Aim: To perform Handling Files, Cameras and GUIs

Objective: To perform Basic I/O Scripts, Reading/Writing an Image File, Converting Between an Image and raw bytes, Accessing image data with numpy.array,Reading /writing a video file, Capturing camera, Displaying images in a window ,Displaying camera frames in a window

Theory:

Basic I/O Script:

OpenCV, an open-source computer vision library, provides a wide range of functions for handling files, cameras, and graphical user interfaces (GUIs). In this experiment, we will explore some fundamental concepts and operations in computer vision using OpenCV.

Reading/Writing an Image File:

One of the primary tasks in computer vision is loading and saving images. OpenCV allows us to read images from various formats, such as JPEG, PNG, or BMP, using the `cv2.imread()` function. Conversely, we can save images using the `cv2.imwrite()` function. These functions are essential for input and output operations in image processing tasks.

Converting Between an Image and Raw Bytes:

Images can be represented as raw binary data, and OpenCV provides methods to convert between image data and raw bytes. This is useful for tasks like transmitting images over networks or storing them in databases. The `cv2.imencode()` and `cv2.imdecode()` functions facilitate this conversion.

Accessing Image Data with numpy.array:

OpenCV stores images as numpy arrays. This allows us to access and manipulate individual pixel values efficiently. Understanding how to work with numpy arrays is crucial for various image processing tasks, such as filtering, transformations, and feature extraction.

Reading/Writing a Video File:

Video processing is an essential aspect of computer vision. OpenCV provides functionalities to read and write video files. We can use the `cv2.VideoCapture()` class to read video frames and the `cv2.VideoWriter()` class to create video files. This is essential for applications like video analysis, surveillance, and object tracking.

Capturing Camera Frames:

Accessing live camera feeds is a common requirement in computer vision applications. OpenCV allows us to capture frames from a connected camera using the `cv2.VideoCapture()` class. Understanding how to interface with cameras is crucial for real-time image processing and computer vision systems.

Displaying Images in a Window:

Visualizing images is essential for debugging and understanding the results of image processing operations. OpenCV provides functions like `cv2.imshow()` to

display images in a window. Proper visualization aids in the interpretation and analysis of computer vision algorithms.

Displaying Camera Frames in a Window:

Displaying live camera frames in a window is valuable when working with camera inputs. By continuously capturing and displaying frames from a camera source, we can create interactive computer vision applications, such as motion detection, face recognition, or augmented reality.

Conclusion:

In this experiment, we explored fundamental operations related to handling files, cameras, and GUIs in computer vision using OpenCV. We learned how to read and write image files, convert images to raw bytes, access image data using numpy arrays, work with video files, capture camera frames, and display images and camera frames in graphical windows. These skills form the foundation for more advanced computer vision tasks and applications.