EECS 4314 - Advanced Software Engineering

OpenCV Conceptual Architecture

Eric Rodrigues, Laura Marin, Alp Baran Sirek, Negar Khalilazar, Danny Le

ericr100@my.yorku.ca, nemira@my.yorku.ca, hiangel@my.yorku.ca, nkhazar@my.yorku.ca,

dannyle3@my.yorku.ca

Dr. Zhen Ming (Jack) Jiang

Monday, October 6, 2025

Table of Contents

Abstract

The purpose of this report is to present the conceptual architecture of OpenCV, an open-source library. It will provide newcomers with a high-level understanding of how the system is organized and how its parts interact. This report will focus on architectural abstracts in order to illustrate the system's structure, design, flow of control, and data between its subsystems. The report presents OpenCV in a broader context to show how it fits into the world of computer vision, and how it interacts with external systems to give desired results to the user. It introduces the library's modular architecture, breaking it down into its components, and analyzing how they collaborate. Attention will be given to control and data flow, concurrency, and performance considerations that influence how the system operates. The report also examines OpenCV's evolution, particularly how its architecture has adapted to new technologies. To further explain the architecture, this report includes use cases, such as image filtering and face detection in video streams. In addition, key concepts will be defined as well as terminology to help newcomers build their understanding of the software. The report will conclude with a discussion of the implications of developer collaboration via open source and directions for future work. By the end of this report, readers should have a clear understanding of OpenCV's conceptual architecture, enabling them to more effectively explore its capabilities and participate in its continued development.

Introduction and Overview

Computer vision is a rapidly evolving field within software development built on a vast collection of algorithms and modules that provide foundational support for image processing, machine learning, and real-time video analysis. Although this range of features has made computer vision essential; however, it also introduces complexity for individuals new to the software. At its core, computer vision is a tool that enables computers to interpret and process visual information from the world. When provided with digital images and video streams, it's able to analyze and understand these resources to extract meaningful information.
Applications of computer vision appear in daily life, often unnoticed. Cars use it to track objects in real time, medical image systems use it to identify abnormalities, and it's a huge part of augmented reality tools that blend digital and physical worlds. Those examples illustrate only a fraction of computer vision's capabilities. Although the potential of computer vision is exciting, it also, as stated before, presents several challenges. Processing visual data in real time requires significant computational power and efficient algorithms. The tasks within its domain are highly diverse, from low-level image filtering to high-level activities such as facial recognition and semantic segmentation. Furthermore, computer vision systems must operate across a variety of platforms, from powerful GPUs in data centers to constrained embedded devices in mobile environments. These demands stress the need for robust, flexible, and optimized software libraries.
OpenCV, or the Open Source Computer Vision Library, is one of the most widely used tools for addressing these challenges. It is an open-source library designed for real-time image processing and computer vision applications. Intel Research initially developed it in 1999 and released it publicly in 2000. Over the years, it has been supported and extended by many developers, and today it is maintained by OpenCV.org. Initially, early versions focused mainly on providing efficient implementations of computer vision algorithms in C and C++ with a strong emphasis on real-time performance. The library, however, has expanded significantly now supporting additional languages such as Java, Python and MATLAB and is also able to be used on mobile platforms, including Android and iOS. More recently, OpenCV has integrated with popular machine learning and deep learning frameworks such as TensorFlow, PyTorch, and ONNX, enabling developers to combine traditional vision techniques with modern AI models. Recently, a key milestone in OpenCV's evolution has been the inclusion of GPU acceleration through CUDA and OpenCL. Before image

processing in OpenCV was executed on the CPU, complex operations were computationally slow. By introducing GPU acceleration, OpenCV can now offload many of these operations to graphics processing units. Unlike CPUs, GPUs consist of thousands of smaller cores optimized for performing many operations in parallel. This parallelism makes them ideal for tasks where the same computation needs to be applied repeatedly across large datasets, such as applying filters to every pixel in an image or running deep learning inference across batches of images. This, along with many other features, has been implemented into OpenCV to cover a wide range of tasks, helping it continue to be one of the most widely used computer vision frameworks in both academics and industry.

Architecture

OpenCV Main Modules:

- **Core functionality**
  - Describes basic data structures.
  - Includes multi-dimensional array cv::Mat and basic functions used by all other modules.
- **Image Processing**
  - This method includes image-processing functions
  - Linear and non-linear image filtering, image transformations (affine transformation, perspective wrapping, resizing, etc.)
- **Image file reading and writing**
  - Read and write image files in many formats
  - Loads and saves JPEG, PNG, etc.
- **Video I/O**
  - Read and write video or images sequence
  - Captures from the camera (video capturing) and video codecs
- **High-level GUI**
  - Designed for simple GUI and display/interaction. (UI capabilities)
  - It provides simple interface to:
    - Create and manipulate windows that can display images without the need to handle repaint events. (window management)
    - Add trackbars to the windows, handle simple mouse events as well as keyboard commands. (user interactions for debugging)
- **Video Analysis**
  - Motion estimation, background subtraction, and object tracking algorithms.
- **3D Reconstruction**

    - Basic 3D geometry (multiple-view algorithms and elements of 3D reconstruction), stereo vision, camera calibration, and pose estimation.
- **2D Features Framework**
    - Keypoints detection, computes and matches descriptors, and finds features correspondence.
- **Object Detection**
    - Classifier for object detection
    - Detection routines for predefined types/classes/objects (ex. Faces, eyes, people, cars, etc…)
- **Deep Neural Network module**
    - API for new layers creation, layers are building bricks of neural networks;
    - A set of built-in most-useful Layers;
    - API to construct and modify comprehensive neural networks from layers;
    - functionality for loading serialized networks models from different frameworks.
    - Functionality of this module is designed only for forward pass computations (i.e. network testing). Network training is in principle not supported.
- **Machine Learning**
    - A set of classes and functions for statistical classification, regression, and clustering of data. (Generic machine learning algorithms)
- **G-API**
    - Lets users express computations as graphs, essentially expresses computer vision pipelines at a higher level of abstraction.
- **Computational Photography**
    - This module includes photo processing algorithms
    - Higher level image manipulation (inpainting, denoising,advanced processing, image restoration, etc…)
- **Images stitching**
    - Camera modules with image stitching
    - Aligning images, blending, wrapping, composing panoramas, etc…

OpenCV Extra Modules:

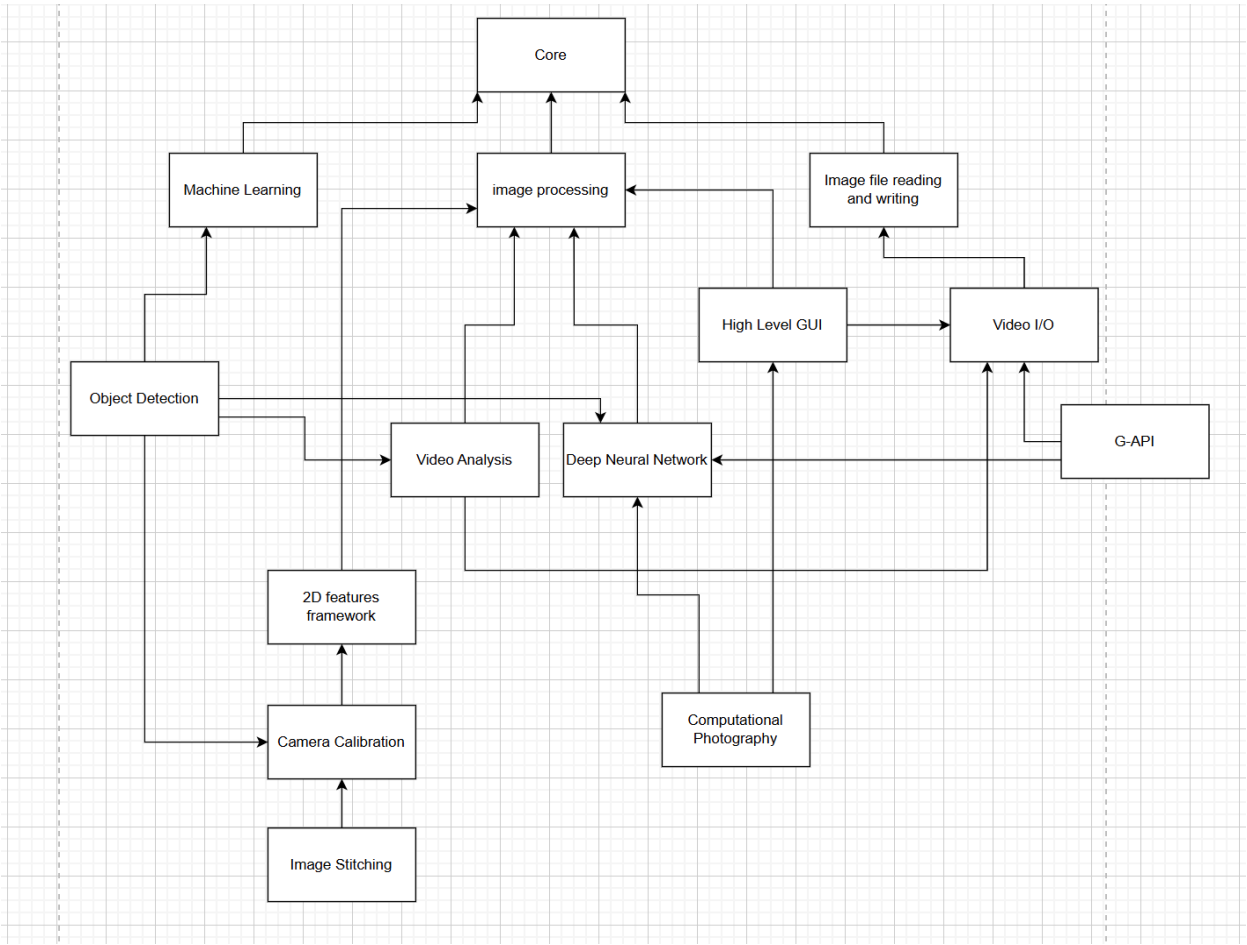
- **Clustering and Search in Multi-Dimensional Spaces**
    - Documents OpenCV's interface to the FLANN library.
    - FLANN (Fast Library for Approximate Nearest Neighbors) is a library that contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features
- **Operations on Matrices**
    - This module uses CUDA-acceleration to provide GPU-based mathematical operations on matrices and images, but requires CUDA-capable hardware.
- **Background Segmentation**

    - This module provides CUDA-accelerated background/foreground segmentation algorithms for video processing on the GPU, but requires CUDA-capable hardware.
- **Video Encoding/Decoding**
    - This module provides CUDA-accelerated video encoding and decoding capabilities to be done on the GPU, but requires CUDA-capable hardware.
- **Feature Detection and Description**
    -
- **Image Filtering**
    - Functions and classes described in this section are used to perform various linear or non-linear filtering operations on 2D images.
- **Legacy support**
    - The obsolete CUDA-accelerated implementations of functions and classes are moved into this module, for the purpose of backward compatibility
- **Object Detection**
    - This is the same as the standard objdetect module, but this version runs on the GPU using NVIDIA CUDA for hardware acceleration. This offers better performance for real-time detection, but requires CUDA-capable hardware.
- **Optical Flow**
    - This module uses CUDA-acceleration to provide GPU implementations of various optical flow algorithms, but requires CUDA-capable hardware.
- **Stereo Correspondence**
    - Contains CUDA-accelerated classes and functions to provide GPU-based stereo vision algorithms for computing disparity maps from stereo image pairs. but requires CUDA-capable hardware.
- **Image Warping**
    - Contains CUDA-accelerated functions that provide image warping operations that are performed on the GPU, but requires CUDA-capable hardware.
- **Device layer**
    - Contains device-level utilities for CUDA programming, such as type traits, vector math operations, and compatibility layers. This enables developers to write custom CUDA kernels for OpenCV.
- **Shape Distance and Matching**
    - Contains classes and functions for shape analysis, comparison, and alignment.
- **Super Resolution**
    - The Super Resolution module contains a set of functions and classes that can be used to solve the problem of resolution enhancement.
- **Video Stabilization**
    - Contains a set of functions and classes that can be used to solve the problem of video stabilization.
- **3D Visualizer**
    - Contains classes and functions that can be used to display and interact with 3D objects, point clouds, camera poses, and coordinate frames.

Interactions Among Major Components:

- Nearly everything is dependent on the Core functionality.
    - Does not depend on any higher level modules
- Image processing is used by modules that do core image manipulation tasks; feature extraction, detection, etc.
    - High-level GUI
    - Video Analysis
    - Camera Calibration and 3D construction
    - 2D Feature Framework
    - Object Detection
    - Computational Photography
    - Image Stitching
- Image file reading and writing is used by any module that needs to load images
    - High level GUI
    - Video I/O
    - Image processing
- Video I/O is used by video analysis modules
    - Video Analysis
    - High level GUI
- Video Analysis
    - Object detection
    - Camera Calibration
    - 2D features framework
- 2D Features framework is used by higher level detection, stitching, and 3D modules.
    - Object detection
    - Camera calibration
    - Image Stitching
- Deep Neural Network
    - Object detection
- Machine Learning
    - Object detection
    - 2D features framework
- Camera Calibration
    - Image Stritching
- Object detection
    - Video analysis
    - 2D features framework
    - Image processing
    - DNN
    - Machine learning
- High level GUI
    - Used by all modules during testing

Dependency Diagram:



Performance Critical Parts:

- Core functionality
  - This is the base layer for all operation
- Image Processing
  - Real-time applications depend on fast image processing
- Video I/O
  - Reads and writes video frame in real time, if it isn't fast frames may be dropped and latency increased
- 2D Features Framework
  - It is computational heavy to do feature matching and detection operations
- Object Detection
  - Detection usually has to run at frame rate
- DNN
  - Most expensive stage in the modern CV pipelines, it is computationally heavy and needs CPU and or GPU optimization
- Camera Calibration and 3D Reconstruction
  - Geometry heavy, most robotics systems need fast and accurate geometry
- Image Stitching

- Combines multiple high resolution images and the performance affects responsiveness in panoramic applications. It needs to be able to handle large image sizes quickly.
- Can also combine images under affine transformations, such as scans of documents.

How does it support future changes to the system?
OpenCV follows a modular structure but was designed with scalability in mind. Each functional module is implemented independent of one another, so that new features may be added without impacting unrelated modules. For example, the DNN module was added recently (version 3.3 released 2017), but its implementation did not require changes to the Core or Image Processing modules. OpenCV also utilises layered abstraction to build higher level functionality on top of lower level layers. This structure allows for improvements to the lower layer to enhance performance in the higher layers without breaking them.

Control Flow

The general control flow of OpenCV starts with the User application, then High-level API, functional modules such as imgproc, video or DNN, core, hardware and backends then it's sent back to the user application. The user application owns the main loop. Functional modules contain domain packages such as imgproc, features2d, calib3d, video, photo and DNN. G-API recasts Open CV's call return pipeline into a declarative graph. The user application still holds external control. [9]

Data Flow

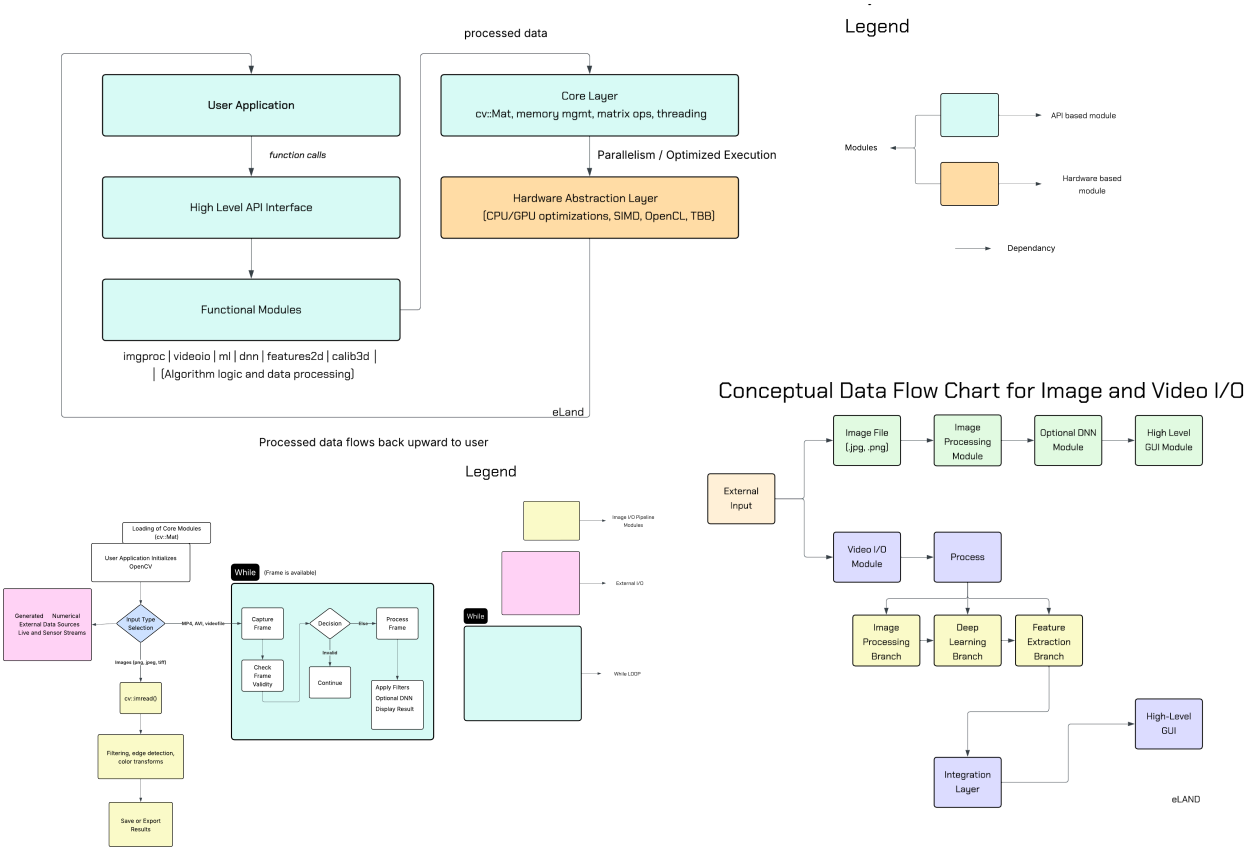
Data on OpenCV is mainly carried as Mat objects/containers for images or frames where the containers are treated as multidimensional arrays. The user application creates, with called methods, a linear data path. The application requests an image file which the library's decoding layer translates the image into a Mat carrier.[12] Then the Mat carrier is forwarded to an image processing module such as filtering, color conversion. Each stage takes arrays and produces new arrays of Mat carriers. When the pipeline creates the final multidimensional array of Mat carriers after every operation that was needed to be done, the application then passes the resulting Mat to the High Level GUI. It is important to note that the application owns the process. The long running pipeline process loop is how videos are processed.

Concurrency

OpenCV integrates concurrency at multiple layers of its architecture where the library's functions from different class instances can be safely executed from multiple threads simultaneously. [10] "The same MAT can be used in different threads because the reference-counting operations use the architecture-specific atomic instructions" [10]. This reference explains cv:Mat objects are thread safe at the memory management level. OpenCV

also uses the “parallel_for_” library in order to achieve optimized computation on sequential implementation [11]. The actual backend is selected at build time with the parallel_for calls.

Beyond CPU thread level parallelism, OpenCV also supports GPU’s and other accelerators such as T-API where the object used for processing is changed from Mat to UMat. If passed as UMat, when OpenCL is available, OpenCV can route the calls for a different execution path, which is more efficient. For NVIDIA GPUs, the CUDA modules are used in which asynchronous execution via CUDA streams are enabled.[12]



System Evolution

Since its initial release, OpenCV has undergone continuous evolution to accommodate hardware capabilities, programming languages, and computer vision paradigms. OpenCV’s first release was primarily optimized for CPU-based image processing and low level operations such as filtering, edge detection, and feature extraction. Over time, it has expanded into a comprehensive set of libraries encompassing many areas like object recognition, motion tracking, 3D reconstruction, and deep learning.

A major milestone in OpenCV’s evolution came with the introduction of GPU acceleration. This began around version 2.0, when the CUDA module was introduced to leverage NVIDIA GPUs for parallel computation.The CUDA module allowed OpenCV to

offload heavy computations like matrix transformations or image filtering to NVIDIA GPUs. This made processing 10x faster in many real-time applications. Later, OpenCV integrated OpenCL support through the Transparent API, this meant that OpenCV could automatically dispatch operations to available hardware, whether it was a CPU or GPU, without requiring developers to rewrite their code. In the following years OpenCV added bindings for multiple programming languages, including Python, Java, and JavaScript. The python binding in particular was huge due its growing popularity in data science and AI, positioning OpenCV as an accessible and versatile tool for both academic and industrial applications.

Another defining step was in the release of OpenCV 3.3 in 2017, which added the Deep Neural Network (DNN) module. This module enabled users to import and run pre-trained neural networks from deep learning frameworks like TensorFlow, Caffe, and Torch directly within OpenCV. Deep learning–based methods were far more accurate than the old object detection programs used before. Importantly, this addition did not disrupt or modify the pre-existing modules, preserving backward compatibility and ensuring that traditional computer vision workflows could continue to operate alongside modern deep learning pipelines. In recent years, OpenCV has significantly expanded its role beyond traditional computer vision by incorporating support for artificial intelligence and deep learning workflows. In an interesting reversal of roles, OpenCV now plays a key role in AI training, offering powerful tools to prepare and transform image data for deep learning models. By uniting classical algorithms with deep learning capabilities, OpenCV empowers developers to build hybrid solutions that combine the speed of optimized vision pipelines with the accuracy of modern neural networks.

Division of Responsibilities Among Participating Developers

OpenCV operates as a modular open-source project in which responsibilities are presumed mainly divided among users, contributors, maintainers, release engineers, and core leadership to ensure continued quality and stability. [14]

1) Users:

Most OpenCV participants interact as users as they file issues, propose bug reproductions, and discuss problems in forums. User feedback acts as the entry point for new fixes and feature proposals, which serves as essential input for future development cycles. The official contribution guide directs newcomers to appropriate forums, resources, and the correct repositories for opening tickets or pull requests. [14]

2) Contributors:

Anyone with a GitHub account can contribute to OpenCV through the project's public repository. Contributors must adhere to established code style, provide documentation, and ensure testing beforehand. [13] The OpenCV process specifies which branches should be targeted for different kinds of changes, which helps preserve modularity among systems. [14]

3) Reviewers / Maintainers:

Maintainers and reviewers, who are specialists in particular modules, handle the technical code reviews and merges. Proposed changes run through automated CI (i.e. GitHub Actions, Buildbot), after which maintainers evaluate coding style, design consistency, and functional integrity. Finally, these merges require explicit acceptance by maintainers. [14]

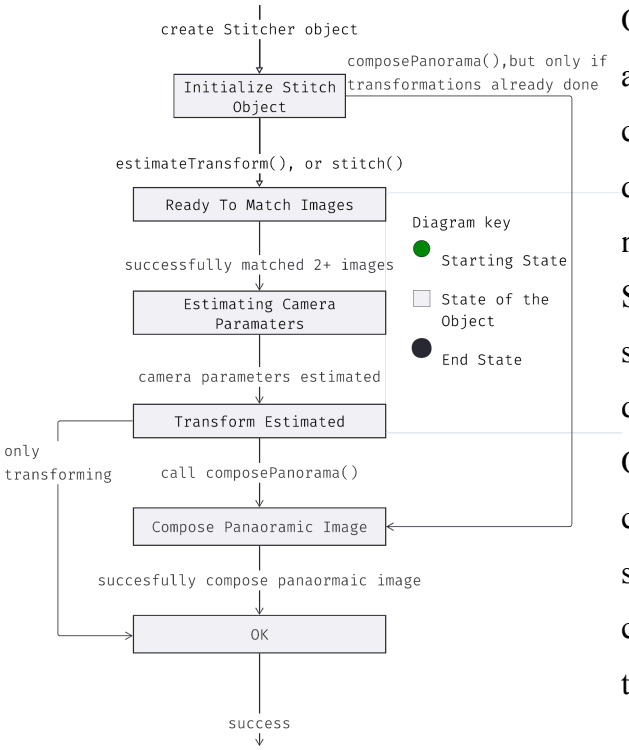
4) Release / CI Engineers:

Release and CI engineers maintain the build pipeline, ensuring that OpenCV runs correctly across operating systems and platforms. All merges are supported by full cross-platform testing, documentation checks, and quality assurance that catches issues early in the development cycle. These requirements are enforced by this team, which means that code is only published once its compliance is checked by them. [14]

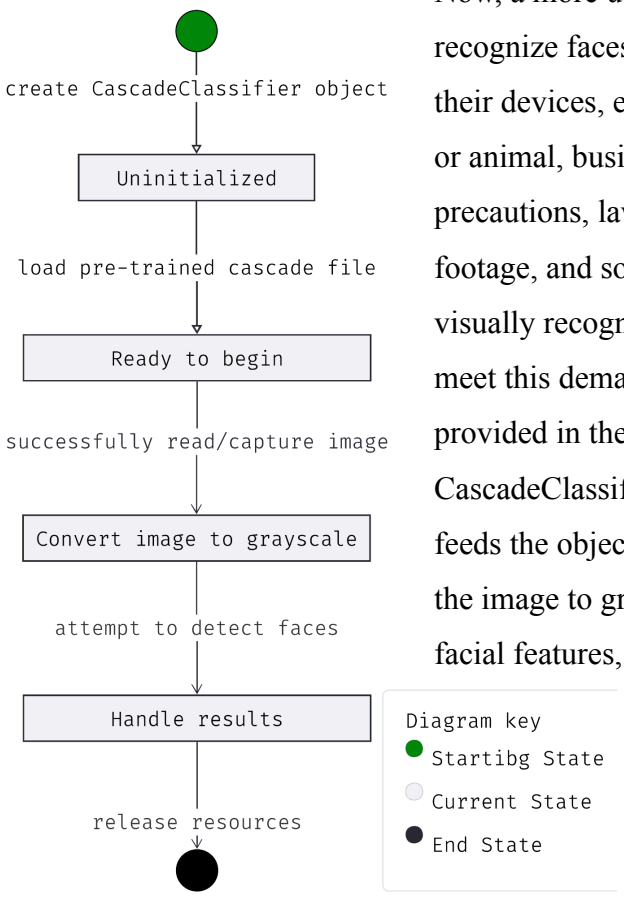
5) Core Leadership / Technical Governance:

A central technical leadership team maintains API stability and long-term project direction. Technical leaders enforce branch policy and manage changing across release lines, while maintaining optimality. [14]

Use Cases



OpenCV’s library contains hundreds of computer vision algorithms, and thus supports a variety of cases where a computer must efficiently process and analyze visual data. One such potential case is the upcoming Moon rover that “MDA Space” is developing for Canada. Such vehicles require the operator to have full situational awareness, as mistakes can cost millions of dollars, years of production, and loss of human life. One solution is to mount multiple cameras, but this creates multiple feeds to review. OpenCV provides a superior solution with the core module’s “Stitcher” class, which can take multiple images or camera feeds to create one panoramic image or video.



Now, a more ubiquitous case would be the ability for a computer to recognize faces. The consumers' desire a quick method to unlock their devices, engineers need robots capable of recognizing a person or animal, businesses' desire automated security and automatic safety precautions, law enforcement needs to quickly identify someone with footage, and so much more. The demand for a computer capable of visually recognizing human faces is omnipresent. And OpenCV can meet this demand with its "CascadeClassifier" class, which is provided in the core module. In essence: the user instantiates a CascadeClassifier object, loads the appropriate pre-trained model, and feeds the object with images or video. The object will then convert the image to grayscale for better efficiency with its algorithms, detect facial features, and output the results with colour.

External Interfaces

In this report, our system of study focus is going to be G-API among OpenCV's external interfaces. G-API provides a graph based execution model that allows image processing and CV pipelines to be expressed as computational graphs. [18]

Data Dictionary

Core Data Structures

- cvMat - Multi-dimensional array container class that serves as the fundamental data structure for storing images, frames, and matrices in OpenCV
- UMat - Unified Mat; alternative to Mat that enables automatic GPU acceleration through OpenCL when available

Architectural Components

- Core Module - Foundation layer providing basic data structures and functions used by all other OpenCV modules
- DNN Module - Deep Neural Network module added in v3.3 (2017) for importing and running pre-trained neural networks from frameworks like TensorFlow and Caffe
- CUDA Module - GPU acceleration module for NVIDIA GPUs, enabling parallel computation for performance-critical operations
- OpenCL - Open Computing Language; hardware acceleration API supporting various GPU vendors through the Transparent API (T-API)
- G-API - Graph API module that allows expressing computer vision pipelines as computational graphs at a higher abstraction level

Key Classes/Functions

- CascadeClassifier - Class for object detection, particularly used for face detection using pre-trained Haar cascades
- Stitcher - Class for combining multiple images or video feeds into panoramic views
- parallel_for_ - Library function for optimized parallel computation on CPUs with configurable backend

External Frameworks

- FLANN - Fast Library for Approximate Nearest Neighbors; external library for efficient nearest neighbor search in high-dimensional spaces
- ONNX - Open Neural Network Exchange; format for representing deep learning models

Development Roles

- Maintainers - Module specialists who handle technical code reviews and merges
- Contributors - GitHub users who submit code changes following OpenCV coding standards
- Release Engineers - Team responsible for CI/CD pipeline and cross-platform testing

Conclusion

This report's high-level analysis of OpenCV has established the library's layered architectural style, such that low-level modules depend on the basic functionality provided by the higher-level modules. Through illustrating the long history of OpenCV, this report has demonstrated how OpenCV has grown more capable by adapting to new technologies, such as GPU acceleration and machine learning. This report has demonstrated that there are a wide range of cases where a computer requires the ability to efficiently process and analyze visual data in real time, and that OpenCV can effectively satisfy this demand through its library of over 2500 algorithms. In conclusion, this report has demonstrated that OpenCV is a robust framework for handling computer vision related tasks, which is the result of the combined effort of its countless users, contributors, reviewers, release managers, and core leadership.

Lessons Learned

The first lesson that we learned was that even though OpenCV is a popular open-source library, some of its modules, classes, and functions are lacking any documentation. Additionally, even the more well-documented modules could be difficult to fully understand, especially in regards to the finer details of the implementation of each class and function. This further compounds onto our next learned lesson, that being how we must begin earlier in the division and execution of our tasks. Given that we are developing our understanding of OpenCV as this project progresses, it can be difficult to estimate the time required to carry out each portion of the project. We have also learned the value of documenting all of our findings and our thought process, both to review our own thoughts and so that we may more efficiently share our findings amongst each other. Finally, we must also attempt to meet more often as a whole team, whether this be through a voice call or in-person meeting. We have found that both methods of team meetings to be a more effective means of communication than messaging each other.

References

[1] “OpenCV Applications in 2023.” *OpenCV*, 2023, https://opencv.org/blog/opencv-applications-in-2023/.

[2] “Introduction to OpenCV.” *OpenCV Documentation*, version 4.x, https://docs.opencv.org/4.x/d1/dfb/intro.html.

[3] “FAQ.” *OpenCV Wiki*, GitHub, https://github.com/opencv/opencv/wiki/FAQ

DNN

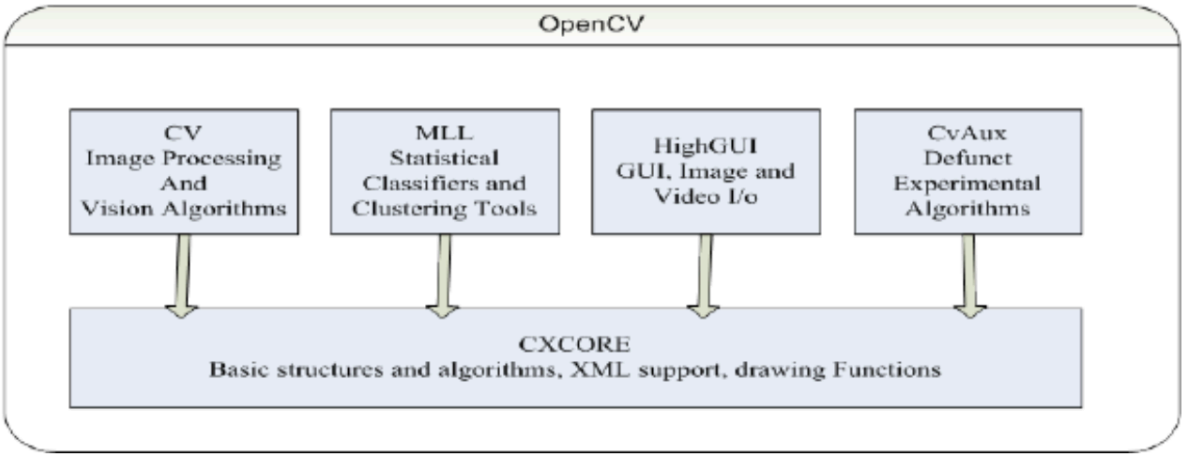
Q: *Cannot read frozen object detection* `.pb` *with OpenCV DNN API after it was trained with TF Object Detection API. What is the reason?*

A: TF Object Detection API provides opportunities for detection model training. There are scripts for TF2, TF1.5 training and further model export, namely `model_main_tf2.py` and `exporter_main_v2.py`, `model_main.py` and `export_inference_graph.py`. After `exporter_main_v2.py` execution the model, checkpoints and variables will be saved in the specified directory. Observing this model after freezing, you will find `StatefulPartitionedCall/...` nodes. It indicates TF Eager Mode, which is not supported in OpenCV.
One of the options is using TF1.5 scripts: `model_main.py` and `export_inference_graph.py`.
Discussion: #19257

Q: *Is there control flow support in OpenCV?*

A: OpenCV doesn't support control flow and currently, there is no plan for the near future to implement it.
Error example: `onnx_graph_simplifier.cpp:592: error: (-210:Unsupported format or combination of formats) Unsupported data type: BOOL in function 'getMatFromTensor'`
The problem appeared during `.onnx` feeding into `cv2.dnn.readNetFromONNX(...)`.
Discussion: #19366, #19977

[4] “Flow of Data Processing in the Design.” *ResearchGate*, https://www.researchgate.net/figure/Flow-of-data-processing-in-the-design-Processing-in-both-the-cores-simultaneously-can_fig7_252973402.



The basic structure of OpenCV .

[5] *OpenCV Reference Manual.* University of North Carolina, https://www.cs.unc.edu/Research/stc/FAQs/OpenCV/OpenCVReferenceManual.pdf.

[6] Ravi, Teja. “Introduction to OpenCV: A Comprehensive Overview.” *Medium*, 2023, https://medium.com/@teja.ravi474/introduction-to-opencv-a-comprehensive-overview-c847dfa1c2fa.

[7] “CUDA Module.” *OpenCV*, https://opencv.org/platforms/cuda/.

[8] Bradski, Gary, et al. *OpenCV Library. arXiv*, 2010, https://arxiv.org/abs/1005.2581.

[9] “HighGUI Module.” *OpenCV Documentation*, version 4.x, https://docs.opencv.org/4.x/d7/dfc/group__highgui.html.

[10] “Introduction to OpenCV.” *OpenCV Documentation*, version 4.x, https://docs.opencv.org/4.x/d1/dfb/intro.html.

[11] “Using Parallel For in OpenCV.” *OpenCV Documentation*, version 4.x, https://docs.opencv.org/4.x/d7/dff/tutorial_how_to_use_OpenCV_parallel_for_.html.

[12] “Coding Style Guide.” *OpenCV Wiki*, GitHub, https://github.com/opencv/opencv/wiki/coding_style_guide.

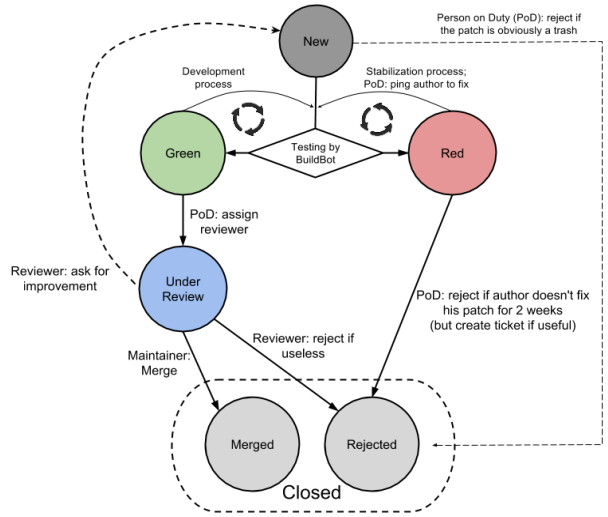
[13] “How to Contribute.” *OpenCV Wiki*, GitHub, https://github.com/opencv/opencv/wiki/how_to_contribut.

[14] “Introduction to Core Module.” *OpenCV Documentation*, *Fossies*, https://fossies.org/linux/opencv/modules/core/doc/intro.markdown.

[15] “OpenCV 3.3 Release.” *OpenCV Blog*, 2017, https://opencv.org/blog/opencv-3-3/.

[16] “Supported Platforms.” *OpenCV*, https://opencv.org/platforms/.

[17] “What Is OpenCV?” *Roboflow Blog*, https://blog.roboflow.com/what-is-opencv/#:~:text=OpenCV's%20architecture%20is%20designed%20to,minimizes%20redundancy%20and%20boosts%20efficiency.



[18]”Graph API” https://docs.opencv.org/4.x/d0/d1e/gapi.html