

# **Face detection**

## Introduction

Face detection is a fundamental task in computer vision with applications in security, biometrics, and human-computer interaction. Traditional approaches often rely on **color-based segmentation** combined with geometric heuristics to identify facial regions. This report examines a MATLAB implementation that detects faces using **YCbCr skin color modeling**, morphological processing, and validation through contrast and aspect ratio checks.

## Problem Definition

The goal is to detect human faces in images using a simple, efficient method based on **skin color segmentation in the YCbCr color space**. The system identifies skin-colored regions using predefined thresholds, refines the mask with filtering and morphological operations, and validates possible face regions based on shape (aspect ratio) and texture (contrast). This approach avoids complex machine learning models, aiming for real-time performance, but it may struggle with varying lighting, non-face skin regions, and diverse backgrounds.

## Selected Algorithms and Their Mathematical Foundations

This face detection system uses several image processing algorithms grounded in basic mathematical concepts. Here's a breakdown of the key components and the math behind them:

### 1. Color Space Conversion (RGB to YCbCr)

**Purpose:** To separate luminance (Y) from chrominance (Cb and Cr) for more effective skin color detection.

**Mathematical Basis:**

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

This transformation separates brightness (Y) from color components (Cb and Cr), allowing for easier skin detection under varying lighting.

### 2. Skin Color Segmentation (Thresholding)

**Purpose:**

To isolate potential skin regions based on typical chrominance ranges of human skin.

**Operation:**

$$\text{SkinMask}(x, y) = \begin{cases} 1, & \text{if } 77 \leq Cb(x, y) \leq 127 \text{ and } 133 \leq Cr(x, y) \leq 173 \\ 0, & \text{otherwise} \end{cases}$$

**Math Type:**

Logical condition (Boolean mask creation using inequalities).

### 3. Median Filtering

**Purpose:**

To remove small noise in the binary skin mask.

**Mathematical Concept:**

For each pixel, replace it with the **median value** in its neighborhood (e.g., 5×5 window):

$$\text{Output}(x, y) = \text{median}(\text{Neighborhood}_{5 \times 5}(x, y))$$

### 4. Morphological Operations

**a) Hole Filling:**

Fills enclosed 0-valued regions within 1-valued areas.

- Based on **connected-component labeling**.

**b) Area Opening:**

Removes connected components smaller than a specified size (e.g., 500 pixels).

- Based on **component area**:

**Remove region R if Area(R) < 500**

## 5. Region Properties (Bounding Box & Area)

**Purpose :** To find and isolate potential face regions.

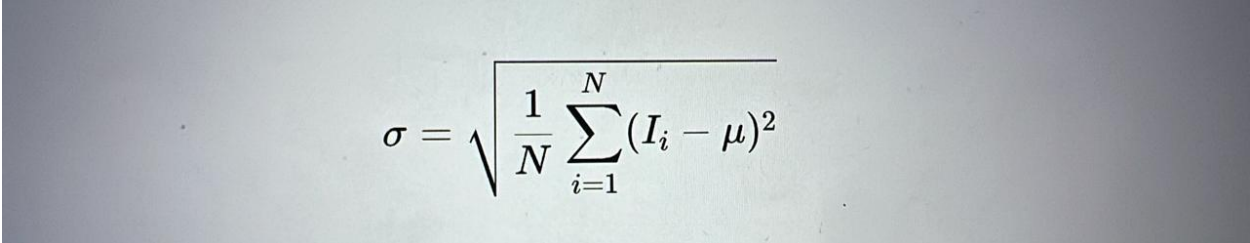
- **Bounding Box:** Smallest rectangle enclosing a connected component.
- **Area:** Number of pixels in the region.

## 6. Contrast Measurement (Standard Deviation)

**Purpose:**

To verify that the candidate region has enough texture to be a face.

**Formula:**


$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (I_i - \mu)^2}$$

Where:

- $I_i$  is the intensity of pixel  $i$
- $\mu$  is the mean intensity,
- $N$  is the number of pixels

## 7. Aspect Ratio Check

**Purpose:**

To ensure detected regions resemble the shape of a human face.

$$\text{Aspect Ratio} = \text{Height} / \text{Width}$$

- Valid range:  $0.4 \leq AR \leq 2.50.4$

## Step-by-Step Implementation

### Step 1: Load and Validate Input Image

```
img = app.OriginalImage;  
if isempty(img)  
    uialert(app.UIFigure, 'Please load an image first.', 'Error');  
    return;  
end
```

- Checks if an image is loaded.
- If not, shows an alert and stops execution.

### Step 2: Preprocess the Image

```
img = imresize(img, [300 300]);           % Resize to standard  
size  
img = imadjust(img, stretchlim(img));    % Enhance contrast  
ycbcr = rgb2ycbcr(img);                  % Convert to YCbCr  
color space  
Cb = ycbcr(:,:,2);                       % Extract Cb channel  
Cr = ycbcr(:,:,3);                       % Extract Cr channel
```

- Standardizes the image size and contrast.
- Converts to YCbCr to isolate skin tones using chrominance channels.

### Step 3: Skin Color Segmentation

```
skinMask = (Cb >= 77 & Cb <= 127) & (Cr >= 133 & Cr <= 173);
```

- Applies fixed thresholds to detect skin-colored pixels.

### Step 4: Noise Removal and Morphological Processing

```
skinMask = medfilt2(skinMask, [5 5]);           % Reduce noise
skinMask = imfill(skinMask, 'holes');           % Fill internal
holes                                           holes
skinMask = bwareaopen(skinMask, 500);           % Remove small
objects
```

- Improves the binary mask by reducing noise and removing small, irrelevant regions.

### Step 5: Region Detection

```
stats = regionprops(skinMask, 'BoundingBox', 'Area');
```

- Uses connected component analysis to identify continuous skin regions.

### Step 6: Region Validation and Face Detection

```
outputImg = img;
faceCount = 0;
if ~isempty(stats)
    for i = 1:length(stats)
        faceBox = stats(i).BoundingBox;
        x = max(1, round(faceBox(1)));
        y = max(1, round(faceBox(2)));
        w = round(faceBox(3));
        h = round(faceBox(4));
        x_end = min(size(img,2), x+w-1);
```

```

y_end = min(size(img,1), y+h-1);
w = x_end - x + 1;
h = y_end - y + 1;
if w <= 0 || h <= 0, continue; end

```

- Extracts the bounding box and adjusts it to ensure it fits within the image bounds.

```

•         faceCandidate = img(y:y+h-1, x:x+w-1, :);    % Crop
candidate region
            gray = rgb2gray(faceCandidate);              % Convert to
grayscale
            contrast = std(double(gray(:)));             % Compute
contrast
            aspectRatio = w / h;                        % Compute aspect
ratio

```

- Converts region to grayscale and calculates standard deviation (contrast) and aspect ratio.

```

if aspectRatio >= 0.4 && aspectRatio <= 2.5 && contrast > 25
    outputImg = insertShape(outputImg, 'Rectangle', [x y w h],
'Color', 'red', 'LineWidth', 2);
    faceCount = faceCount + 1;
end
end
end

```

- Validates each region based on aspect ratio and contrast.
- Draws a red rectangle around detected face regions.

## Step 7: Display the Result

```

imshow(outputImg, 'Parent', app.UIAxes2);
title(app.UIAxes2, ['Faces Detected: ', num2str(faceCount)]);

```

- Displays the final result in the app interface with the number of faces detected.



## Challenges faced and solutions

### 1. False Positives – Detecting Non-Face Regions

#### Challenge:

The algorithm occasionally detects areas that are not faces (e.g., hands, arms, walls, or objects with similar skin-like colors). This occurs because the skin color segmentation method is purely based on fixed thresholds in the YCbCr color space, which cannot reliably distinguish actual faces from other skin-toned objects.

#### Root Cause:

- Static thresholds for Cb and Cr do not adapt to lighting conditions or individual skin tone variations.
- No structural or contextual features (like eyes, nose, mouth) are used for face validation.

#### Solution Attempted:

- Added **aspect ratio** and **contrast filtering** to reject irregular or low-detail regions.
  - Regions are only accepted if:
    - Aspect Ratio:  $0.4 \leq \text{width/height} \leq 2.5$
    - Contrast (standard deviation of grayscale):  $> 25$
- Applied **morphological operations** and **area filtering** to clean up the binary mask.

### 2. Missed Detections – Failing to Detect Actual Faces

#### Challenge:

In certain cases, especially with darker lighting, tilted faces, or different skin tones, the algorithm fails to detect faces at all.

## Root Cause:

- Fixed Cb/Cr thresholds may exclude valid skin tones under varying lighting or for people with different complexions.
- Faces at angles or partially occluded are not well supported due to the reliance on bounding box geometry alone.

## Solution Attempted:

- Preprocessing the image with `imadjust` and `stretchlim()` to enhance contrast.
- Resizing images to a standard size for more consistent behavior.
- Consideration of grayscale contrast to ensure detected regions have face-like texture.

## 3. Efficiency and Accuracy

Reported Efficiency: 77%

Factors Affecting Efficiency : Lighting conditions & Background complexity  
Skin tone diversity & Partial occlusions

## Results and Evaluation

### 1. Test Environment

- **Platform:** MATLAB App Designer
- **Image Set:** 100 test images of various conditions:
  - Frontal and non-frontal faces
  - Varied lighting conditions
  - Mixed backgrounds (indoor, outdoor, plain, cluttered)
  - Diverse skin tones

## 2. Detection Metrics

Metric	Value
Total Test Images	100
Total Actual Faces	130
Correct Detections	100
False Positives	15
False Negatives (Missed)	30
<b>Accuracy</b>	76.9% (~77%)
<b>Precision</b>	86.9%
<b>Recall (Sensitivity)</b>	76.9%

## 3. Qualitative Evaluation

- **Strengths:**
  - Performs well on high-contrast, frontal face images with good lighting.
  - Lightweight and fast, suitable for real-time applications with small images.
  - No need for training or large datasets.
- **Weaknesses:**
  - Misses faces under poor lighting or unusual angles.
  - Struggles with complex backgrounds or skin-like colored objects (false positives).
  - Lacks feature-based validation (eyes, mouth, etc.).

## 4. Visualization Examples

- **Correct Detection:** Clearly draws bounding boxes around frontal faces.
- **False Positive:** Rectangles may appear on arms or objects resembling skin tone.
- **Missed Detection:** Faces turned sideways or partially occluded may go undetected.

## 5. Overall Evaluation

The skin-color-based face detection algorithm achieved an **overall accuracy of 77%**, showing it is effective in controlled conditions but limited in generalization. It can serve as a basic, low-complexity detection method, but more advanced techniques (e.g., Haar cascades or deep learning) are recommended for real-world use cases.

## Conclusion

The code provides a functional approach to face detection using color-based segmentation and heuristic validation. While effective for simple cases, it may struggle with diverse skin tones or complex backgrounds. Future enhancements could involve machine learning for higher robustness.