Task: OpenCV Fundamentals - Week 1

INTRODUCTION

OpenCV is one of the most widely used computer vision libraries, offering tools for image processing, video analysis, and machine learning integration. This report documents my Week 1 exploration of OpenCV, focusing on building a practical foundation through Python-based exercises, theoretical notes, and research on real-world applications.

The repository is organized into exercises, notes, and research summaries. The exercises demonstrate fundamental image operations such as reading, displaying, filtering, edge detection, and thresholding. Additional scripts explore OpenCV's handling of GUI windows on different operating systems. The notes provide an overview of image processing concepts and OpenCV modules, while the research highlights applications of OpenCV across diverse domains and discusses differences in OpenCV's behaviour between Linux and Windows environments.

This combination of coding practice, structured notes, and application-focused research establishes a well-rounded understanding of OpenCV fundamentals, setting the stage for more advanced exploration in future weeks.

TOPICS

1. **Introduction to OpenCV:**
   Sources Consulted: OpenCV official site and YouTube tutorials (e.g. CodeBasics, Murtaza's Workshop)
   https://docs.opencv.org/4.x/df/d65/tutorial_table_of_content_introduction.html
   Key Learnings and Insights:
   - OpenCV is an open-source CV library with Python, C++, Java bindings.
   - opencv-python vs opencv-contrib-python (extra modules like SIFT, SURF, etc.).
   - Python uses NumPy arrays for images, enabling easy math operations.
   - Different builds provide or restrict patented features (SIFT/ SURF).

   Conclusion: OpenCV forms the foundation of most CV projects; knowing its modules and installation variants is essential.

2. **The Core Functionality (core module):**
   Sources Consulted: OpenCV Core Tutorials
   https://docs.opencv.org/4.x/d0/de1/group__core.html
   Key Learnings and Insights:
   - Data structures: cv::Mat (C++) ≈ ndarray (Python).
   - Type conversions, arithmetic ops, masks, random number generation.
   - Careful with copy vs reference when manipulating arrays.

- Performance is best when using OpenCV functions rather than plain NumPy.

Conclusion: Core is the "plumbing" - everything else builds on this efficient matrix representation.

3. Image Processing (imgproc module):

Sources Consulted: Imgproc Module Docs
https://docs.opencv.org/4.x/d7/dbd/group__imgproc.html
Key Learnings and Insights:

- Color conversions, smoothing, edge detection, morphological ops.
- Histograms and equalization improve contrast.
- Choice of filter (Gaussian vs median vs bilateral) depends on noise type.
- Morphological operations are powerful for cleaning binary masks.

Conclusion: Understanding imgproc is crucial for preparing images for higher-level tasks (segmentation, detection).

4. Application Utils (highgui, imgcodecs, videoio modules):

Sources Consulted: HighGUI Docs
https://docs.opencv.org/4.x/d7/dfc/group__highgui.html
Key Learnings and Insights:

- Image I/O (imread, imwrite).
- Video handling (VideoCapture, VideoWriter).
- GUI functions like imshow often fail on servers without display.
- Alternatives: save images to file or use matplotlib.

Conclusion: I/O modules are practical utilities bridging algorithms with the real world (files, cameras).

5. Camera Calibration and 3D Reconstruction (calib3d module):

Sources Consulted: Camera Calibration Tutorial
https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
Key Learnings and Insights:

- Calibrates intrinsic & extrinsic parameters using chessboards.
- Undistorts fisheye/barrel distortion.
- Requires multiple images at different angles.
- Precision depends heavily on corner detection accuracy.

Conclusion: Calibration ensures geometric accuracy in robotics, AR, and 3D reconstruction.

6. Object Detection (objdetect module):

Sources Consulted: Cascade Classifier Docs
https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html
Key Learnings and Insights:

- Haar cascades for face/object detection (fast, classical).
- Alternatives: HOG+SVM, or modern DNN detectors.
- Haar cascades are lightweight but error-prone on real-world data.
- DNNs (SSD, YOLO) are more accurate but heavier.

Conclusion: Objdetect is useful for quick/simple tasks; production-grade projects often need DNNs.

7. 2D Features Framework (feature2d module):
Sources Consulted: Feature2D Tutorials
https://docs.opencv.org/4.x/d9/d97/tutorial_table_of_content_features2d.html
Key Learnings and Insights:

- Keypoint detectors: SIFT, SURF, ORB, BRISK.
- Descriptors + matchers (BF, FLANN).
- ORB is free and fast, SIFT is robust but heavier.
- Feature matching enables stitching, recognition, SLAM.

Conclusion: Feature2D powers many classical CV tasks where geometry and matching matter.

8. Deep Neural Networks (dnn module):
Sources Consulted: DNN Docs https://docs.opencv.org/4.x/d6/d0f/group__dnn.html
Key Learnings and Insights:

- Supports inference from Caffe, TensorFlow, ONNX.
- Easy to deploy pre-trained models without full frameworks.
- Best for light inference, not for training.
- Requires downloading weights separately.

Conclusion: cv2.dnn provides a quick way to integrate deep models in OpenCV pipelines.

9. Graph API (gapi module):
Sources Consulted: G-API Overview
https://docs.opencv.org/4.x/df/d7e/tutorial_table_of_content_gapi.html
Key Learnings and Insights:

- Declarative pipeline for CV tasks.
- Optimizes execution via graph compilation.
- Still experimental, limited adoption.
- Useful for embedded/real-time optimization.

Conclusion: Potential future direction - performance-focused workflows for production.

10. Other Tutorials (ml, objdetect, photo, stitching, video):
Sources Consulted: Photo Module and Video Analysis
https://docs.opencv.org/4.x/d3/dd5/tutorial_table_of_content_other.html
Key Learnings and Insights:

- ML: classical classifiers (kNN, SVM).
- Photo: denoising, inpainting.
- Stitching: panoramas.
- Video: background subtraction, optical flow.
- Many tasks now replaced by deep learning but still useful.
- Lightweight methods often perform surprisingly well for small problems.

Conclusion: These modules add specialized but practical utilities for real-world tasks.

11. Theory of Image Processing:

Sources Consulted: Gonzalez & Woods - Digital Image Processing and OpenCV docs on convolution and filters
https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html
Key Learnings and Insights:

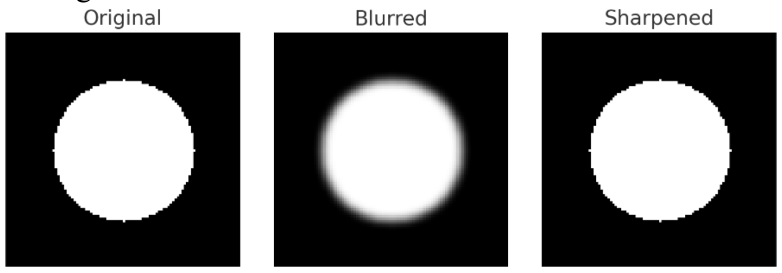
- Pixels, color models (RGB, HSV).
- Convolution and kernels (smoothing, edge detection).
- Frequency domain (Fourier transform).
- Many filters are just kernels applied over neighborhoods.
- Edge detection relates to high-frequency components.

Conclusion: Theory connects math to implementation - helps in designing or tuning algorithms.

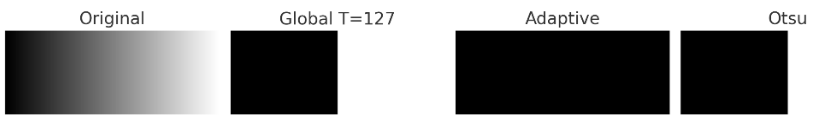
12. Miscellaneous:

- **Understand Image Processing Concepts**:

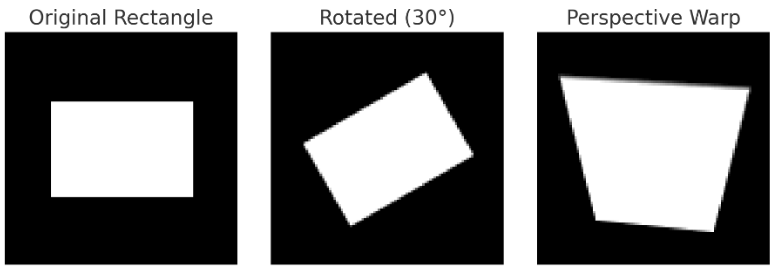
Filtering:



Thresholding:

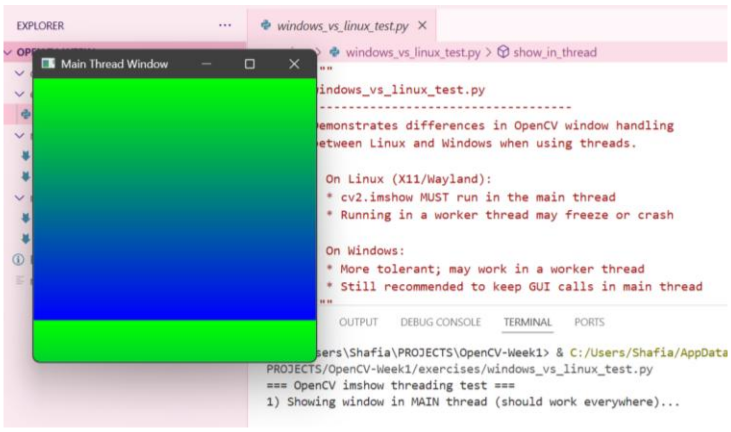


Transformations:



13. Exercises:

- windows_vs_linux_test.py



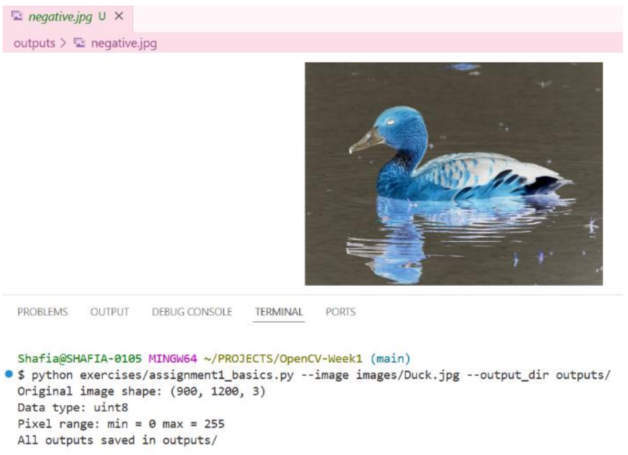
```
"""
indows_vs_linux_test.py
-----------------------------------
emonstrates differences in OpenCV window handling
etween Linux and Windows when using threads.

 On Linux (X11/Wayland):
 * cv2.imshow MUST run in the main thread
 * Running in a worker thread may freeze or crash

 On Windows:
 * More tolerant; may work in a worker thread
 * Still recommended to keep GUI calls in main thread
"""
```

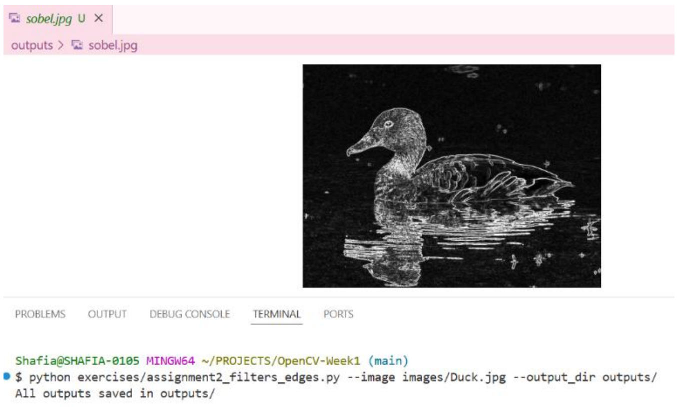
```
ers\Shafia\PROJECTS\OpenCV-Week1> & C:/Users/Shafia/AppData
PROJECTS/OpenCV-Week1/exercises/windows_vs_linux_test.py
=== OpenCV imshow threading test ===
1) Showing window in MAIN thread (should work everywhere)...
```

- assignment1_basics.py



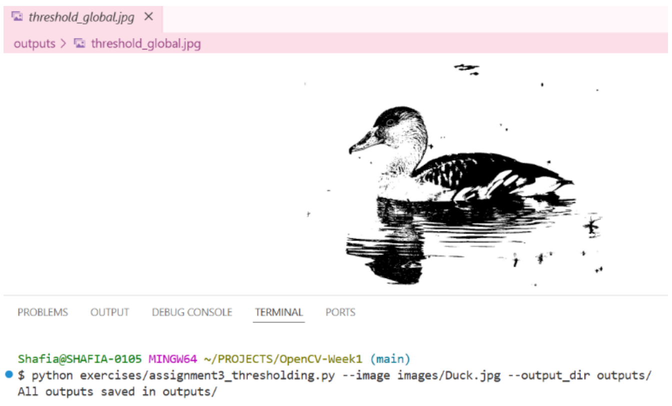
```
Shafia@SHAFIA-0105 MINGW64 ~/PROJECTS/OpenCV-Week1 (main)
$ python exercises/assignment1_basics.py --image images/Duck.jpg --output_dir outputs/
Original image shape: (900, 1200, 3)
Data type: uint8
Pixel range: min = 0 max = 255
All outputs saved in outputs/
```

- assignment2_filters_edges.py



```
Shafia@SHAFIA-0105 MINGW64 ~/PROJECTS/OpenCV-Week1 (main)
$ python exercises/assignment2_filters_edges.py --image images/Duck.jpg --output_dir outputs/
All outputs saved in outputs/
```

- assignment3_thresholding.py



```
Shafia@SHAFIA-0105 MINGW64 ~/PROJECTS/OpenCV-Week1 (main)
$ python exercises/assignment3_thresholding.py --image images/Duck.jpg --output_dir outputs/
All outputs saved in outputs/
```

CONCLUSION

Through this week’s work, I gained hands-on experience with the core functionalities of OpenCV and a deeper appreciation of its role in computer vision. The exercises reinforced practical skills such as image transformations, edge detection, and thresholding, while the notes clarified theoretical concepts like colour spaces, convolution, and module structure. Research into OpenCV’s applications demonstrated its versatility in areas ranging from robotics and medical imaging to surveillance and augmented reality.

A key insight was recognizing how platform-specific differences, such as OpenCV’s window handling on Linux versus Windows, can influence implementation and debugging. Documenting these differences will help in developing more robust and portable code.

Overall, this week laid a strong foundation for working with OpenCV. The organized repository containing code, notes, outputs, and research provide a valuable reference point for continued learning and experimentation with more advanced modules such as object detection, feature extraction, and deep learning integration.