How My Method Works

The system captures frames from a webcam using OpenCV and converts each frame to grayscale to simplify processing, as ArUco detection doesn't require color—this reduces computational load. It then loads a predefined dictionary (DICT_6X6_250) that defines how each marker looks and maps it to a unique ID. Detection parameter such as threshold window size, perimeter rate, and corner refinement are configured to enhance accuracy and speed. OpenCV's ArucoDetector scans each frame for black-square patterns that match known marker designs. If a valid marker is detected, its ID is decoded and displayed with a green bounding box. These results are drawn directly on the live video stream, and the user can exit the program by pressing 'q' or clicking the window's X button.

Is It Efficient?

Yes, Here's Why:

The detection method is efficient because it relies on classical computer vision techniques like thresholding and contour detection rather than deep learning or GPU acceleration, making it lightweight and CPU-only. It delivers fast real-time performance at 15–30 FPS even on mid-range laptops without any lag and requires no external dependencies beyond OpenCV. By processing frames in grayscale, it reduces input size and avoids unnecessary computations on color channels. The use of adaptive parameters—such as perimeter rate and threshold window—enhances detection accuracy while minimizing false positives, effectively balancing speed and precision. Additionally, the system processes each image in a single pass without resizing, upsampling, or multiple iterations, further improving performance.

Performance & Computational Advantages

Feature	Advantage
CPU-Only	Runs on low-end hardware (Raspberry Pi, laptops) without GPU
OpenCV Built-ins	Highly optimized C++ backend = faster Python execution
No Training Required	No training time or dataset required like YOLO or CNN models
Instant Feedback	Detects markers in real-time with minimal latency
Portable	Works cross-platform (Windows, Linux, Mac)
Energy-Efficient	Minimal CPU/memory load = suitable for embedded systems