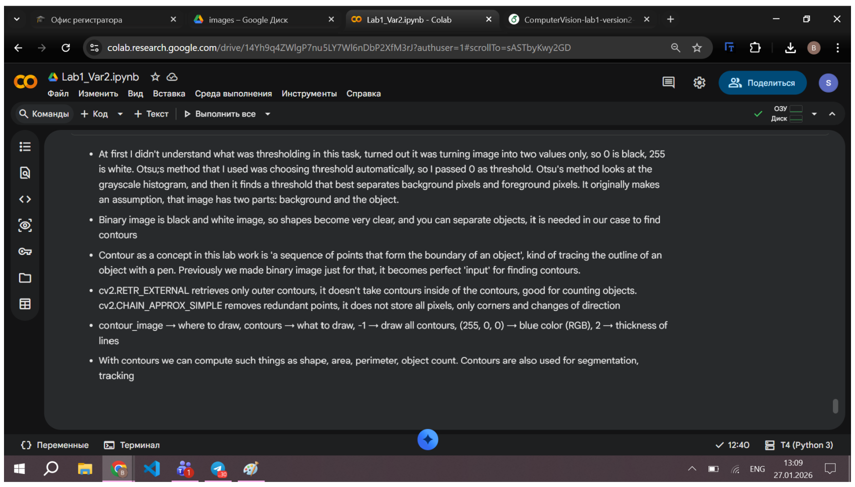
Мой диск > CV-Spring-2026 > lab1

Название	Владелец	Дата изменения	Размер фай.
images	я	23 янв.	—
Lab1_Var2.ipynb	я	12:55	1,2 МБ

… > lab1 > images

Название	Владелец	Дата изменения	Размер фай.
binary_image.jpg	я	12:40	26 КБ
ColorfulBuildingBlueSky.jpg	я	23 янв.	23 КБ
edges_canny.jpg	я	12:40	28 КБ
enhanced_image.jpg	я	12:40	38 КБ

Figure 10: Proof 1. Where my repository and structure, time of the lab work is shown



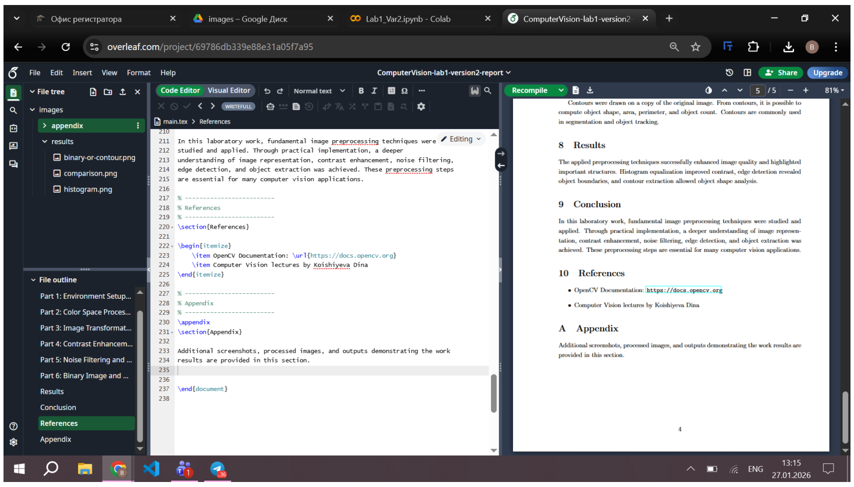
colab.research.google.com/drive/14Yh9q4ZWlgP7nu5LY7Wl6nDbP2XfM3rJ?authuser=1#scrollTo=sASTbyKwy2GD

Lab1_Var2.ipynb

- At first I didn't understand what was thresholding in this task, turned out it was turning image into two values only, so 0 is black, 255 is white. Otsu;s method that I used was choosing threshold automatically, so I passed 0 as threshold. Otsu's method looks at the grayscale histogram, and then it finds a threshold that best separates background pixels and foreground pixels. It originally makes an assumption, that image has two parts: background and the object.
- Binary image is black and white image, so shapes become very clear, and you can separate objects, it is needed in our case to find contours
- Contour as a concept in this lab work is 'a sequence of points that form the boundary of an object', kind of tracing the outline of an object with a pen. Previously we made binary image just for that, it becomes perfect 'input' for finding contours.
- cv2.RETR_EXTERNAL retrieves only outer contours, it doesn't take contours inside of the contours, good for counting objects. cv2.CHAIN_APPROX_SIMPLE removes redundant points, it does not store all pixels, only corners and changes of direction
- contour_image → where to draw, contours → what to draw, -1 → draw all contours, (255, 0, 0) → blue color (RGB), 2 → thickness of lines
- With contours we can compute such things as shape, area, perimeter, object count. Contours are also used for segmentation, tracking

Офис регистратора
images – Google Диск
Lab1_Var2.ipynb - Colab
ComputerVision-lab1-version2
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка
Команды + Код + Текст Выполнить все
Поделиться
Переменные
Терминал
12:40
T4 (Python 3)
ENG
13:09
27.01.2026

Figure 11: Proof 2. Where I wrote every markdown, comment by myself



```latex
In this laboratory work, fundamental image preprocessing techniques were
studied and applied. Through practical implementation, a deeper
understanding of image representation, contrast enhancement, noise filtering,
edge detection, and object extraction was achieved. These preprocessing steps
are essential for many computer vision applications.

% -----------------------
% References
% -----------------------
\section{References}

\begin{itemize}
    \item OpenCV Documentation: \url{https://docs.opencv.org}
    \item Computer Vision lectures by Koishiyeva Dina
\end{itemize}

% -----------------------
% Appendix
% -----------------------
\appendix
\section{Appendix}

Additional screenshots, processed images, and outputs demonstrating the work
results are provided in this section.



\end{document}
```

Contours were drawn on a copy of the original image. From contours, it is possible to compute object shape, area, perimeter, and object count. Contours are commonly used in segmentation and object tracking.

8 Results

The applied preprocessing techniques successfully enhanced image quality and highlighted important structures. Histogram equalization improved contrast, edge detection revealed object boundaries, and contour extraction allowed object shape analysis.

9 Conclusion

In this laboratory work, fundamental image preprocessing techniques were studied and applied. Through practical implementation, a deeper understanding of image representation, contrast enhancement, noise filtering, edge detection, and object extraction was achieved. These preprocessing steps are essential for many computer vision applications.

10 References

- OpenCV Documentation: https://docs.opencv.org
- Computer Vision lectures by Koishiyeva Dina

A Appendix

Additional screenshots, processed images, and outputs demonstrating the work results are provided in this section.

4

Офис регистратора
images – Google Диск
Lab1_Var2.ipynb - Colab
ComputerVision-lab1-version2-
overleaf.com/project/69786db339e88e31a05f7a95
File Edit Insert View Format Help
ComputerVision-lab1-version2-report
File tree
images
appendix
results
binary-or-contour.png
comparison.png
histogram.png
File outline
Part 1: Environment Setup...
Part 2: Color Space Proces...
Part 3: Image Transformat...
Part 4: Contrast Enhancem...
Part 5: Noise Filtering and ...
Part 6: Binary Image and ...
Results
Conclusion
References
Appendix
Code Editor
Visual Editor
Normal text
WRITEFULL
main.tex > References
Editing
Recompile
5 / 5
81%
Share
Upgrade
ENG
13:15
27.01.2026

Figure 12: Proof 3. Where I am writing this LaTeX report by myself