Conceptual Architecture of OpenCV

Authors

Gabriel Rincon - 21879132 - gabrielenrrincon@gmail.com
Arash Saffari - 218791632 - arashrt@my.yorku.ca
Nargis Rafie - 220785903 - nargis.rafie00@gmail.com
Joshua Zuker - 218135574 - joshz@my.yorku.ca
Leonard Schickedanz - 222634661 - leon.schickedanz@gmail.com
Shaheer Lone - 217277807 - shaheerlone@gmail.com

Abstract

This report examines the conceptual architecture of OpenCV, an open-source library used for computer vision and machine learning applications that involve analyzing 2D and 3D features in videos and images. It provides an overview of the system's overall architectural style, key components, and their interactions, highlighting how modularity supports maintainability, testability, and future evolution. The most important modules, such as Core, Image Processing, Image I/O, Video I/O, High-level GUI, Video Analysis, Camera Calibration and 3D Reconstruction, 2D Features Framework, Object Detection, Deep Neural Network module, Machine Learning, Clustering and Search in Multi-Dimensional Spaces, Computational Photography, Image Stitching, and Graph API are described in terms of functionality, data flow, and their dependencies. The report also discusses the Hardware Abstraction Layer, which optimizes performance across many different platforms. Furthermore, concurrency and parallelism functionalities, including TBB, OpenMP, and GPU acceleration through CUDA and OpenCL, are described. OpenCV's architecture emphasizes robustness, extensibility, and efficiency, enabling it to remain a widely trusted and adaptable tool for computer vision applications. The system's repository, layered, and pipe-and-filter architecture styles are pillars that uphold OpenCV's evolution and reliability. Having to reverse engineer a large-scale open source software such as OpenCV showed the complexity that goes into understanding a system's architecture beyond just documentation. Modular design, layering, and sound data flow are what keep these systems running. Overall, OpenCV demonstrates robustness and efficiency, keeping it a cornerstone of computer vision applications and remaining as a widely trusted and adaptable tool.

Introduction and Overview

The purpose of this report is to present the conceptual architecture of OpenCV, an open-source software library designed to provide a common infrastructure for computer vision applications and commercial machine perception products. OpenCV offers over 2,500 optimized algorithms for tasks such as detecting and recognizing people and objects, distinguishing human actions, tracking movements, and other computer vision and machine learning capabilities [1]. Several well-known companies, including Google, Intel, and IBM, utilize the library, which has over 40 million monthly downloads and plays a significant role in the technology sector, making it a prime example of a software system essential to modern computer vision applications [1]. Created in 1999 at Intel by Gary Bradsky, OpenCV was developed to advance CPU-intensive applications and provide a unified infrastructure for computer vision research and development [2]. The first public release occurred in 2000, and by 2005, OpenCV played a pivotal role in the success of Stanley, the vehicle that won the DARPA Grand Challenge [2].

Beyond its core functionalities, OpenCV has evolved to offer a suite of additional resources and services. OpenCV University provides structured courses and certifications in computer vision and deep learning, catering to both beginners and professionals seeking to enhance their skills [3]. The OpenCV AI Kit (OAK) is a hardware platform that integrates depth perception and neural inference capabilities, facilitating the development of edge AI applications [4]. OpenCV also maintains an active blog, sharing insights, tutorials, and updates on the latest advancements in the field [5].

Collaborations with industry leaders further strengthen OpenCV's ecosystem. JetBrains has partnered with OpenCV, making PyCharm the official Python IDE for OpenCV development, which streamlines the coding experience for developers [6]. This creates a more intuitive and purpose-driven environment for developers when working on projects. Additionally, RunPod collaborates with OpenCV to provide GPU resources for students and researchers, promoting the next generation of AI innovators [7]. For those who may not have access to expensive hardware and resources, this creates a great learning environment. These collaborations help enhance the usage and accessibility of OpenCV through various levels of expertise.

Architecture

In this section, the overall structure of OpenCV will be examined. The term *architecture* is defined by IEEE as: “The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution” [8]. The architecture of OpenCV consists of several interrelated components such as subsystems, modules, files, and processes. Together, these components form the conceptual architecture, providing a high-level understanding of how the system is organized and how its implementation reflects this underlying structure.

Before discussing OpenCV’s structure in detail, its overall architectural styles are examined. Three main styles can be identified in OpenCV: Repository Style, Layered Style, and Pipe and Filter Style. These architectural styles are described in the *Software Architecture: Intro and*

Styles course slides by Zhen Ming (Jack) Jiang, which provide the theoretical basis for this analysis [9].

Starting with the Repository Style, the literal core of OpenCV is the Core module, which provides a centralized store of data structures and fundamental functions that all other modules depend on [15]. According to the documentation, it defines the main data types and essential operations—such as array manipulation, type handling, and arithmetic—that form the shared foundation of the entire library [10]. Acting as the central body of information, the Core module stores and manages image and video data, which are accessed and manipulated by various modules such as Image Processing, Machine Learning, Deep Neural Networks, and Video Analysis. These modules function as independent components operating on a shared repository, enabling efficient data management and memory access while supporting diverse computer vision operations.

Next is the Layered Style. OpenCV consists of multiple modules organized hierarchically, where higher-level modules depend on the functionality provided by lower-level ones. According to the documentation, the framework is structured so that low-level components like the Hardware Abstraction Layer (HAL) handle direct hardware interaction, the Core module provides data structures and mathematical operations above it, and advanced modules such as Machine Learning or High-Level GUI build upon these foundations [11]. HAL operates at the lowest level, closest to the hardware, while the High-Level GUI module resides at the top of the hierarchy, offering user-facing functionality.

Finally, the Pipe and Filter Style is visible within the internal workflows of many OpenCV modules. Most algorithms are designed to take an input, perform a transformation or computation, and produce an output that can be passed to the next algorithm—forming a processing pipeline [11]. Each algorithm or function acts as a filter, while the data (such as images, matrices, or video frames) flows through the pipes connecting them. For instance, in a typical image-processing sequence, an image might first be converted to grayscale, then blurred, and finally passed to an edge-detection algorithm. Each of these steps independently processes the data and forwards the result, exemplifying the Pipe and Filter pattern. This approach allows OpenCV to maintain flexibility and composability, enabling developers to easily chain operations for complex computer vision tasks [11].

Functionality, Parts, and Interaction - Modules

OpenCV is organized into several modules that interact with one another and specialize in implementing functionality such as machine learning, image analysis, object detection, and much more [12]. This acts as a software layer where, at its core, is the accurately named Core module. Connecting all of this together is the Hardware Abstraction Layer (HAL), which ensures that the CPUs, GPUs, and other instruction sets' performance are optimized and efficient [13]. HAL acts as a hardware layer that allows code to run just as efficiently on different hardware platforms, regardless of the type [13]. Additionally, OpenCV also provides several standalone applications that serve as auxiliary tools for tasks such as annotation, training, or visualization. While not part of the core architecture, they demonstrate and support the practical use of the modules [14]. The following section lists the main modules of OpenCV, providing a short examination of their functionality, interaction and architecture.

- **Image Processing:** Image Processing is responsible for image processing operations such as filtering, transforming, and edge detection [16]. It uses data structures from Core to prepare images for modules such as objdetect, stitching, dnn, etc [17]. Imgproc tools such as histograms, colour space conversions, or shape descriptors are used by developers to analyze images, extract data, and manipulate said data to make them more useful [18].
- **Image File Reading and Writing:** Imgcodecs uses functions to allow for reading and writing images to disk storages. Imencode and imdecode are used to both read and write a file using the memory buffer [19]. Other functions are also used to read one or many images using the Mat object type.
- **Video I/O:** The Videoio module in OpenCV handles all the heavy lifting for reading (input), writing (output), and manipulating video streams and image sequences [20]. One main class is VideoCapture, which is used to open a webcam, read video files, and more. Another is VideoWriter, which is used to write back frames to a video file [20]. Videoio abstracts underlying systems like Direct Show (DSHOW), Microsoft Media Foundation (MSMF), Video 4 Linux (V4L), etc. It acts as a bridge between the code and external video frameworks. The beauty of this is that OpenCV wraps all the complexity in a single consistent API working on different operating systems [21]. In short, Videoio is the backbone of all video-based computer vision tasks in OpenCV.
- **High-Level GUI:** HighGUI module is OpenCV's built-in graphical user interface system. This module is used to handle window creation, image and video display, and simple user interface interactions by handling mouse events and keyboard commands [21]. It gives easy-to-use functions to show images or video frames, capture keyboard input, close all the opened display windows cleanly, and get the client's screen coordinates (image width and height). It is a lightweight toolkit for visualization and debugging and helpful for testing computer vision code [22]. In short, HighGUI displays images, plays video, captures camera feeds, and interacts with them visually. It is used for prototyping and visualization in computer vision workflows.
- **Video Analysis:** The Video module (Video Analysis) in OpenCV is designed for processing and analyzing motion in video streams. It turns raw video footage into meaningful motion data to help in understanding movement and behaviour. It includes algorithms for background subtraction, object tracking, motion, and optical flow. This module's goal is to extract temporal information (how objects/subjects change from frame to frame), which is the backbone of applications like surveillance, activity recognition, autonomous driving, and gesture tracking.

  A few key features and classes inside this module include:
  - Background Subtraction: great for detecting people, cars, or any motion in footage [23].
  - Optical flow: track pixel movements between frames to estimate motion direction and speed. Perfect for projects like video stabilization [24].
  - Object tracking: the module integrates with OpenCV's tracking API to follow objects through frames [24].
  - Motion: visualize and quantify how motion evolves over time in frames [25].

- **Camera Calibration and 3D Reconstruction:** Calib3d simulates a pinhole camera model. The view of a scene is obtained by projecting a scene's 3D point into the image plane using a perspective transformation, which forms the corresponding pixel [26]. Both are represented in homogeneous coordinates, i.e., as 3D and 2D homogeneous vectors, respectively [26].
- **2D Features Framework:** Features2d is the collection of tools and algorithms for detecting and describing salient features (like corners or edges) in 2D images, along with methods to match these features across different images [27]. Such algorithms include the BOW (Bag Of Words) algorithm, which is used to detect features of an image and form a histogram of words based on those features [27]. In addition, the FAST algorithm is used to efficiently detect corners [27].
- **Object Detection:** Objdetect is used for object detection such as faces, documents, and more [28]. Cascade classifiers, such as Haar Cascade, are prominent algorithms used for detection [29]. In addition, applications such as traincascade are used to help train the detection feature of the algorithm. Facial recognition is achieved by analyzing the input image and outputting the coordinates of the various notable facial features within the image. These coordinates are saved as a Mat object [30].
- **Deep Neural Network:** The DNN module contains an API for new layers, which are building blocks of neural networks - complex artificial networks that learn complex patterns and features from large datasets [31, 56]. It is primarily made for forward pass computations, such as network testing. This leads to efficient image classification and object detection. It consists of many layers that each perform some important aspect of deep learning and compute some aspect of image detection, such as normalizing, permuting, and reshaping layers [31].
- **Machine Learning:** The Machine Learning module provides several machine learning algorithms such as SVM, Decision Trees, Random Trees, K Nearest Neighbors, and more [32]. This module allows for training and classification of different objects, images, or characteristics found in images or videos. As a result, this module assists in decision making [32]. The ML module contains all the necessary components required to train data. It includes a training set, which is composed of sample vectors, each with an identical structure of ordered features. In supervised learning, these samples are paired with responses—such as labels for classification or values for regression—to teach an algorithm the mapping between them [33]. In contrast, unsupervised learning uses only the samples themselves to find underlying patterns [34]. ML implements feed-forward artificial neural networks - multi-layer perceptrons, the most commonly used type of neural networks [35].
- **Fast Library for Approximate Nearest Neighbours:** It is a library that contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high-dimensional features [36].

- **Computational Photography:** The Computational Photography module allows for the use of digital computation and software to enhance and enable new forms of digital imaging, moving beyond the limitations of traditional optics by merging computing, digital sensors, and modern optics to create novel applications [37]. Within this context, the module provides a variety of photo processing algorithms, e.g., Inpainting or Denoising. It follows a component-based style with elements of a layered and filter-oriented architecture, as each algorithm is encapsulated as a distinct component building on lower-level modules, while allowing data (images) to be passed through successive processing stages. To function properly, these algorithms depend on the Image Processing and Core modules [38].
- **Images Stitching:** The Image Stitching module in OpenCV enables the creation of seamless panoramic images by combining multiple overlapping inputs into a single view [39]. It implements a range of distinct components, e.g., Feature Finding and Image Matching. Architecturally, the module follows a Pipe-and-Filter style, where input images are passed through successive processing stages [39]. The pipeline can be divided into two areas: Registration, which involves operations such as resizing, detecting, and matching features, and estimating transformations, and Compositing, which covers warping, seam estimation, blending, and final panorama generation. As with all other modules, it relies on the Core module as its foundation [40].
- **Graph API:** The G-API module in OpenCV differs from other modules as its main purpose is to decouple pipeline description from execution [41]. It enables users to express computations as graphs of operations, which are compiled at runtime into optimized execution plans for specific backends. Architecturally, G-API follows the Layered Style, consisting of three main layers: the API layer for graph construction, the Graph Compiler for transforming and optimizing operations into a directed acyclic graph, and the Backend layer for executing these graphs on hardware targets such as CPUs or GPUs [41]. Designed as an isolated module with minimal dependencies, G-API can also be leveraged by other OpenCV modules to optimize multi-step workflows. Its backend abstraction ensures extensibility and portability, while the stateless graph execution model enhances testability and consistency. Overall, this layered design supports both high performance and long-term maintainability [41].

Hardware Abstraction Layer

The Hardware Abstraction Layer is an interface in the OpenCV Core module that provides optimized hardware implementations [13]. This allows OpenCV to run efficiently on different platforms without the need to change any code. OpenCV can be built in three separate modes:

1. No HAL support - default implementation used [42, 43]
2. Statically-linked HAL - accelerated function used [42, 43]
3. Dynamically-loadable HAL support - HAL implementation at runtime and default if it fails [42, 43]

Testability & Support for Future Changes

As described in the previous chapters, OpenCV follows a modular structure where functionality is divided into modules. This architectural choice directly supports both testability and the integration of future changes. Each module is internally organized into small, well-defined entities, which allows functions to be tested in isolation and ensures that changes in one component have minimal impact on others. Most modules contain a dedicated “test” subdirectory with unit tests for functional validation, as well as a “perf” subdirectory for performance testing [44]. This approach represents the standard in the industry and enables the project to ensure everything works correctly, while also monitoring efficiency.

From an architectural perspective, this structure provides a strong foundation for evolution. New features can be introduced as separate modules or extensions to existing ones, accompanied by their own tests. In this way, OpenCV ensures that future development remains consistent with the existing codebase, while maintaining high reliability and performance. This is especially important because so many different developers are working on it concurrently.

System Evolution

OpenCV was initially released as a relatively simple core of the computer vision software field. Over time, it has gone through numerous larger changes. For example, OpenCV 2.0 was released in 2008 with a new C++ interface [45]. In 2011, OpenCV 2.3 was released with support for GPU acceleration using CUDA. In 2015, OpenCV 3.0 featured support for deep learning. In 2018, OpenCV 4.0 builds upon 3.0 with further support for deep learning and AI capabilities [45].

OpenCV continues to look to further build upon their library with proposals such as establishing bit-exact functions which aims to minimize the variance in computation when performed on different branches with different optimization algorithms which can result in things such as slight variance in floating-point results [46, 47]. These new features are as always being developed while developers continue to maintain the existing code base with some current objectives being to clean up the API, revise basic modules such as Core, Imgproc, Objdetect, etc… and improve the efficiency on certain architectures, CPUs and GPUs [48].

Control and Data Flow

The control and data flow architecture of OpenCV is fundamentally designed around a data-flow-centric, imperative programming model. The control flow is typically sequential and driven by the application code, manifesting in several key patterns [49]. For static image processing, a linear pipeline is common, where a single image is passed through a series of functions, with the output of one operation becoming the input to the next. In contrast, video processing employs a continuous loop control flow, where frames are sequentially acquired from a cv::VideoCapture object, processed, and then displayed or saved [50]. For interactive applications, OpenCV utilizes an event-driven control flow within its HighGUI module; an infinite loop waits for user input like trackbar movements or mouse clicks, which trigger callback functions to modify the image data or parameters in real-time. In advanced

implementations, these basic patterns are often combined into multi-threaded architectures to decouple acquisition from processing.

Concurrency

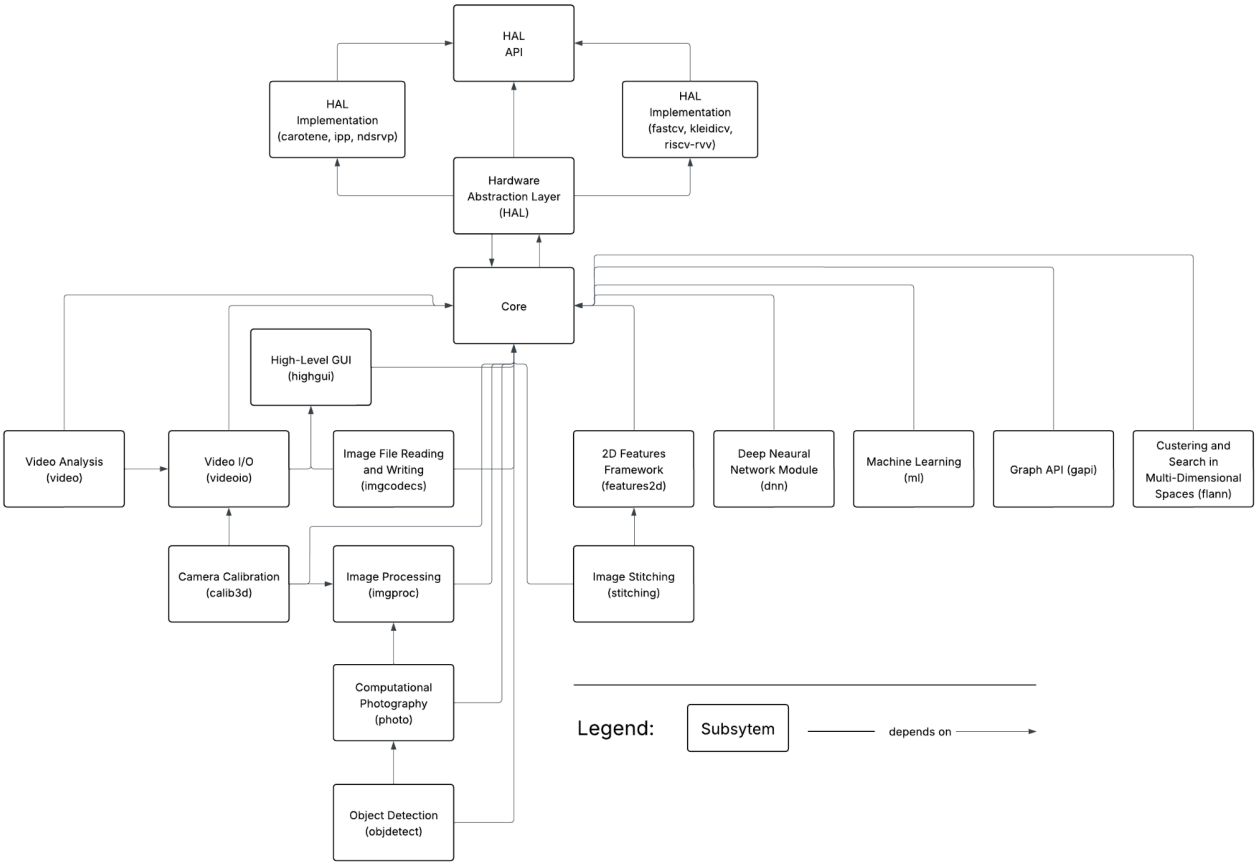
When interacting with the library at a high-level, single API calls can result in multiple threads being created in parallel to run an algorithm for faster code execution utilizing the fact a lot of computer vision is linear algebra for which the majority of computations per operation are independent of other computation within the same operation [51]. OpenCV employs CUDA/OpenCL to make use of different hardware to maximize algorithm efficiency. These tools are what allow the matrix computations to be run optimally on the GPU as opposed to the CPU which results in a night and day difference in computation speed [52, 53].

Developer Division of Responsibilities

1) Core Maintainers / Architects
   - Responsibilities: approve large API changes, decide major design/architecture, merge core PRs, enforce API stability.
2) Module Owners / Area Chairs (per-module leads)
   - Responsibilities: maintain a specific module (e.g., imgproc, videoio, or video module), review PRs in that area, keep tests & docs up to date [54].
   - Evidence: OpenCV uses an Area Chairs program and opencv_contrib hosts contributed modules; module-centric ownership is the de facto pattern [54].
3) Release / CI / Build Team
   - Responsibilities: CI pipelines (GitHub Actions / buildbots), cross-platform builds, release tagging, packaging.
   - Evidence: OpenCV GitHub Actions and external buildbots are integral to PR validation and release processes [55].
4) Quality Assurance / Testing Engineers
   - Responsibilities: unit and integration tests, regression benchmarks, performance tests (CPU/GPU), platform compatibility (ARM, Windows, macOS).
   - Evidence: CI badges and testing references in repo/Wiki; many PRs are gated by CI.
5) Documentation
   - Responsibilities: maintain docs site, tutorials, examples, API docs, tutorial notebooks.
   - Evidence: docs.opencv.org and module tutorials are core resources; coding style/wiki emphasize docs [12].

Diagrams

The following conceptual architecture diagram demonstrates the repository, layered, and pipe & filter architecture styles and shows many of the different subsystems (modules).



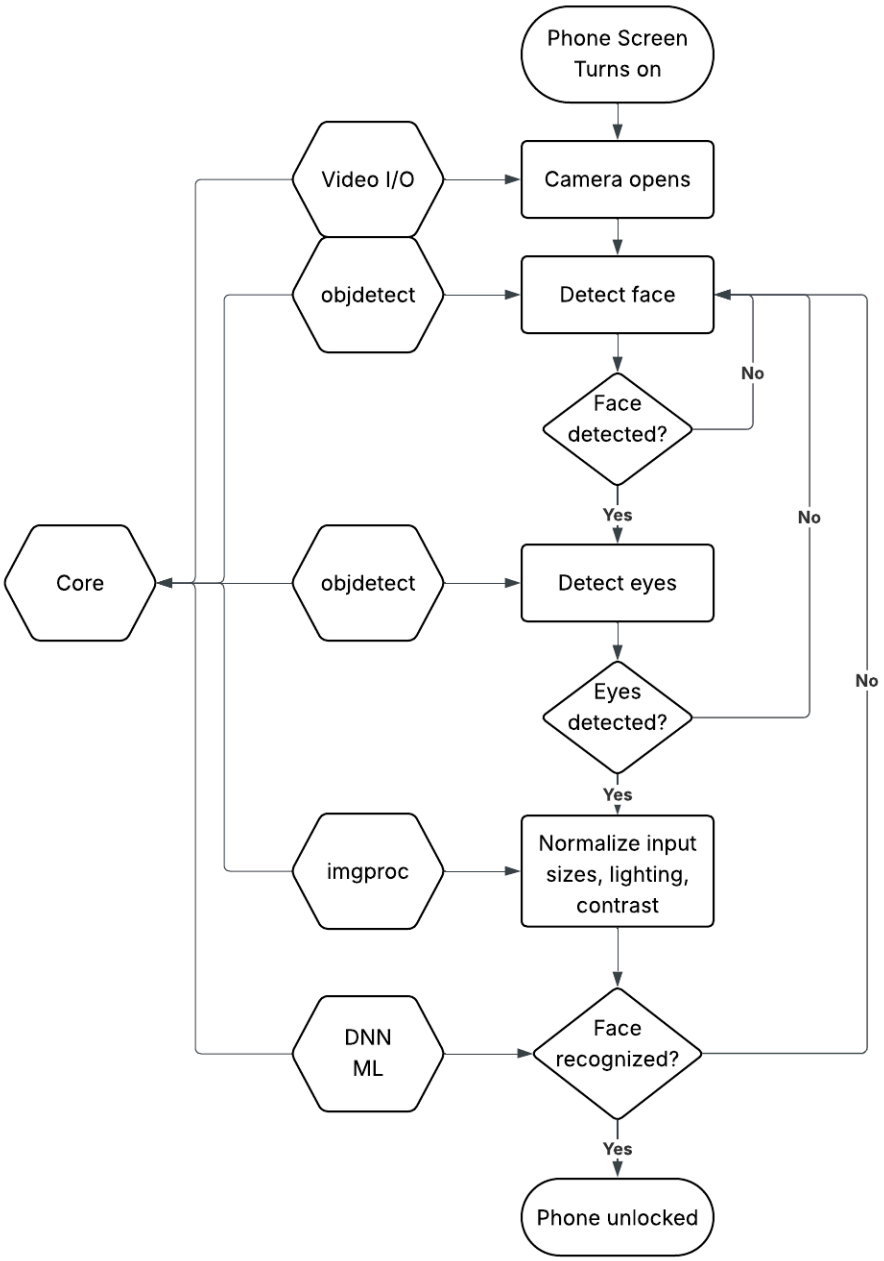
External Interfaces

There are a variety of modules discussed that make use of hardware or software outside of the bounds of OpenCV. In addition, steps are taken in order to allow the software to be used across multiple different devices without difficulty. Examples of these modules include the Video I/O module and the G-API module [41, 50] . These modules assist with connecting to hardware, optimizing the CPU, and to allow for easy portability between different devices.

Use Cases

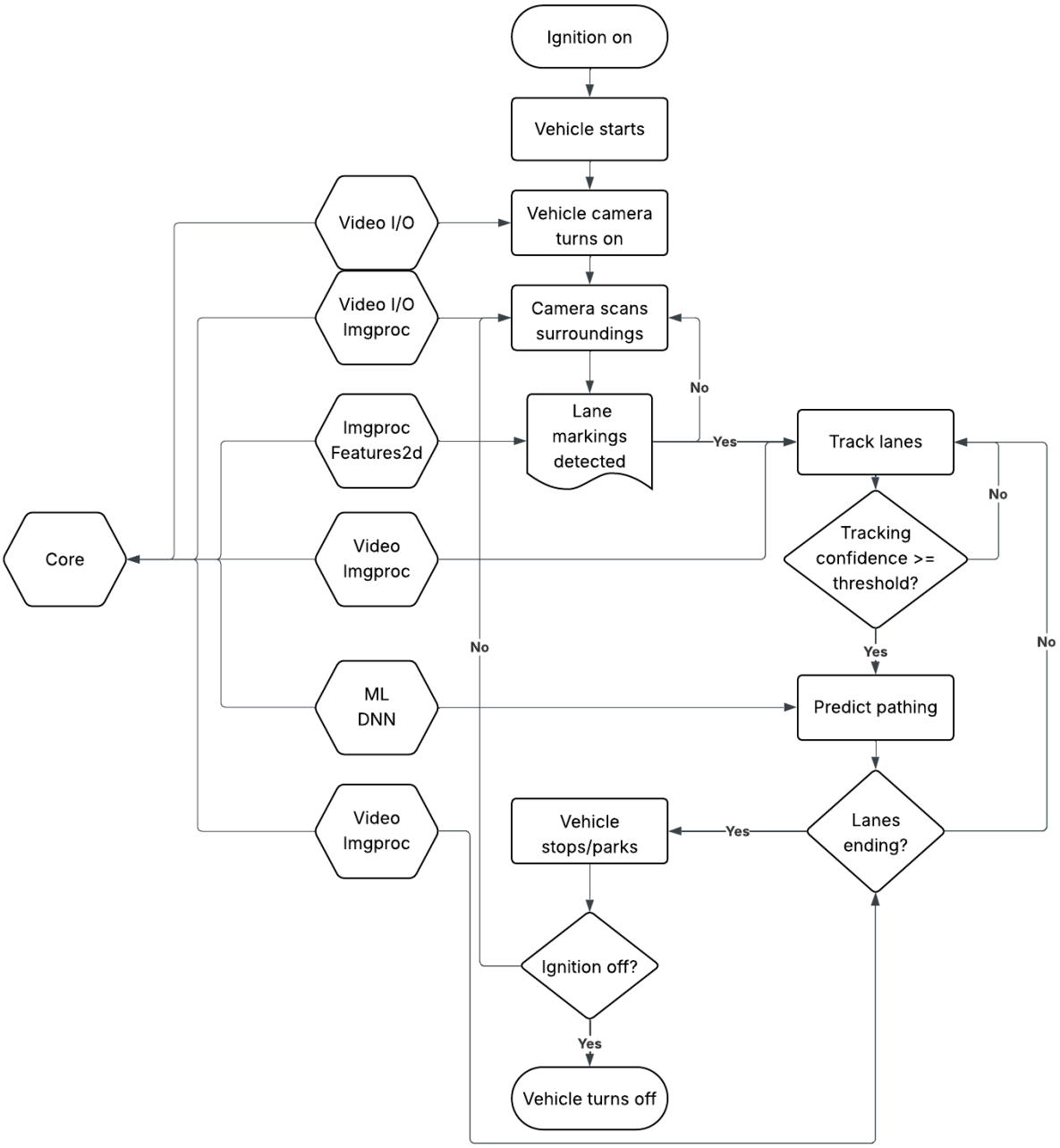
1. Facial Recognition Through Phone Unlock

One of the most common methods of unlocking a smartphone in the modern age is through face unlock. Apple, Samsung, and Google are some of the notable companies that have incorporated this feature in their devices.



2. Self Driving Vehicles

Self driving vehicles require expert analysis of what is going on while driving. Lane changes and navigation requires several modules to work together to ensure accuracy and safety of the vehicle and its positioning.



Data Dictionary

N/A

Naming Conventions

N/A

Conclusions

Through the analysis of the various modules of OpenCV, the baseline software architecture present throughout each of them has revealed itself. It can be seen that numerous architectural styles have been employed throughout the modules that have been developed in order for the software to remain efficient and optimized as it evolves and grows in complexity.

Investigation into the modules divulges the present styles employed. This was shown to be the layered, repository, and pipe and filter architectural styles. With the layered style, three layers have been developed: the hardware abstraction layer, core, and high-level layers. The hardware abstraction layer provides support at a hardware level which allows for the software to be implemented easily across multiple devices. Next, the core layer has been shown to provide the higher level modules with functions and algorithms to be used. Finally, the high-level layer contains modules that cover advanced features pertaining to the usage of OpenCV, such as the camera calibration and machine learning modules. The repository architectural style is made prevalent through the core layer, as a vast assortment of the modules created connect to this layer through their usage of its various features. Pipe and filter is applied through each function and algorithm throughout OpenCV. Data is filtered as it passes through them, allowing for it to be used in different situations.

The application of these architectural styles brings many benefits to the software, such as by bringing ease in testability and maintenance, and allowing for more optimization. These benefits have allowed the system to easily evolve over time. Due to the separation of layers, as well as having each part of the program portioned off into separate modules, testing is made more efficient due to the ability to test each segment isolated. Having the system formatted in this manner increases optimization through its ability to be concurrent. This is shown through OpenCV being able to have multiple execution threads in operation at once. Resulting from these benefits is the evolution of the software over time, which is displayed by the various versions which have been released over time, each introducing new features which further the capabilities of this software. In addition, more is planned for future versions, such as revising the core module to allow for greater efficiency.

The investigation into the architectural styles made use of by OpenCV allows for a greater understanding of how the software operates, and reveals how the system maintains, and will continue to maintain its effectiveness and efficiency for years to come.

Lessons Learned

A key lesson from this project was the significant challenge of reverse-engineering software architecture, even for well-documented open-source projects like OpenCV. We discovered that publicly accessible code and dedicated documentation do not automatically translate to a clear understanding of the underlying architectural design. Consequently, our team could only produce a conceptual model; at least until architecture extraction software is used to gain a concrete version, highlighting a critical gap between code availability and genuine architectural comprehension. This experience shows the inherent complexity of large-scale software and emphasizes that effective contribution and maintenance require a deep, foundational understanding of how the software was originally constructed and is managed.

References

1. OpenCV, “About OpenCV,” *OpenCV*, 2018. https://opencv.org/about/
2. “OpenCV: Introduction to OpenCV-Python Tutorials,” docs.opencv.org. https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html
3. “OpenCV,” Opencv.org, 2025. https://courses.opencv.org/
4. P. Nelson, “Introducing OAK: Spatial AI Powered by OpenCV,” OpenCV, Jun. 18, 2020. https://opencv.org/blog/introducing-oak-spatial-ai-powered-by-opencv/
5. “Blogs | LearnOpenCV,” LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow with code, & tutorials, Sep. 04, 2023. https://learnopencv.com/blogs/
6. P. Nelson, “Press Release: PyCharm Becomes Official IDE of OpenCV, JetBrains Joins as Silver Member,” OpenCV, Oct. 15, 2024. https://opencv.org/blog/press-release-pycharm-becomes-official-ide-of-opencv-jetbrains-joins-as-silver-member/
7. “RunPod Partners With OpenCV to Empower the Next Gen of AI Builders | Runpod Blog,” Runpod, 2023. https://www.runpod.io/blog/runpod-opencv-partnership
8. “IEEE Standards Association,” IEEE Standards Association. https://standards.ieee.org/ieee/1471/2187/
9. “York University - Shibboleth - Loading Session Information,” Yorku.ca, 2025. https://eclass.yorku.ca/pluginfile.php/7564805/mod_resource/content/1/EECS4314_04_SoftwareArchIntoAndStyles.pdf
10. “OpenCV: The Core Functionality (core module),” Opencv.org, 2025. https://docs.opencv.org/4.x/de/d7a/tutorial_table_of_content_core.html
11. OpenCV, “OpenCV: OpenCV modules,” docs.opencv.org. https://docs.opencv.org/4.x/index.html
12. Opencv, “OpenCV: OpenCV modules,” docs.opencv.org. https://docs.opencv.org/4.x/
13. “Blocked Page,” Github.com, 2025. https://github.com/vinograd47/opencv-hal-proposal/blob/master/README.md
14. opencv, “opencv/apps at 4.x · opencv/opencv,” GitHub, 2025. https://github.com/opencv/opencv/tree/4.x/apps
15. “OpenCV: Core functionality,” Opencv.org, 2025. https://docs.opencv.org/4.x/d0/de1/group__core.html
16. “OpenCV: Image Processing,” docs.opencv.org. https://docs.opencv.org/4.x/d7/dbd/group__imgproc.html
17. “OpenCV: Image Processing (imgproc module),” docs.opencv.org. https://docs.opencv.org/4.x/d7/da8/tutorial_table_of_content_imgproc.html
18. “OpenCV: Changing Colorspaces,” docs.opencv.org. https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html
19. “OpenCV: Image file reading and writing,” docs.opencv.org. https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html
20. “OpenCV: Video I/O,” Opencv.org, 2025. https://docs.opencv.org/4.x/dd/de7/group__videoio.html
21. “OpenCV: Video I/O with OpenCV Overview,” Opencv.org, 2025. https://docs.opencv.org/4.x/d0/da7/videoio_overview.html
22. “OpenCV: High-level GUI,” Opencv.org, 2024. https://docs.opencv.org/4.x/d7/dfc/group__highgui.html

23. “OpenCV: Motion Analysis,” Opencv.org, 2025. https://docs.opencv.org/4.x/de/de1/group__video__motion.html
24. “OpenCV: Object Tracking,” Opencv.org, 2025. https://docs.opencv.org/4.x/dc/d6b/group__video__track.html
25. “OpenCV: cv::motempl Namespace Reference,” Opencv.org, 2025. https://docs.opencv.org/3.4/da/d2c/namespacecv_1_1motempl.html
26. “OpenCV: Camera Calibration and 3D Reconstruction,” docs.opencv.org. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html
27. “OpenCV: Feature Detection and Description,” Opencv.org, 2025. https://docs.opencv.org/3.4/d5/d51/group__features2d__main.html
28. “OpenCV: Object Detection,” docs.opencv.org. https://docs.opencv.org/4.x/d5/d54/group__objdetect.html
29. “OpenCV: Cascade Classifier for Object Detection,” Opencv.org, 2025. https://docs.opencv.org/4.x/de/d37/group__objdetect__cascade__classifier.html
30. “OpenCV: cv::FaceDetectorYN Class Reference,” Opencv.org, 2025. https://docs.opencv.org/4.x/df/d20/classcv_1_1FaceDetectorYN.html
31. “OpenCV: Deep Neural Network module,” Opencv.org, 2025. https://docs.opencv.org/4.x/d6/d0f/group__dnn.html
32. “OpenCV: Machine Learning,” Opencv.org, 2025. https://docs.opencv.org/4.x/dd/ded/group__ml.html
33. I. Belcic and C. Stryker, “Supervised Learning,” Ibm.com, Sep. 23, 2021. https://www.ibm.com/think/topics/supervised-learning
34. IBM, “Unsupervised Learning,” Ibm.com, Sep. 23, 2021. https://www.ibm.com/think/topics/unsupervised-learning
35. mbeyeler, “opencv-machine-learning/notebooks/09.02-Implementing-a-Multi-Layer-Perceptron-in-OpenCV.ipynb at master · mbeyeler/opencv-machine-learning,” GitHub, 2017. https://github.com/mbeyeler/opencv-machine-learning/blob/master/notebooks/09.02-Implementing-a-Multi-Layer-Perceptron-in-OpenCV.ipynb
36. “OpenCV: Clustering and Search in Multi-Dimensional Spaces,” Opencv.org, 2025. https://docs.opencv.org/4.x/dc/de5/group__flann.html
37. R. Raskar and J. Tumblin, “Computational Photography.” Available: https://web.media.mit.edu/~raskar/Adm/Book06Feb08/Stuff/forMorganRaskarTumblinNov2007.pdf
38. opencv, “opencv/modules/photo/include/opencv2/photo.hpp at 4.x · opencv/opencv,” GitHub, 2025. https://github.com/opencv/opencv/blob/4.x/modules/photo/include/opencv2/photo.hpp
39. W. Lyu, Z. Zhou', L. Chen, and V. Zhou, “-NC-ND License (http://creativecommons.org/licenses/by-nc-nd/). A survey on image and video stitching.” Accessed: Oct. 03, 2025. [Online]. Available: https://pdfs.semanticscholar.org/e34c/69229fb37a741ba75f676fdfba74d8a03a22.pdf
40. “OpenCV: Images stitching,” docs.opencv.org. https://docs.opencv.org/4.x/d1/d46/group__stitching.html
41. “OpenCV: High-level design overview,” Opencv.org, 2025. https://docs.opencv.org/4.x/de/d4d/gapi_hld.html

42. “OpenCV: Hardware Acceleration Layer,” Opencv.org, 2025. https://docs.opencv.org/4.x/de/d85/group__core__hal.html
43. vpisarev, “New CPU HAL for OpenCV 5.0,” GitHub, Feb. 13, 2024. https://github.com/opencv/opencv/issues/25019
44. opencv, “opencv/opencv,” GitHub, Dec. 12, 2019. https://github.com/opencv/opencv
45. “History of OpenCV: Evolution, Founders, and Key Milestones,” Techskillguru.com, 2015. https://techskillguru.com/opencv/history-of-opencv
46. opencv, “Evolution Proposals,” GitHub, Mar. 07, 2025. https://github.com/opencv/opencv/wiki/Evolution-Proposals
47. opencv, “OE 15. Bit Exactness,” GitHub, Feb. 21, 2019. https://github.com/opencv/opencv/wiki/OE-15.-Bit-Exactness
48. opencv, “OE 5. OpenCV 5,” GitHub, Nov. 03, 2024. https://github.com/opencv/opencv/wiki/OE-5.-OpenCV-5
49. “OpenCV: Mat - The Basic Image Container,” docs.opencv.org. https://docs.opencv.org/4.x/d6/d6d/tutorial_mat_the_basic_image_container.html
50. “OpenCV: cv::VideoCapture Class Reference,” Opencv.org, 2025. https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html
51. opencv, “opencv/modules/core/src/parallel.cpp at 4.x · opencv/opencv,” GitHub, 2020. https://github.com/opencv/opencv/blob/4.x/modules/core/src/parallel.cpp
52. opencv, “opencv/modules/core/include/opencv2/core/cuda.hpp at 4.x · opencv/opencv,” GitHub, 2025. https://github.com/opencv/opencv/blob/4.x/modules/core/include/opencv2/core/cuda.hpp
53. opencv, “opencv/modules/core/include/opencv2/core/ocl.hpp at 4.x · opencv/opencv,” GitHub, 2025. https://github.com/opencv/opencv/blob/4.x/modules/core/include/opencv2/core/ocl.hpp
54. “Welcome To Zscaler Directory Authentication,” Opencv.org, 2025. https://opencv.org/opencv-area-chairs/
55. opencv, “How_to_contribute,” GitHub, Dec. 13, 2024. https://github.com/opencv/opencv/wiki/How_to_contribute
56. IBM, “What is a Neural Network?,” IBM, Oct. 06, 2021. https://www.ibm.com/think/topics/neural-networks