
BARCODE
QR
CODE
2D CODE

Barcode and QR Code Detection

Introduction

Barcode and QR code detection is an essential application in modern computer vision, with widespread use in inventory management, retail, logistics, and other industries. This project aims to develop a real-time detection system using Python and OpenCV. The system processes video frames captured from a camera to identify, decode, and annotate barcodes and QR codes, thereby providing a robust solution for scanning and interpreting encoded data.

The system leverages various image processing techniques, including edge detection, sharpening, thresholding, and histogram equalization, to enhance the visual features of barcodes and QR codes. Additionally, the built-in OpenCV `QRCodeDetector` and `barcode_BarcodeDetector` APIs are utilized for precise decoding of encoded data.

Methodology

The system implementation can be broken down into the following steps:

1. Capturing Video Frames

- A video stream is initialized using `cv2.VideoCapture(0)` to capture frames from the camera.
- Each frame is processed in real-time to detect barcodes and QR codes.

2. Preprocessing

Preprocessing is applied to enhance the input frame and improve the accuracy of detection:

- **Edge Detection**:
  - A custom kernel is defined to highlight vertical edges, which are common in barcodes.
  - Convolution is applied using `cv2.filter2D()` for edge detection.
- **Sharpening**:
  - A sharpening kernel is applied to enhance the contrast and visibility of the features.
- **Grayscale Conversion**:
  - The sharpened frame is converted to grayscale using `cv2.cvtColor()` to simplify further processing.
- **Gaussian Blur**:
  - Noise reduction is achieved using Gaussian blurring (`cv2.GaussianBlur()`).
- **Adaptive Thresholding**:

    - Binary thresholding is applied adaptively using `cv2.adaptiveThreshold()` to increase the contrast of the regions containing codes.
- **Histogram Equalization**:
    - Contrast Limited Adaptive Histogram Equalization (CLAHE) is used to enhance contrast further.

3. QR Code Detection and Decoding

- The `cv2.QRCodeDetector()` is initialized and used to detect and decode QR codes.
- If a QR code is detected:
    - Its decoded value is printed.
    - The detected region is marked with a green polygon using `cv2.polylines()`.

4. Barcode Detection and Decoding

- The `cv2.barcode_BarcodeDetector()` is used to detect and decode barcodes.
- If barcodes are detected:
    - Decoded information is printed.
    - Detected regions are highlighted with green polygons using `cv2.polylines()`.

5. Display and User Interaction

- The annotated frame, containing the detected codes and regions, is displayed using `cv2.imshow()`.

- The application runs in a loop, continuously processing frames until the user presses the 'a' key to quit.

6. Error Handling

- Checks are performed to ensure the camera is accessible and frames are read correctly.
- The application gracefully exits in case of errors.

Results

The system performs the following tasks effectively:

1. **Detection**: Identifies QR codes and barcodes in real-time from video frames.
2. **Decoding**: Extracts the encoded data from the detected QR codes and barcodes.
3. **Visualization**: Highlights detected regions with green polygons for user feedback.
4. **Interaction**: Provides a user-friendly interface with real-time display and a termination key ('a').

Example Outputs:

- **QR Code Detection**:
  - Detected Value: `https://example.com`
  - Region: Marked with a green polygon.
- **Barcode Detection**:

- Detected Value: `123456789012`
- Region: Marked with a green polygon.