ArUco Marker Detection

Colab Link:

https://colab.research.google.com/drive/1xVsim13LBErospTFZnZQnfkFbPgXcFPm?usp=sharing

Objective:

The objective of this project was to detect ArUco markers from an image, identify their unique IDs, estimate their 3D pose (position and orientation) relative to a calibrated camera, and calculate the distance between each marker and the camera. To ensure real-world accuracy, the camera was calibrated using images of a printed checkerboard pattern. The process enables precise spatial awareness of markers, which is essential for applications in robotics, augmented reality, and computer vision.

Process:

The entire ArUco marker detection and pose estimation pipeline was implemented using **Python and OpenCV**, and executed in **Google Colab** for ease of access, platform independence, and reproducibility. This setup allowed seamless integration with Google Drive for loading input images and saving calibration outputs.

To handle cases where the type of ArUco marker used in the image was unknown, I implemented a **dictionary auto-detection** approach. It iteratively tested all commonly used ArUco dictionaries (such as `DICT_4X4_50`, `DICT_5X5_50`, `DICT_6X6_50`, and others). For each dictionary, the number of detected markers was recorded, and the dictionary yielding the highest number of valid detections was selected for the rest of the process.

Accurate pose estimation required **camera calibration**, which was conducted using a printed **7×10 inner corner checkerboard**. Multiple calibration images were taken from different angles and distances. Using `cv2.findChessboardCorners` and `cv2.calibrateCamera`, the intrinsic camera matrix and lens distortion coefficients were computed. These were saved as `camera_matrix.csv` and `dist_coeffs.csv` respectively, and reused for all future pose estimation tasks without needing to recalibrate.

After dictionary selection and calibration, the actual **marker detection and pose estimation** process was performed. Each input image was first converted to grayscale for optimal detection. The `cv2.aruco.detectMarkers()` function was used to locate markers and extract their IDs. With the known physical size of each marker (2.8 cm), the `cv2.aruco.estimatePoseSingleMarkers()`

function computed each marker's pose in 3D space, returning a rotation vector (`rvec`) and translation vector (`tvec`). These were used to draw 3D coordinate axes (X, Y, Z) on each marker using `cv2.drawFrameAxes()` for visual confirmation.

Finally, the **distance from the camera to each marker** was calculated using the Euclidean norm of the translation vector. This was computed as the square root of the sum of squares of the X, Y, and Z components (`distance = √(x² + y² + z²)`), providing a real-world measurement of how far each marker is from the camera. These values were printed alongside their corresponding marker IDs for easy analysis.

Efficiency and Performance

1. Modular & Automated:

- Automatically selects the best dictionary
- Modular calibration and pose estimation steps

2. Minimal Dependencies:

- No deep learning or GPU needed
- Pure OpenCV-based, CPU-efficient

3. Real-Time Ready:

- Pose estimation and distance computation per frame is lightweight
- Ideal for integration into real-time applications (robotics, AR, etc.)

4. Reusable Calibration:

- Camera parameters stored and reloaded for future sessions, avoiding recalibration

Computational Advantages:

The system offers several computational advantages that make it both lightweight and efficient. It relies entirely on OpenCV, which uses an optimized native C++ back-end accessed through

Python bindings. This eliminates the need for heavy external libraries such as TensorFlow or PyTorch, resulting in significantly lower memory usage and faster execution. Because of its minimal resource requirements, the method performs well even on basic CPUs without GPU acceleration. This efficiency also makes it ideal for deployment on embedded systems and edge devices such as Raspberry Pi, NVIDIA Jetson, or other low-power platforms commonly used in robotics and IoT applications.