웹캠 기반 QR 코드 리더 및 성능 분석

- QR코드 인식률 개선과 발전
- QR코드 인식을 어떠한 조건(야외 심한 빛 반사, 어두운 조명, 흔들림, 약간의 훼손, 구겨짐)에서도 인식률 향상을 위한 프로젝트
- 개발 기간 8/13 ~ 8/21
-https://github.com/jaemin6/qr_scan

문제 정의 및 목표
해결하고자 한 문제:
- 기존의 복잡하고 불안정한 QR 코드 인식 시스템을 개선하여, 외부 라이브러리 의존성 없이도 안정적이고 효율적인 실시간 QR 코드 스캔 기능을 구현하고자 함

측정 가능한 목표
- 기존 QR 코드 인식보다 인식 성공률 30% 이상 달성

- 평균 인식 시간 0.5초 이내로 단축

문제 선택 이유:
- 상용 서비스에서 QR 코드 인식 오류로 인한 사용자 불편이 잦다는 점에 주목,
특히 저화질, 왜곡된 QR 코드도 정확하게 인식하는 솔루션이 필요하다고 판단하여 이 프로젝트를 시작하게 됨

기술 스택
주요 언어: Python

핵심 프레임워크/라이브러리: OpenCV, Pillow, Numpy

개발 도구: Git, VS Code

AI 사용: Gemini 2.5 Flash (대화형 피드백을 통한 코드 디버깅 및 솔루션 탐색)

핵심 기능
기능명: 실시간 웹캠 QR 코드 스캔 및 전처리

- 기능 설명: 웹캠 영상을 실시간으로 분석하여 화면에 나타난 QR 코드를 즉시 인식하고, 인식 실패 시 흑백 변환 및 히스토그램 평활화(Equalization)를 통해 이미지 품질을 향상시킴

- 사용된 핵심 기술: cv2.VideoCapture 및 cv2.QRCodeDetector, cv2.cvtColor, cv2.equalizeHist

- 달성한 성과: 복잡한 모델 없이도 빠르고 안정적으로 QR 코드를 인식하는 데 성공했으

며, 전처리를 통해 인식률을 높임

기능명: QR 코드 경계선 시각화

- 기능 설명: 웹캠 영상을 실시간으로 분석하여 화면에 나타난 QR 코드를 즉시 인식, 인식 실패 시 **그레이스케일 변환 및 히스토그램 평활화(Equalization)**를 통해 이미지 품질을 향상

- 사용된 핵심 기술: OpenCV의 cv2.polylines(경계선 시각화), cv2.QRCodeDetector, cv2.cvtColor (그레이스케일 변환), cv2.equalizeHist (히스토그램 평활화)

- 달성한 성과: 복잡한 모델 없이도 빠르고 안정적으로 QR 코드를 인식하는 데 성공했으며, 전처리를 통해 저품질 이미지의 인식률을 높임

기능명: 자동 URL 연결 및 통계 분석
- 기능 설명: 인식된 QR 코드가 유효한 URL일 경우 자동으로 웹 브라우저를 열고, 스캔 과정의 통계(총 프레임, 인식 프레임, 인식률, 평균 인식 시간)를 계산해 보여줌

- 사용된 핵심 기술: 파이썬의 webbrowser 모듈, time 모듈 및 numpy를 활용한 통계 연산

- 달성한 성과: 수동적인 작업 없이 QR 코드의 정보를 바로 활용할 수 있게 하여 편의성을 극대화

기술적 문제 해결 사례

사례 1: 이미지 품질 저하로 인한 인식률 문제 해결
- 직면한 문제: 초기 테스트에서 웹캠의 조명이 어둡거나 QR 코드가 흐릿할 때 인식 성공률이 크게 떨어지는 문제가 발생

- 문제 분석 과정: 원본 이미지가 낮은 대비와 밝기 때문에 QR 코드의 패턴이 뭉개져 디코더가 이를 인식하지 못한다고 결론

- 시도한 해결 방법들:

- 1차 시도: 단순히 **이진화(Binarization)**를 적용해 보았지만, 이미지의 전반적인 밝기 편차가 심할 경우 오히려 인식률이 떨어지는 부작용이 있었음

- 2차 시도: **오츠(Otsu's) 이진화**를 시도, 이는 이미지의 히스토그램을 분석해 자동으로 최적의 임계값을 찾아주는 기법으로, 이진화의 성능을 개선했지만 여전히 한계가 명확함

- 3차 시도: **블랙햇 모폴로지(Black Hat Morphology)**를 적용해 보았으나, 이는 이미지의 어두운 부분을 강조하여 뭉개진 패턴을 복구하는 데 효과적이었지만, QR 코드 주변의 노이즈도 함께 강조되는 문제가 생김

- 4차 시도: **컨투어(Contour) 및 다중 컨투어(Multiple Contours)**를 사용하여 QR 코드의 외곽선을 찾으려 했지만 정확한 네 모서리를 찾는 것이 까다로웠고, 배경 노이즈까지 컨투어로 잡히는 문제가 발생함

- 5차 시도: **가우시안 블러(Gaussian Blur)**: 초기 시도로 가우시안 블러를 적용하여 이미지의 노이즈를 줄여보려 했지만 QR 코드의 미세한 패턴까지 흐려져 오히려 인식률이 떨어지는 부작용이 생김

- 1~5차 시도를 하면서 느낀 점은 지금껏 배운 이미지 전 처리 기술들을 거의 다 써봤지만 결국 처리하는 과정에서 노이즈를 제거한다고 했던 점이 오히려 QR인식에 있어 새로운 노이즈를 만들 수 있고, 이미 한번 인식 한 QR을 다시 인식을 할 시 이미 개선된 이미지라는 점에서 오히려 인식이 늦어지는 오류를 발견함

- 최종 해결 방법: 복잡한 전처리 기법을 하나씩 적용하기보다, 가장 안정적이고 효율적인 조합을 찾기로 결정했습니다. **그레이스케일 변환 (cv2.cvtColor)**과 **히스토그램 평활화 (cv2.equalizeHist)**를 함께 적용하는 방법을 채택하였고, 이 전처리 과정은 이미지의 전반적인 밝기와 대비를 고르게 만들어 QR 코드 패턴을 뚜렷하게 부각시켰고, 가장 높은 인식률을 보임

- 해결 결과: 전처리 과정 추가 후, 저조도 환경에서의 인식률이 15% 가량 향상됨

- 학습한 내용: 다양한 이미지 전처리 및 객체 탐지 기법(이진화, 모폴로지, 컨투어)을 실습하며 각 기법의 장단점과 한계를 명확히 이해함,
문제 해결에는 여러 기술을 시도해보고, 가장 적합한 조합을 찾아내는 경험이 중요하다는 것을 깨달았음

사례 2: AI 모델 학습 실패와 기술 스택 재정립
- 직면한 구체적 문제: 처음에는 다양한 환경(각도, 조명, 왜곡)에서 안정적인 QR 코드 인식 성능을 확보하기 위해 딥러닝 기반의 AI 모델을 도입하려 함

- 문제 분석 과정: 초기에는 모든 문제를 최신 AI 기술로 해결할 수 있을 것이라 생각

- 시도한 해결 방법들:

- 1. Roboflow 데이터 학습: Roboflow 플랫폼을 이용해 직접 QR 코드 데이터를 수집하고 라벨링하여 학습용 데이터셋을 구축

- 2. yolov11n.pt 모델 생성: 이 데이터셋을 바탕으로 YOLOv11n.pt 모델을 직접 학습시켜 커스텀 모델을 만듦

- 최종 해결 방법: 모델의 높은 복잡성과 성능 병목 현상으로 인해 프로젝트의 본질적인 목표(빠른 인식)에 부합하지 않는다고 판단, 과감하게 AI 모델 사용을 포기하고 OpenCV

기반의 경량화된 솔루션으로 방향을 전환하게 됨

- 해결 결과: 불필요한 복잡성을 제거하여 개발 속도를 높이고, OpenCV 기반의 효율적인 파이프라인을 구축하는 데 성공함

- 학습한 내용: 모든 문제에 최신 기술이 최적의 해결책은 아님을 깨달았고 문제의 본질을 파악하고 상황에 맞는 기술을 선택하는 것이 가장 중요한 역량임을 배우게 됐음

기술적 학습 성과
새로 학습한 기술 3가지:
- 1. AI 모델 학습 워크플로우: Roboflow를 활용한 데이터셋 구축부터 yolov11n.pt 모델 학습 및 내보내기까지, 딥러닝 모델 개발의 전체 과정을 경험하게 됨

- 2. OpenCV의 영상 처리 파이프라인: 웹캠 입력(VideoCapture)부터 이미지 전처리, 객체 탐지(QRCodeDetector), 그리고 시각화(polylines, imshow)에 이르는 일련의 과정을 직접 구현하며 영상 처리의 기본 구조를 이해하는데 큰 도움이 됨

- 3. Pillow 라이브러리: OpenCV와 함께 사용하여 이미지에 한글 텍스트를 오버레이하는 방법을 학습, 이 과정에서 폰트 로드 및 이미지 색상 채널 변환(BGR에서 RGB로) 등 이미지 처리 라이브러리 간의 호환성 문제를 해결하는 실무 역량을 키우게 됨

학습 과정에서 어려웠던 점과 극복 방법:

- 어려웠던 점: 초기에는 딥러닝 모델 학습 및 환경 설정에서 반복적인 오류에 직면, 이로 인해 프로젝트 진행에 큰 어려움을 겪음

- 극복 방법: 문제를 단순화하는 접근법을 채택하게 됨.
 과도한 AI 모델 사용 대신, 문제의 본질인 QR 코드 인식을 위해 OpenCV와 같은 더 경량화되고 안정적인 라이브러리를 사용하기로 결정, 이 결정은 프로젝트를 그나마 기한내에 마무리하는 계기가 됨...

 ## Git 커밋 히스토리:
총 커밋 수: 66

의미 있는 커밋 메시지:
- 1. 다 실패작
- 2. yolo + 조명 개선 / yolo 에러 / 학습된 객체 못찾음:
이 커밋들은 딥러닝을 시도했지만 실패했다는 중요한 커밋, 이 이후 방향전환을 하게 됨
- 3. 히스토그램 평활화, 컨투어 버림:
이미지 전 처리 기술을 많이 시도했으나 효과가 없고 오히려 인식률이 떨어지는 결론을 내리고 과감히 포기하고 다른 방법을 찾게됨. 불필요한 기술에 매달리지 않고 문제 해결에 효율적인 방법을 찾아나가려고 함
- 4. + 가우시안 블러, 임계값 이진화 / 그레이스케일, 미디언 블러, 샤프닝 적용:

최종적으로 성공한 기술 해결책을 구현했지만 그래도 핵심인 인식률에 있어서 안좋은 기술을 뭔지 알게 됨
- 5. 최종 pdf 저장 프로그램

프로젝트 성찰

목표 달성도 평가: `기존 QR 코드 인식보다 인식 성공률 30% 이상 달성`에서 50% 수준으로 달성

잘 된 점과 아쉬운 점:

잘 된 점:
- 복잡한 AI 모델에 의존하지 않고, 기본적인 영상 처리 기술만으로도 충분히 안정적인 QR 코드 리더를 개발한 점. 문제의 본질에 집중하여 효율적인 솔루션을 찾은 것이 가장 큰 성과임

아쉬운 점:
- 1. 다양한 조명 환경(매우 밝거나 어두운 곳)과 QR 코드의 심한 왜곡까지 완벽하게 처리하는 데는 한계가 있었고, 특히 웹캠의 다양한 환경에 맞춘 최적화가 부족
- 2. 너무 최신화 기술에 많은 시간을 투자하여 시간을 많이 소비함, 이미지 전 처리 기술, 로보플로우를 활용한 yolov11 모델 학습에 많은 시간이 소비 됨

다시 한다면 달리 할 점:
1. 프로젝트 주제를 시간이 조금 들어가더라도 신중하게 선택할 것 같음
2. 초기 단계에서부터 문제 해결에 가장 적합한 기술을 신중하게 선택할 것
3. 딥러닝 모델이 모든 문제의 답이 아님을 미리 파악하여 개발 시간을 단축했을 것 같음