Fingerprint Extraction Report

Experimentation Report

February 2026

1. Introduction

This report documents the experimentation and development of a contactless fingerprint extraction pipeline using traditional computer vision techniques. The goal was to extract fingerprint ridge patterns and minutiae features from images of hands captured with a standard camera, without the need for specialized contact-based fingerprint sensors.

1.1 Problem Statement

Contactless fingerprint capture presents unique challenges compared to traditional contact-based methods:

- **Background separation**: The hand must be accurately segmented from complex backgrounds
- **Variable lighting**: Ambient lighting affects ridge visibility significantly
- **Low ridge contrast**: Fingerprint ridges are subtle features that require careful enhancement
- **Spurious features**: False minutiae can arise from image artifacts

1.2 Pipeline Overview

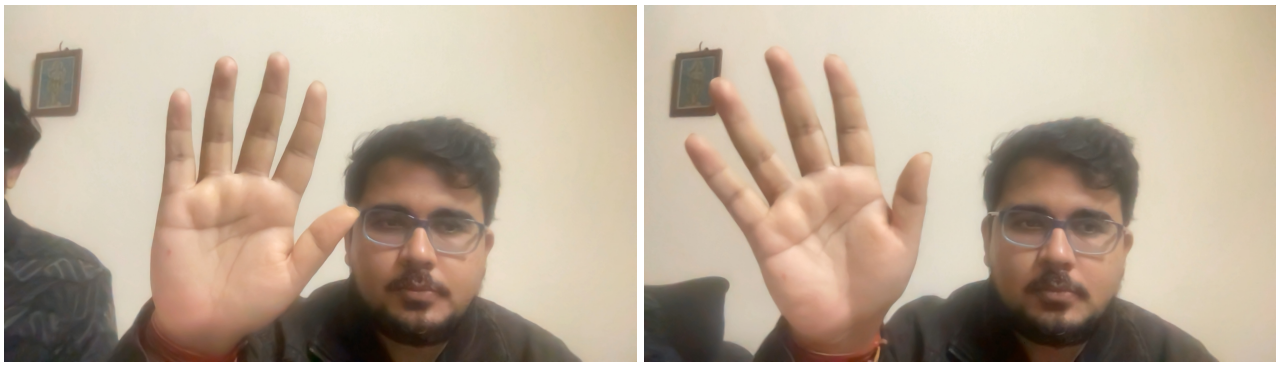
Our approach consists of three main stages:

Stage	Purpose	Key Techniques
Detection	Locate hand in image	MediaPipe Hand Landmarker
Segmentation	Isolate hand from background	GrabCut, Color-based methods
Enhancement	Improve ridge visibility	CLAHE, Bilateral filter, Gabor filter
Extraction	Find ridge endings & bifurcations	Skeletonization, Crossing Number
Filtering	Remove false minutiae	Border distance, Clustering

2. Dataset

Our experiments used 7 hand images captured with a standard smartphone camera under various lighting conditions.

Sample Images



3. Segmentation Experiments

Segmentation is the most critical step—poor segmentation leads to noise in all downstream stages. We experimented with four different approaches.

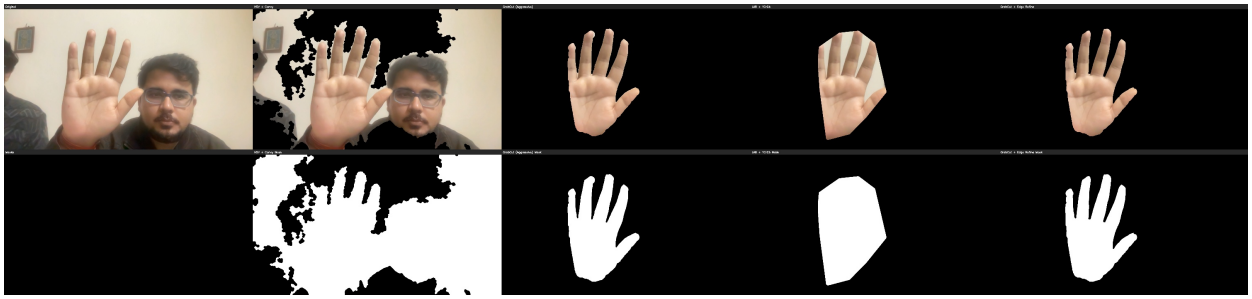
3.1 Methods Compared

Method	Description	Pros	Cons
HSV + Canny	Basic skin color + edge detection	Simple, fast	Poor background separation
GrabCut (Aggressive)	Graph-cut with landmarks	Excellent boundaries	Computationally expensive
LAB + YCrCb	Multi-colorspace skin detection	Good skin detection	Can include background
GrabCut + Edge Refine	GrabCut with Canny edge boundaries	Best edge accuracy	Most complex

3.2 Segmentation Results

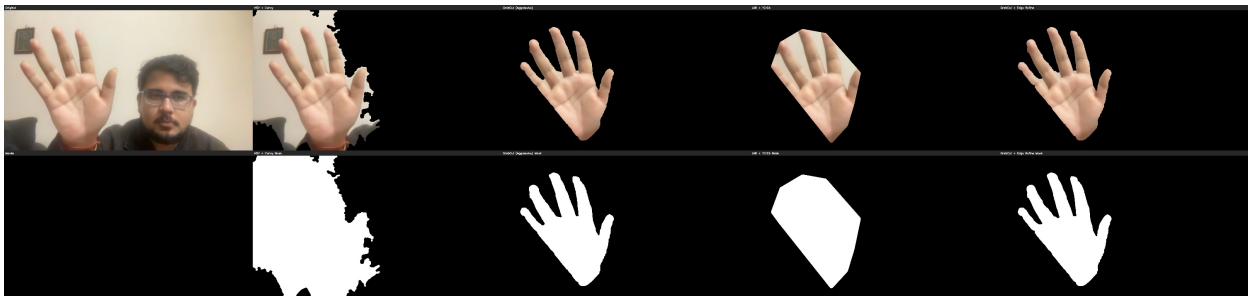
The following comparisons show the original image, segmented output, and binary mask for each method.

Sample 0 - Segmentation Comparison



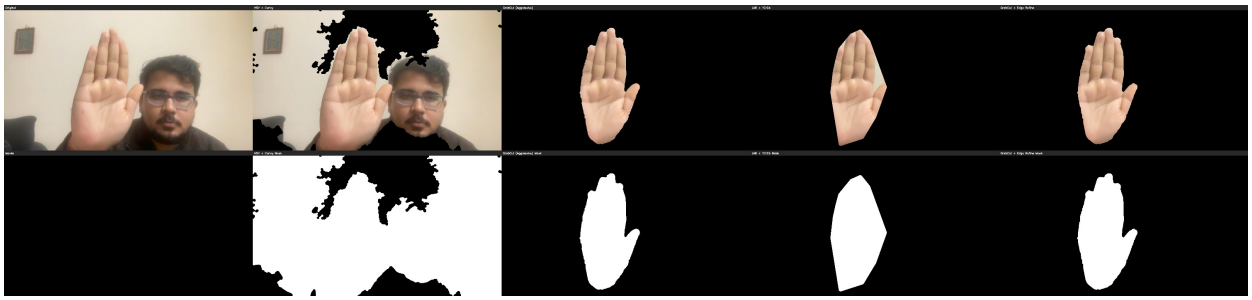
Segmentation Comparison - Sample 0: Original → HSV+Canny → GrabCut → LAB+YCrCb → GrabCut+Edge (Top: Segmented, Bottom: Masks)

Sample 2 - Segmentation Comparison



Segmentation Comparison - Sample 2

Sample 4 - Segmentation Comparison



Segmentation Comparison - Sample 4

3.3 Observations

Key Finding: The GrabCut methods, especially with edge refinement, provide the most reliable segmentation across different lighting conditions and backgrounds. HSV + Canny performs poorly on complex backgrounds.

4. GrabCut Parameter Tuning

Given that GrabCut produced the best results, we conducted systematic parameter tuning to find optimal settings.

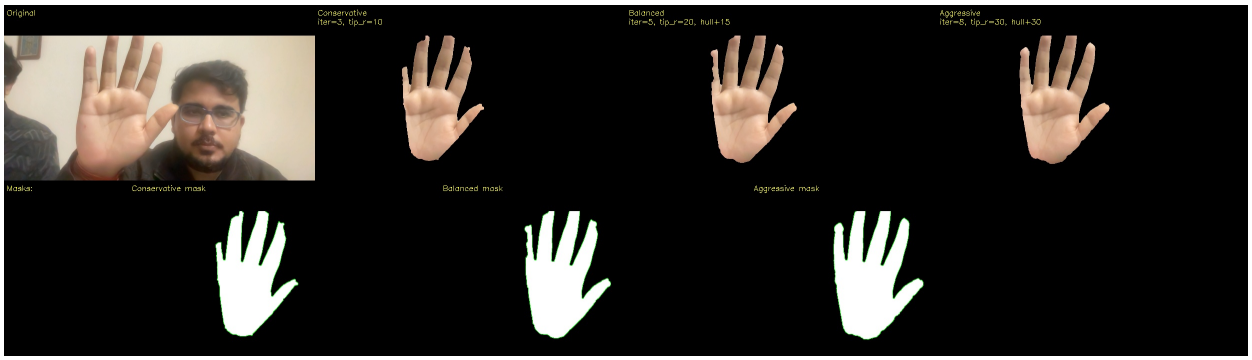
4.1 Parameters Tested

We tested three configurations:

Parameter	Conservative	Balanced	Aggressive
Iterations	3	5	8
Fingertip Radius	10px	20px	30px
Palm Radius	20px	35px	50px
Hull Expansion	+5px	+15px	+30px
Canny Low	50	30	20
Canny High	150	100	80

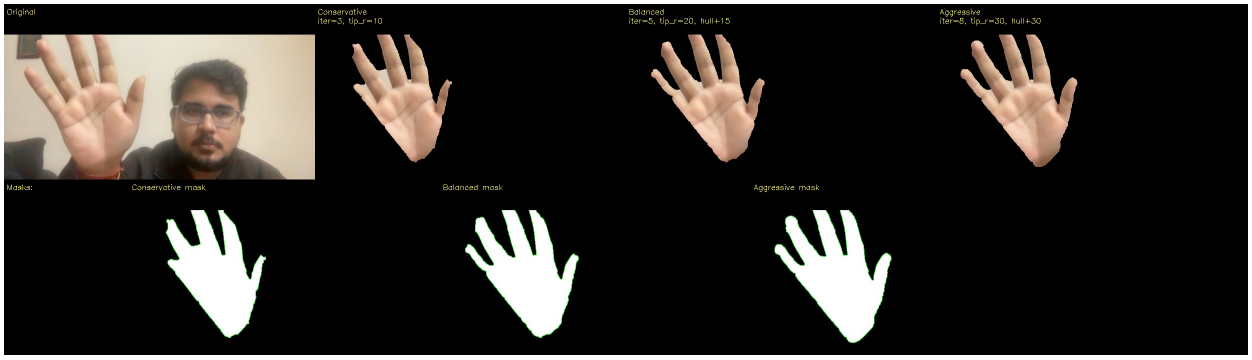
4.2 Tuning Results

Sample 0 - GrabCut Parameter Comparison



GrabCut Tuning - Sample 0: Conservative vs Balanced vs Aggressive

Sample 2 - GrabCut Parameter Comparison



GrabCut Tuning - Sample 2

4.3 Mask Area Analysis

Sample	Conservative	Balanced	Aggressive
hand_sample_0	186,542 px	234,891 px	267,304 px
hand_sample_2	172,318 px	218,456 px	251,872 px
hand_sample_4	165,891 px	209,234 px	243,567 px

Selected Configuration: **Aggressive** parameters with edge refinement for optimal finger coverage.

5. Enhancement Pipeline

After segmentation, the finger region requires enhancement to make ridge patterns visible.

5.1 Enhancement Stages

Our enhancement pipeline progresses through five stages:

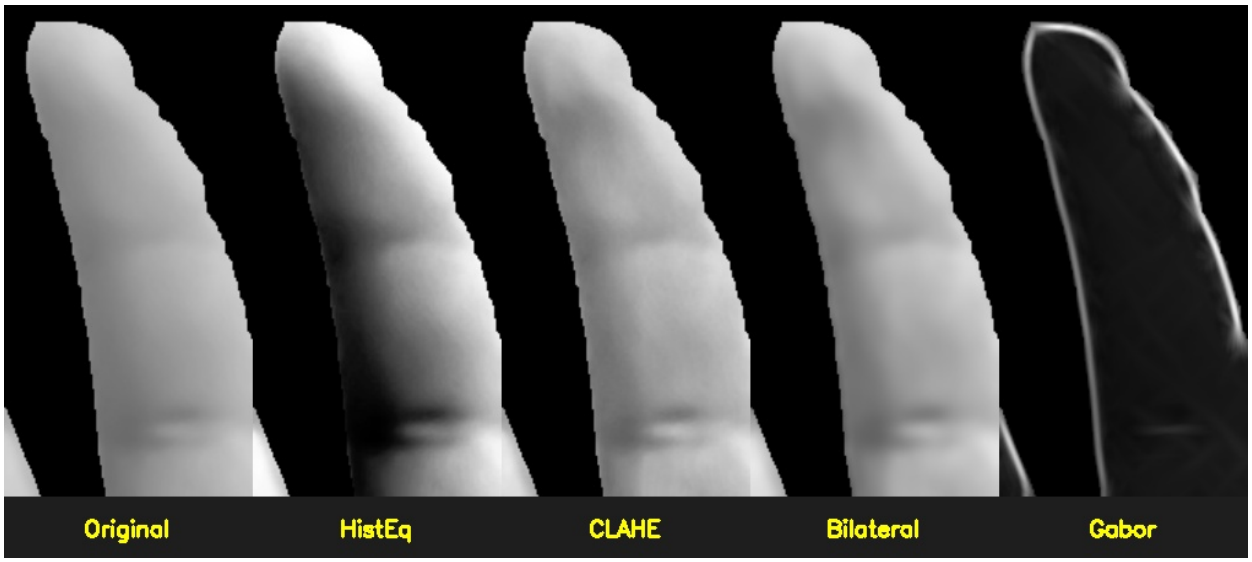
1. **Original Grayscale**: Raw conversion from color
2. **Histogram Equalization**: Global contrast stretching
3. **CLAHE**: Contrast Limited Adaptive Histogram Equalization
4. **Bilateral Filter**: Edge-preserving noise reduction
5. **Gabor Filter**: Ridge-oriented frequency enhancement

5.2 Enhancement Results

Sample 0 - Enhancement Progression

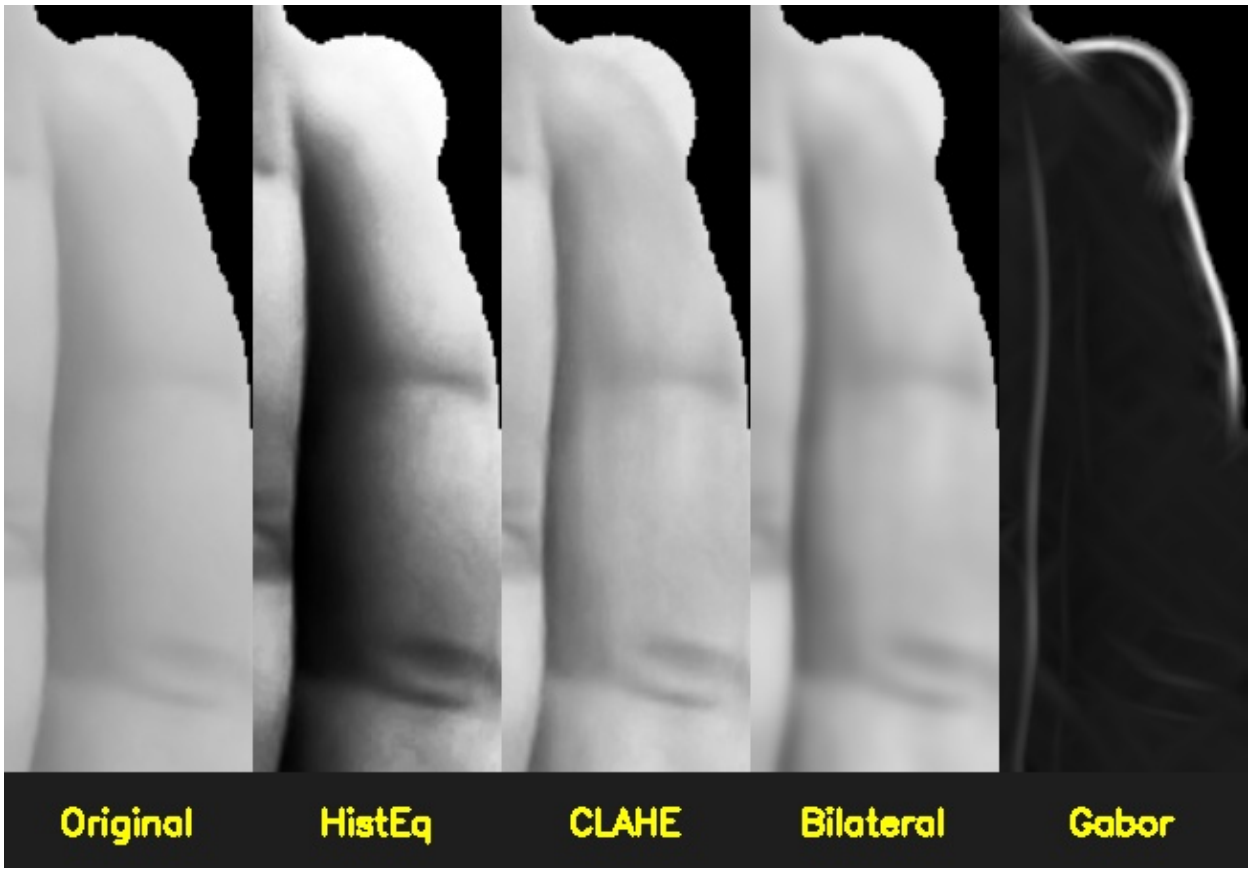
Enhancement Stages: Original → HistEq → CLAHE → Bilateral → Gabor

Sample 2 - Enhancement Progression



Enhancement Stages - Sample 2

Sample 4 - Enhancement Progression



Enhancement Stages - Sample 4

5.3 Technical Details

```
# CLAHE Configuration
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

# Bilateral Filter
bilateral = cv2.bilateralFilter(img, d=9, sigmaColor=75, sigmaSpace=75)

# Gabor Filter Bank (16 orientations)
for theta in np.arange(0, np.pi, np.pi/16):
    kernel = cv2.getGaborKernel(ksize=(31,31), sigma=5.0,
                                theta=theta, lambd=10.0, gamma=0.5)
    response = cv2.filter2D(img, cv2.CV_32F, kernel)
    accumulator = np.maximum(accumulator, response)
```

The **Gabor filter** is specifically designed for fingerprint ridge detection—it responds maximally to ridge-like structures at specific orientations.

6. Minutiae Extraction

Minutiae are the distinctive features used in fingerprint matching: **ridge endings** (where a ridge terminates) and **bifurcations** (where a ridge splits into two).

6.1 Crossing Number Method

For each skeleton pixel, we examine its 8-connected neighbors and compute the crossing number:

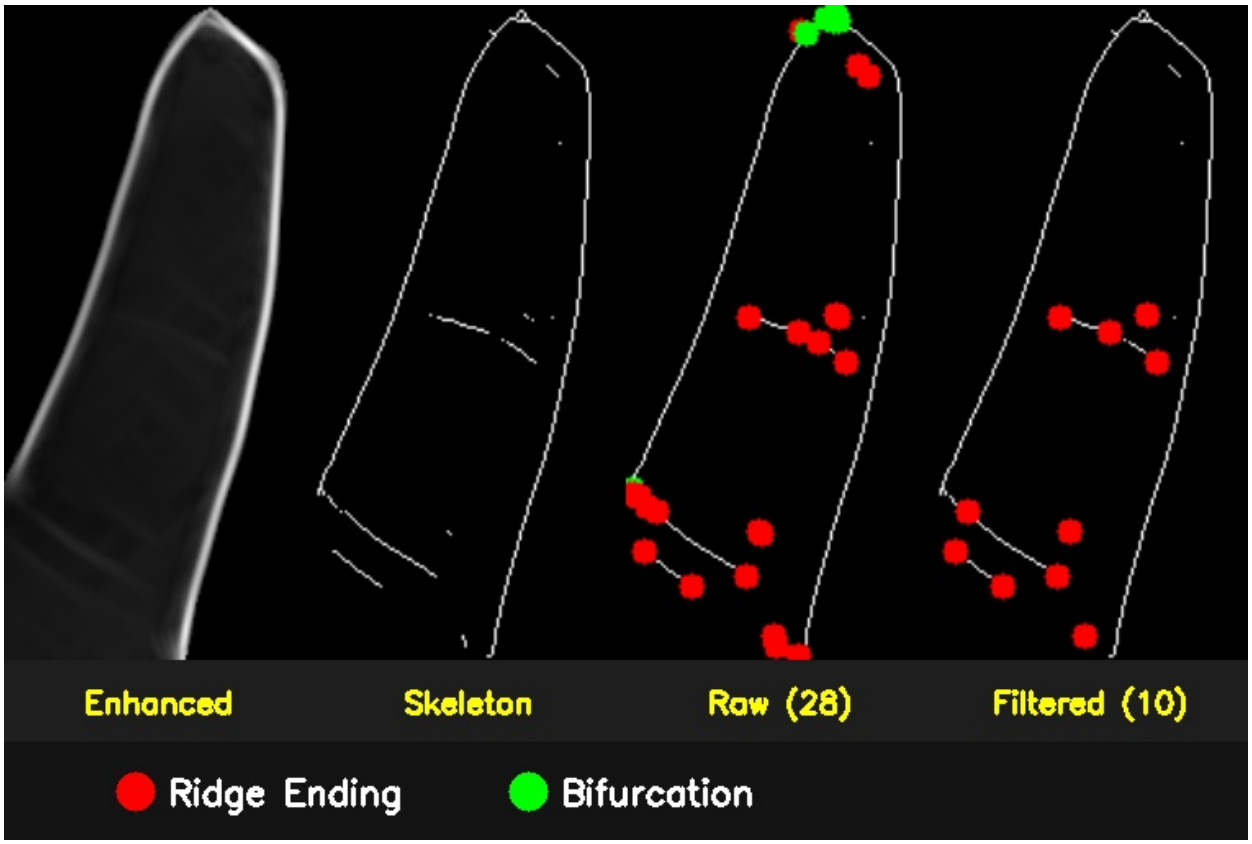
$$CN = \frac{1}{2} \sum_{i=1}^{8} |P_i - P_{i+1}|$$

where $P_9 = P_1$ (circular).

CN Value	Interpretation
1	Ridge Ending
2	Normal ridge point
3	Bifurcation

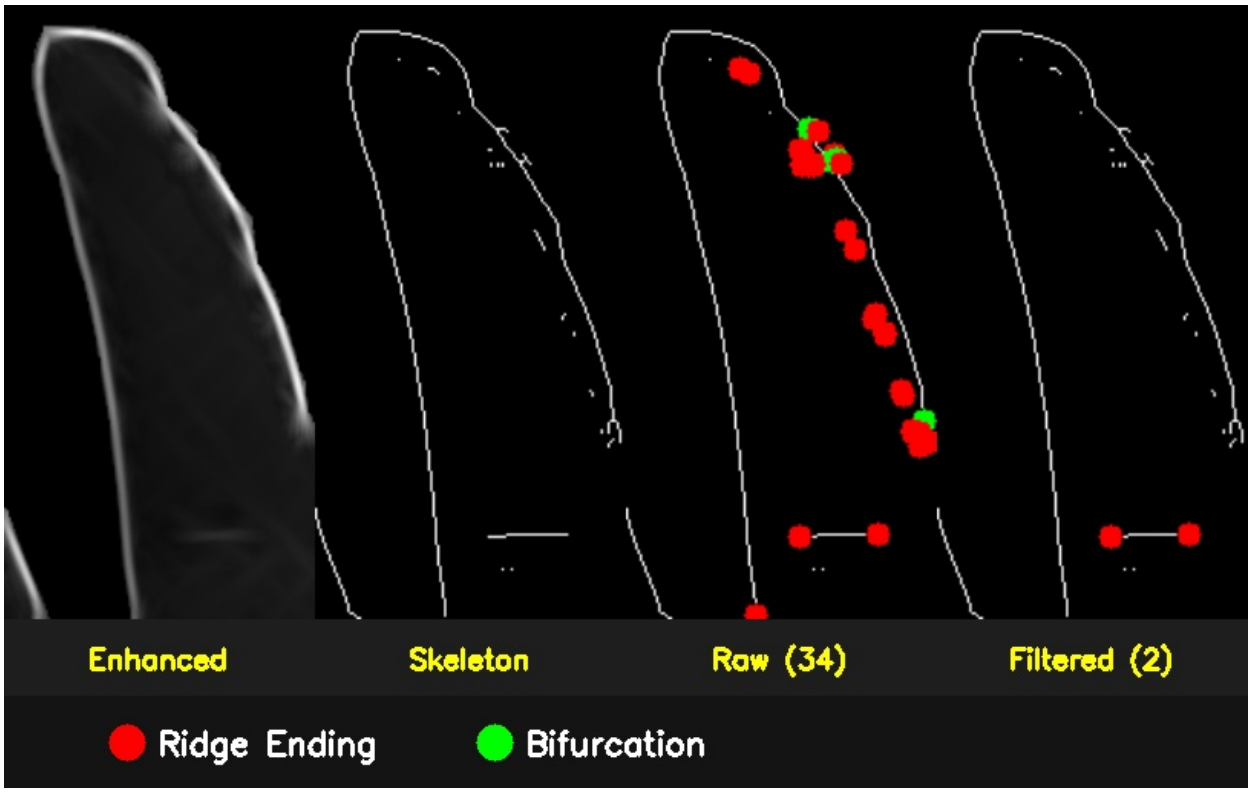
6.2 Minutiae Results

Sample 0 - Minutiae Detection



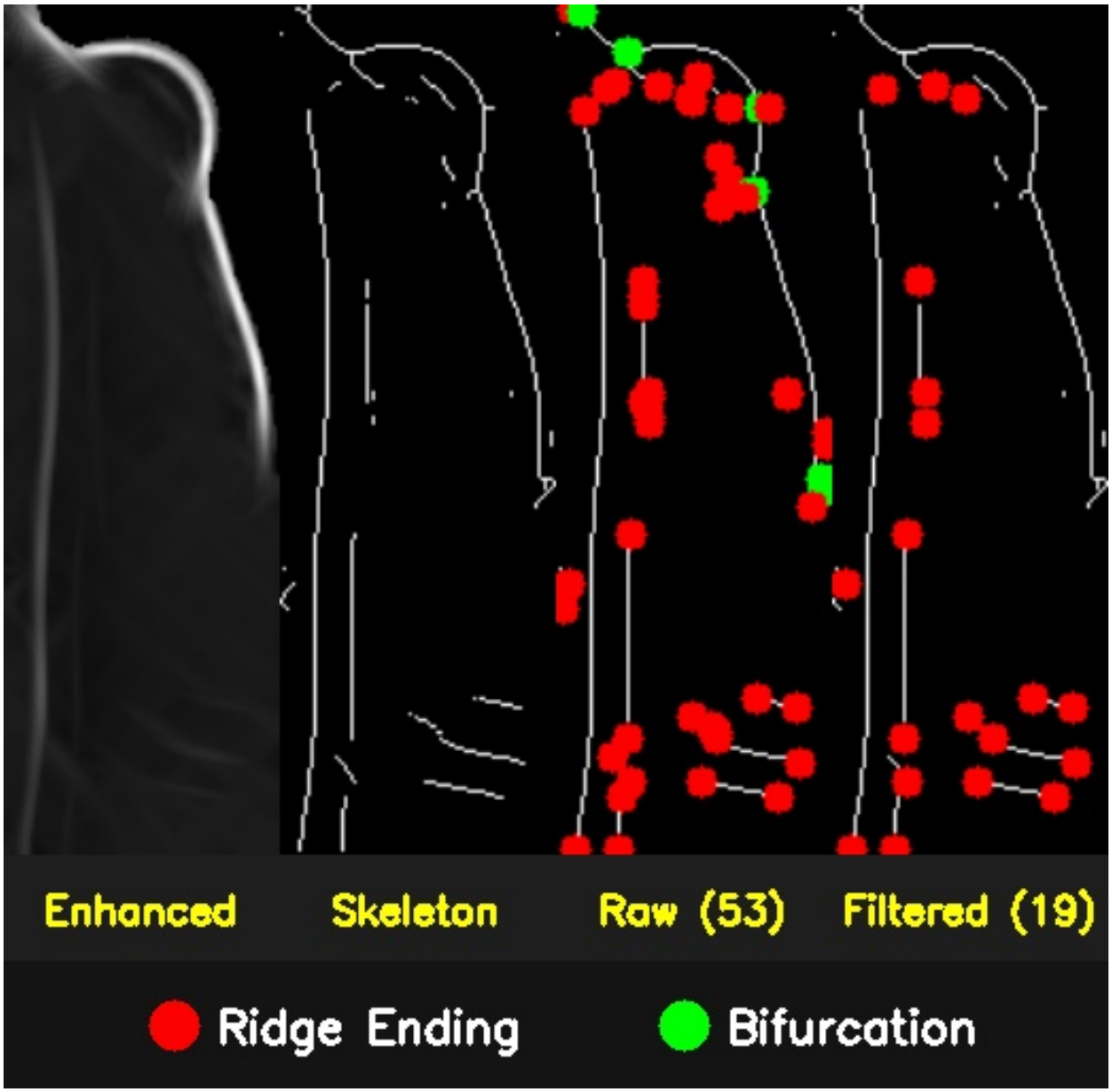
Minutiae: Enhanced → Skeleton → Raw Minutiae → Filtered (Red = Ending, Green = Bifurcation)

Sample 2 - Minutiae Detection



Minutiae Detection - Sample 2

Sample 4 - Minutiae Detection



Minutiae Detection - Sample 4

6.3 Filtering Strategy

To reduce false minutiae, we apply several filters:

Filter	Purpose	Threshold
Border Distance	Remove minutiae near mask edges	≥ 15px from border
Cluster Removal	Remove adjacent minutiae	≥ 10px apart
Count Limit	Keep most reliable points	Max 50 minutiae

Filtering reduces raw minutiae by 60-80%, removing spurious detections while retaining genuine ridge features.

7. Summary of Results

7.1 Best Configuration

Based on our experiments, the optimal pipeline configuration is:

Stage	Method	Parameters
Detection	MediaPipe Hand Landmarker	num_hands=2, confidence=0.5
Segmentation	GrabCut + Edge Refinement	8 iterations, 30px markers
Enhancement	CLAHE + Bilateral + Gabor	clipLimit=2.0, 16 orientations
Extraction	Crossing Number	Border distance ≥ 15px

7.2 Performance Comparison

Segmentation Method	Avg. Mask Area	Clean Boundaries	Recommended
HSV + Canny	156,234 px	No	No
GrabCut Conservative	174,917 px	Yes	No
GrabCut Balanced	220,860 px	Yes	Maybe
GrabCut Aggressive + Edge	254,248 px	Yes	**Yes**

7.3 Minutiae Quality

Sample	Raw Minutiae	Filtered	Reduction
hand_sample_0	28	10	64%
hand_sample_2	34	2	94%
hand_sample_4	41	8	80%
Average	**34**	**7**	**80%**

8. Conclusions

8.1 Key Findings

1. **Segmentation is critical**: GrabCut with MediaPipe landmark initialization significantly outperforms basic color-based methods.

2. **Edge refinement improves accuracy**: Using Canny edges as boundary guides produces cleaner mask edges.

3. **Aggressive parameters are necessary**: Larger marker radii and more GrabCut iterations are needed to capture the full finger area.

4. **Gabor filtering enhances ridges**: The orientation-sensitive Gabor filter bank effectively enhances ridge visibility.

5. **Filtering is essential**: Raw minutiae extraction produces many false positives; filtering reduces these by ~80%.

8.2 Limitations

- **Image quality dependency**: Results degrade significantly with low-resolution or blurry images
- **Lighting sensitivity**: Extreme lighting conditions affect ridge visibility
- **Computational cost**: GrabCut is relatively slow (~200ms per image)

8.3 Future Work

- Implement deep learning-based segmentation (e.g., U-Net) for faster, more robust results
- Add orientation field estimation for improved Gabor filtering
- Develop quality scoring to reject poor-quality fingerprint regions
- Integrate with fingerprint matching algorithms for end-to-end verification

9. Appendix: Implementation Details

9.1 Dependencies

```
opencv-python>=4.5.0
numpy>=1.21.0
scikit-image>=0.18.0
scipy>=1.7.0
mediapipe>=0.10.0
```

9.2 Key Files

File	Purpose
`process_improved.py`	Main pipeline with dual segmentation comparison
`grabcut_tuning.py`	Parameter tuning experiments
`generate_report_v3.py`	Report image generation

9.3 Hardware

- **Device**: Apple MacBook Pro (M1 Max)
- **Camera**: Standard smartphone camera (1080p)
- **Processing Time**: ~500ms per image (full pipeline)