Document Scanner - Advanced Edge Detection and Corner Detection

A highly optimized C++ application for automatic document detection in images using OpenCV. The system employs advanced computer vision techniques to identify document boundaries with high precision, achieving an IoU improvement from 0.765692 to 0.839870 through parameter optimization.

📁 Data Structure

The application expects the following directory structure:

```
project_root/
├── docscan.cpp          # Main source code
├── data/                # Dataset directory
│   ├── img_1.png        # Input images (numbered 1-10)
│   ├── img_1_optc.txt   # Ground truth coordinates for img_1
│   ├── img_2.png
│   ├── img_2_optc.txt
│   └── ...              # Continue for img_3 through img_10
├── json/                # Generated JSON reports (auto-created)
├── output/              # Visualization images (auto-created)
└── README.md
```

Ground Truth Format (`*_optc.txt`)

Ground truth files contain document corner coordinates in a specific format:

```
(x1,y1),(x2,y2),(x3,y3),(x4,y4)
```

Important: Ground truth coordinates are normalized to a fixed coordinate system (0,0) to (449,599), regardless of actual image dimensions. The application automatically scales these coordinates to match the processed image size.

🔧 Build Requirements

- **C++17** compatible compiler (g++ recommended)
- **OpenCV 4.x** with development headers
- **pkg-config** for library management
- **Optional**: OpenCV contrib modules (`ximgproc`) for enhanced line detection

Installation (Ubuntu/Debian)

bash

```
sudo apt update
sudo apt install build-essential pkg-config
sudo apt install libopencv-dev libopencv-contrib-dev
```

Build Command

bash

```
g++ -std=c++17 docscan.cpp -o docscan `pkg-config --cflags --libs opencv4`
```

🚀 Usage

Single Image Processing

bash

```
./docscan input_image.png ground_truth.txt
```

Dataset Processing (Batch Mode)

bash

```
./docscan --dataset data/
```

Help

bash

```
./docscan
# Displays: Usage: ./docscan img.png gt.txt | ./docscan --dataset DIR
```

🔍 Algorithm Pipeline

1. Image Preprocessing

- **Input Scaling**: Images are resized to max dimension of 600px for consistent processing
- **Grayscale Conversion**: RGB → Grayscale using OpenCV's optimized conversion
- **CLAHE Enhancement**: Contrast Limited Adaptive Histogram Equalization
  - Clip limit: 3.875 (optimized)
  - Tile size: 9×9 (optimized)

2. Edge Detection & Enhancement

- **Sobel Operators**: Compute gradients in X and Y directions using 32-bit float precision

- **Gradient Magnitude**: Calculate `√(Gx² + Gy²)` for edge strength
- **Adaptive Thresholding**: OTSU method for automatic threshold selection
- **Morphological Closing**: 4 iterations (optimized) to connect broken edges

3. Contour Analysis

- **Contour Extraction**: Find all contours using `RETR_LIST` mode
- **Median Gradient Calculation**: Statistical measure of edge strength for scoring

4. Candidate Generation

The system generates document candidates through three complementary methods:

Method 1: Polygon Approximation

- **Douglas-Peucker Algorithm**: Simplify contours to polygons
- **Epsilon**: 0.005 × contour perimeter (optimized)
- **Filter**: Only convex 4-sided polygons are considered

Method 2: Minimum Area Rectangle

- Applied to each contour individually
- Provides robust rectangular approximations
- Handles rotated documents effectively

Method 3: Line Segment Detection (Optional)

- **Fast Line Detector (FLD)**: Detects straight line segments
- **Canny Edge Detection**: Preprocessing for line detection
- **Top 4 Lines**: Select longest segments and fit bounding rectangle

5. Candidate Scoring System

Each candidate quadrilateral is evaluated using four weighted criteria:

Area Fitness (Weight: 0.329)

```cpp
areaFit = 1 - |candidateArea - 0.458×imageArea| / (0.458×imageArea)
```

Favors documents covering ~45.8% of image area (optimized target).

Whiteness Score (Weight: 0.266)

```cpp
whiteness = clamp((documentMean/backgroundMean - 1) / 0.5, 0, 1)
```

Documents typically appear brighter than background.

Gradient Fitness (Weight: 0.208)

```cpp
gradFit = edgeMean / (edgeMean + medianGradient)
```

Measures edge strength along document borders.

Aspect Ratio Fitness (Weight: 0.197)

```cpp
ARfit = 1 - min(|aspectRatio - √2| / 1.0, 1.0)
```

Optimized for A4-like documents (√2 ≈ 1.414 ratio).

6. Quality Filters

Before scoring, candidates must pass strict quality checks:

- **Self-Intersection**: Reject self-crossing quadrilaterals
- **Minimum Area**: Must cover ≥2.9% of image area (optimized)
- **Border Kill**: Reject if >47.6% of perimeter touches image borders (optimized)

7. Selection & Refinement

- **Best Candidate**: Highest scoring candidate with score ≥0.3 (optimized threshold)
- **Fallback**: If no good candidate, use minimum area rectangle of largest contour
- **Border Refinement**: Expand edges by ±2 pixels if candidate is safely inside borders
- **Coordinate Clipping**: Ensure all points remain within image boundaries

📊 Output Files

1. Text Coordinates (`*_predc.txt`)

```
(x1,y1),(x2,y2),(x3,y3),(x4,y4)
```

Detected document corners in counter-clockwise order, starting from top-left.

2. JSON Reports (`json/*.json`)

```
json
```

```
{
  "image": "img_1.png",
  "size": [450, 600],
  "quad": [[x1,y1], [x2,y2], [x3,y3], [x4,y4]],
  "gt_quad": [[gx1,gy1], [gx2,gy2], [gx3,gy3], [gx4,gy4]],
  "iou": 0.8234
}
```

3. Visualization Images (`output/*_boxes.png`)

- **Red Lines**: Detected document boundaries with "DET" label
- **Green Lines**: Ground truth boundaries with "GT" label
- **Line Width**: 3 pixels for clear visibility
- **Labels**: Positioned near the top-left corner of each quadrilateral

4. Console Output

```
"img_1.png": IoU=0.8234
"img_2.png": IoU=0.7891
...
Mean IoU=0.8398
```

📈 Performance Metrics

- **IoU (Intersection over Union)**: Primary evaluation metric
- **Optimized Performance**: Mean IoU improved from 0.7657 to 0.8399
- **Processing Speed**: ~600px max dimension ensures fast processing
- **Robustness**: Multiple detection methods provide fallback options

🔧 Parameter Optimization

The following parameters have been fine-tuned for optimal performance:

Parameter	Value	Description
CLAHE Clip Limit	3.875	Contrast enhancement
CLAHE Tile Size	9×9	Local adaptation window
Morphology Iterations	4	Edge connection strength
Polygon Epsilon	0.005	Approximation precision
Area Threshold	0.029	Minimum candidate area
Border Kill Threshold	0.476	Maximum border contact
Target Area Ratio	0.458	Expected document size
Minimum Score	0.3	Candidate acceptance threshold
Scoring Weights	[0.329, 0.266, 0.208, 0.197]	Feature importance

🛠️ Advanced Features

Multi-Method Detection

- **Redundancy**: Three independent detection methods
- **Consensus**: Score-based selection from all candidates
- **Fallback**: Guaranteed output even for challenging images

Coordinate System Handling

- **Automatic Scaling**: Handles different image resolutions
- **Ground Truth Mapping**: Converts fixed coordinates to actual dimensions
- **Precision**: Sub-pixel accuracy in coordinate detection

Robust Preprocessing

- **Adaptive Enhancement**: CLAHE adjusts to local image characteristics
- **Multi-scale Gradients**: Sobel operators capture edges at optimal scale
- **Statistical Thresholding**: OTSU method adapts to image content

🔍 Troubleshooting

Common Issues

1. **Build Errors**

   bash

   ```
   # Check OpenCV installation
   pkg-config --modversion opencv4
   # Verify headers
   find /usr -name "opencv2" 2>/dev/null
   ```

2. **Missing Ground Truth Files**
   - Ensure `*_optc.txt` files exist for each image
   - Check file naming convention matches exactly
3. **Low IoU Scores**
   - Verify ground truth coordinate format
   - Check image quality and document visibility
   - Ensure proper lighting and contrast
4. **Memory Issues**
   - Large images are automatically resized
   - Monitor system resources during batch processing

Debug Mode

For detailed debugging, modify the code to output intermediate processing steps:

- Edge detection results
- Contour visualization
- Candidate scoring details

📋 License & Attribution

This implementation uses optimized parameters derived from systematic tuning and evaluation. The algorithm combines classical computer vision techniques with modern optimization approaches for robust document detection in varying conditions.