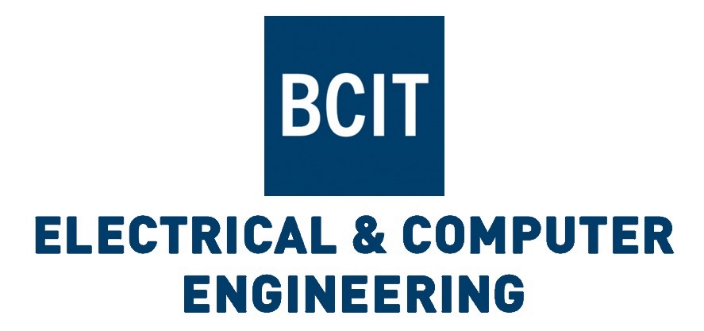
Digital Image & Video Processing

Final Project Report: Face Detection Project



Modified Date: 11/23/2025

For:

John Dian, Ph.D., P.Eng., SMIEEE

By:

Yizuo Chen

# Flowchart of the project

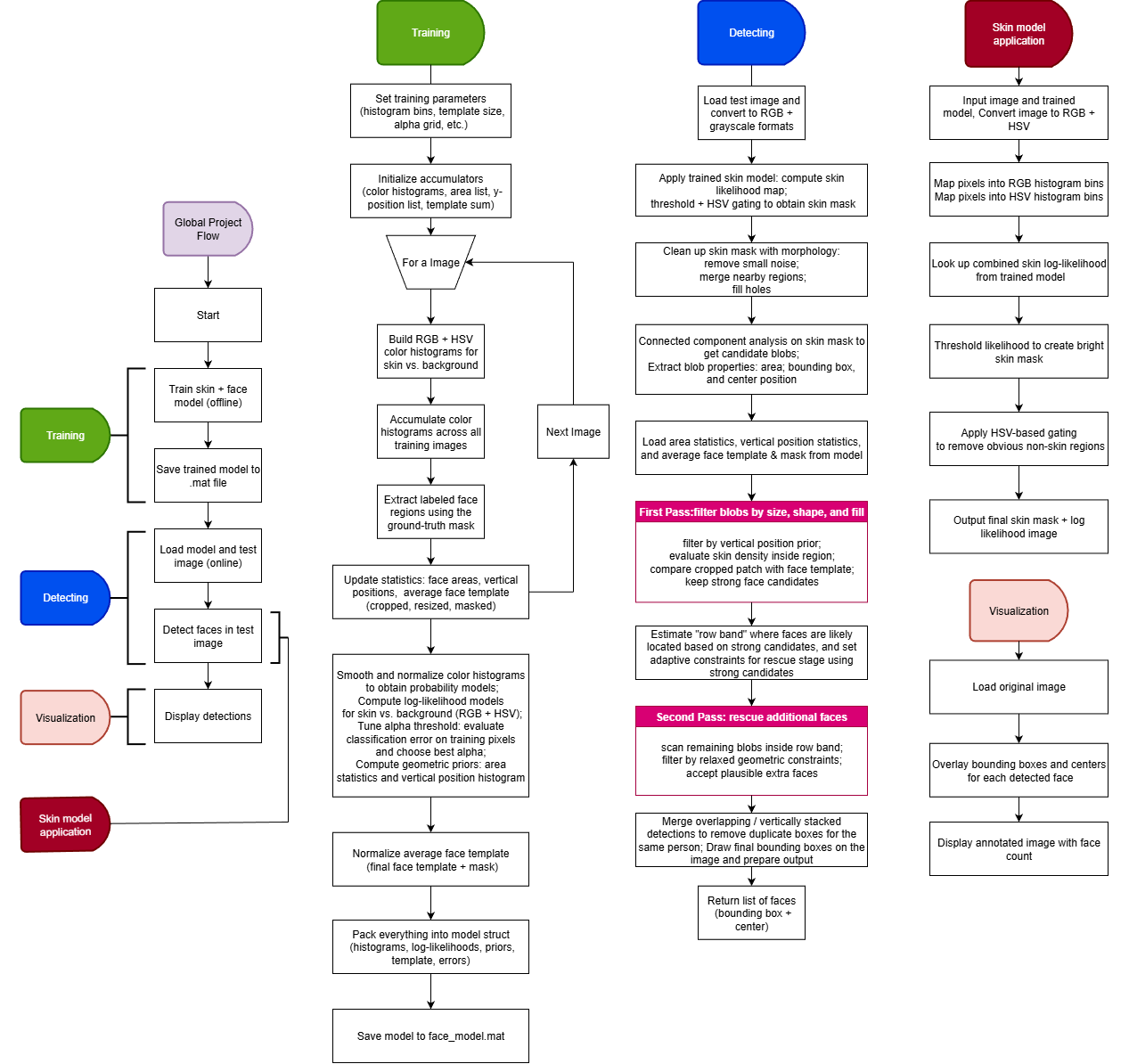


Figure Flowchart

## Global Project Flow

The overall system was divided into an offline training phase and an online detection phase, as summarized in Figure 1. During the training phase, a set of labeled face images and corresponding ground-truth masks were used to learn a probabilistic skin model, geometric priors, and an average face template. The resulting model was stored in a MATLAB .mat file for later reuse. During the detection phase, this pre-trained model was loaded and applied to unseen test images to locate faces. The detection pipeline produced a set of face bounding boxes and center locations, which were overlaid on the input image for visualization.

## Offline Training

In the training lane of the flowchart, the process began with the selection of training parameters such as histogram bin counts, template size, and the search grid for the decision threshold. Accumulators were then initialized to store RGB and HSV color histograms for skin and background, as well as lists of face areas, normalized vertical positions, and the running sum of face templates. For each training image, the corresponding ground-truth mask was used to separate skin from background. RGB and HSV color histograms were computed for both classes and accumulated across all images. Labeled face regions were extracted from the mask and used to update geometric statistics (area and vertical position) and to construct an average face template by cropping, resizing, masking with an ellipse, and summing over all faces. After all images were processed, the color histograms were smoothed and normalized to obtain probability models, a log-likelihood classifier was derived, and the alpha threshold was tuned by minimizing misclassification on the training pixels. Finally, area statistics, a vertical position histogram, and a normalized average face template were computed and packed into a single model structure, which was saved to face\_model.mat.

## Online Detection

The detecting lane in the flowchart describes the online stage. First, a test image was loaded and converted into RGB and grayscale formats. The pre-trained model was then applied to compute a skin-likelihood image and a corresponding binary skin mask. This mask underwent a sequence of morphological operations in order to remove small noise, merge nearby regions, and fill interior holes. Connected component analysis was then performed on the cleaned mask, and for each candidate region, basic properties such as area, bounding box, and centroid were extracted. Using the learned area statistics, vertical position priors, and the face template from the training phase, a first pass of filtering was performed to select high-confidence faces based on geometric criteria, skin density, and template correlation. The vertical positions of these strong detections were used to estimate a likely “row band” where faces were located, and a second pass was executed within this band to rescue additional plausible faces using relaxed constraints. Finally, overlapping or vertically stacked detections corresponding to the same subject were merged before the resulting bounding boxes and centers were returned.

## Skin Model Application and Visualization

The skin model application lane represents the subroutine that maps an input image to a binary skin mask using the trained color model. The input image was converted to RGB and HSV spaces, and each pixel was mapped into the corresponding histogram bins stored in the model. The pre-computed log-likelihoods in RGB and HSV were combined with the trained threshold to generate a per-pixel skin-likelihood image, which was then normalized for debugging display. A brightness-based threshold and an HSV-based gating step were applied to reject obvious non-skin regions and produce the final skin mask. In the visualization lane, the original test image was reloaded, the detected face bounding boxes and centers were drawn on top, and an annotated image with the total face count was displayed. This completed the end-to-end pipeline from offline model training to online face detection and visualization.

# Detection Results



Figure Detection Results

# Training Images and Refs

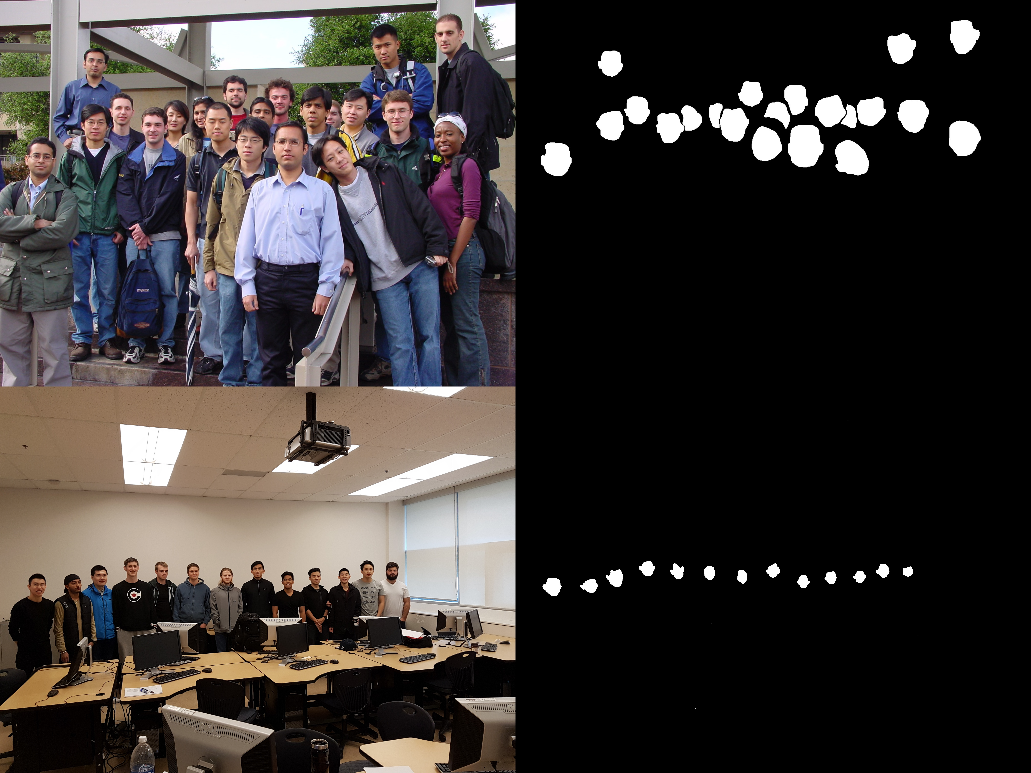


Figure Training Image Sets

Each training image (e.g., Training\_1.jpg) has a matching reference mask (e.g., ref1.png) where:

* white pixels = face skin
* black pixels = background

From these pairs, the program can separate skin pixels from non-skin pixels and build two large color histograms:

Face-skin histograms

* RGB skin histogram
* HSV skin histogram

Background histograms

* RGB non-skin histogram
* HSV non-skin histogram

# Source Code

## Offline Training

% //Yizuo Chen  
function model = train\_face\_models()  
% Train RGB/HSV skin likelihood model + face template  
% using Training\_1.jpg..Training\_11.jpg and ref masks.  
% This model is later used for:  
% - Skin detection (apply\_skin\_model)  
% - Template matching (strong detections)  
% - Basic size and position statistics  
  
 % --------------------------------------------------------------  
 % Training parameters  
 % --------------------------------------------------------------  
 numTrain = 11; % # training images  
 nbinsRGB = 32; % RGB histogram bin count  
 nbinsHSV = 32; % HSV histogram bin count  
 sigmaHist = 1.0; % Gaussian smoothing for histograms  
 tplH = 80; % template height (pixels)  
 tplW = 60; % template width  
 nBinsYpos = 20; % vertical-position histogram bins  
 alphaGrid = -2:0.2:4; % candidate alpha thresholds  
  
 % --------------------------------------------------------------  
 % Histogram accumulators (RGB + HSV)  
 % --------------------------------------------------------------  
 H\_face\_rgb = zeros(nbinsRGB, nbinsRGB, nbinsRGB);  
 H\_bg\_rgb = zeros(nbinsRGB, nbinsRGB, nbinsRGB);  
 H\_face\_hsv = zeros(nbinsHSV, nbinsHSV, nbinsHSV);  
 H\_bg\_hsv = zeros(nbinsHSV, nbinsHSV, nbinsHSV);  
  
 % Accumulated geometry + template data  
 areaList = [];  
 yList = [];  
 sumTemplate = zeros(tplH, tplW);  
 faceCount = 0;  
  
 % Elliptical mask to define valid pixels in template  
 [xx, yy] = meshgrid(linspace(-1,1,tplW), linspace(-1,1,tplH));  
 ellMask = (xx.^2/0.9^2 + yy.^2/1.0^2) <= 1;  
  
 % Store pixel indices for alpha tuning  
 allIdxRGB = {};  
 allIdxHSV = {};  
 allGT = {};  
  
 fprintf('Training on %d images (RGB + HSV)...\n', numTrain);  
  
 % --------------------------------------------------------------  
 % Loop through all training pairs (image + GT mask)  
 % --------------------------------------------------------------  
 for k = 1:numTrain  
 imgName = sprintf('Training\_%d.jpg', k);  
 refName = sprintf('ref%d.png', k);  
  
 I = im2uint8(imread(imgName));  
 GT = imread(refName);  
  
 if size(GT,3) > 1  
 GT = rgb2gray(GT);  
 end  
 GT = GT > 0; % ground-truth mask  
  
 % ----------------------------------------------------------  
 % Build RGB/HSV histograms for face vs. background  
 % ----------------------------------------------------------  
 [Hf\_rgb, Hb\_rgb, Hf\_hsv, Hb\_hsv, idxRGB, idxHSV, pixelGT] = ...  
 build\_color\_histograms(I, GT, nbinsRGB, nbinsHSV);  
  
 % Accumulate histograms  
 H\_face\_rgb = H\_face\_rgb + Hf\_rgb;  
 H\_bg\_rgb = H\_bg\_rgb + Hb\_rgb;  
 H\_face\_hsv = H\_face\_hsv + Hf\_hsv;  
 H\_bg\_hsv = H\_bg\_hsv + Hb\_hsv;  
  
 % Store pixel mappings for alpha optimization  
 allIdxRGB{end+1} = idxRGB;  
 allIdxHSV{end+1} = idxHSV;  
 allGT{end+1} = pixelGT;  
  
 % ----------------------------------------------------------  
 % Extract face components for:  
 % - size statistics  
 % - vertical position stats  
 % - template averaging  
 % ----------------------------------------------------------  
 L = bwlabel(GT);  
 stats = regionprops(L, 'Area', 'BoundingBox', 'Centroid');  
  
 for i = 1:numel(stats)  
 a = stats(i).Area;  
 bb = stats(i).BoundingBox;  
 c = stats(i).Centroid;  
  
 % Collect global statistics  
 areaList(end+1) = a;  
 yList(end+1) = c(2) / size(GT,1);  
  
 % Crop expanded region around face (includes chin/hair)  
 pad = 0.3;  
 x = bb(1); y = bb(2); w = bb(3); h = bb(4);  
 x1 = max(1, floor(x - pad\*w));  
 y1 = max(1, floor(y - pad\*h));  
 x2 = min(size(I,2), ceil(x + w + pad\*w));  
 y2 = min(size(I,1), ceil(y + h + pad\*h));  
 rect = [x1 y1 x2-x1+1 y2-y1+1];  
  
 patchRGB = imcrop(I, rect);  
 patchGray = rgb2gray(patchRGB);  
 patchGray = im2double(imresize(patchGray, [tplH tplW]));  
  
 % Apply elliptical mask around face region  
 patchGray(~ellMask) = 0;  
  
 % Accumulate template  
 sumTemplate = sumTemplate + patchGray;  
 faceCount = faceCount + 1;  
 end  
 end  
  
 if faceCount == 0  
 error('No faces found in training masks.');  
 end  
  
 % --------------------------------------------------------------  
 % Smooth histograms + convert to probability densities  
 % --------------------------------------------------------------  
 fprintf('Smoothing 3D histograms...\n');  
 G\_rgb = gaussian3d\_kernel(sigmaHist);  
 G\_hsv = gaussian3d\_kernel(sigmaHist);  
  
 H\_face\_rgb\_s = convn(H\_face\_rgb, G\_rgb, 'same');  
 H\_bg\_rgb\_s = convn(H\_bg\_rgb, G\_rgb, 'same');  
 H\_face\_hsv\_s = convn(H\_face\_hsv, G\_hsv, 'same');  
 H\_bg\_hsv\_s = convn(H\_bg\_hsv, G\_hsv, 'same');  
  
 % Normalize to proper PDFs, avoid zero probabilities  
 P\_face\_rgb = H\_face\_rgb\_s + 1;  
 P\_bg\_rgb = H\_bg\_rgb\_s + 1;  
 P\_face\_hsv = H\_face\_hsv\_s + 1;  
 P\_bg\_hsv = H\_bg\_hsv\_s + 1;  
  
 P\_face\_rgb = P\_face\_rgb / sum(P\_face\_rgb(:));  
 P\_bg\_rgb = P\_bg\_rgb / sum(P\_bg\_rgb(:));  
 P\_face\_hsv = P\_face\_hsv / sum(P\_face\_hsv(:));  
 P\_bg\_hsv = P\_bg\_hsv / sum(P\_bg\_hsv(:));  
  
 % Log-likelihood ratios for classification  
 logR\_rgb = log(P\_face\_rgb) - log(P\_bg\_rgb);  
 logR\_hsv = log(P\_face\_hsv) - log(P\_bg\_hsv);  
  
 logR\_rgb\_flat = logR\_rgb(:);  
 logR\_hsv\_flat = logR\_hsv(:);  
  
 % --------------------------------------------------------------  
 % Tune alpha parameter for optimal skin classification  
 % --------------------------------------------------------------  
 fprintf('Tuning alpha (RGB + HSV)...\n');  
 bestAlpha = alphaGrid(1);  
 bestErr = inf;  
 totalPix = 0;  
  
 for a = alphaGrid  
 errCount = 0;  
 pixCount = 0;  
  
 for k = 1:numTrain  
 idxR = allIdxRGB{k};  
 idxH = allIdxHSV{k};  
 gt = allGT{k};  
  
 % Combined RGB + HSV discriminator  
 scores = logR\_rgb\_flat(idxR) + logR\_hsv\_flat(idxH) + a;  
 pred = scores > 0;  
  
 errCount = errCount + sum(pred ~= gt);  
 pixCount = pixCount + numel(gt);  
 end  
  
 errRate = errCount / pixCount;  
 if errRate < bestErr  
 bestErr = errRate;  
 bestAlpha = a;  
 end  
 totalPix = pixCount;  
 end  
  
 fprintf('Best alpha = %.3f, training error = %.4f\n', ...  
 bestAlpha, bestErr);  
  
 % --------------------------------------------------------------  
 % Area + vertical statistics for prior information  
 % --------------------------------------------------------------  
 meanArea = mean(areaList);  
 stdArea = std(areaList);  
 minArea = min(areaList);  
  
 [yHist, yEdges] = histcounts(yList, nBinsYpos, ...  
 'BinLimits',[0 1], ...  
 'Normalization','probability');  
 yCenters = (yEdges(1:end-1) + yEdges(2:end)) / 2;  
  
 % --------------------------------------------------------------  
 % Build average face template (normalized)  
 % --------------------------------------------------------------  
 avgTpl = sumTemplate / faceCount;  
 avgTpl = avgTpl - min(avgTpl(:));  
 if max(avgTpl(:)) > 0  
 avgTpl = avgTpl / max(avgTpl(:));  
 end  
 avgTpl(~ellMask) = 0;  
  
 % --------------------------------------------------------------  
 % Pack all learned components into model struct  
 % --------------------------------------------------------------  
 model = struct();  
 model.nbinsRGB = nbinsRGB;  
 model.nbinsHSV = nbinsHSV;  
  
 model.P\_face\_rgb = P\_face\_rgb;  
 model.P\_bg\_rgb = P\_bg\_rgb;  
 model.P\_face\_hsv = P\_face\_hsv;  
 model.P\_bg\_hsv = P\_bg\_hsv;  
  
 model.logR\_rgb = logR\_rgb\_flat;  
 model.logR\_hsv = logR\_hsv\_flat;  
 model.alpha = bestAlpha;  
  
 model.tpl = avgTpl;  
 model.tplMask = ellMask;  
 model.tplSize = [tplH tplW];  
  
 model.areaStats = struct('mean', meanArea, 'std', stdArea, 'min', minArea);  
 model.yPosHist = struct('centers', yCenters, 'hist', yHist);  
  
 model.trainErr = bestErr;  
 model.totalPixels = totalPix;  
  
 fprintf('Training complete. Faces accumulated for template: %d\n', faceCount);  
end  
  
% -------------------------------------------------------------------------  
% Helper: build RGB/HSV histograms from a single training image  
% -------------------------------------------------------------------------  
function [H\_face\_rgb, H\_bg\_rgb, H\_face\_hsv, H\_bg\_hsv, idxRGB, idxHSV, pixelGT] = ...  
 build\_color\_histograms(I, GT, nbinsRGB, nbinsHSV)  
  
 I\_rgb = im2uint8(I);  
 I\_hsv = rgb2hsv(im2double(I\_rgb));  
 [H, W, ~] = size(I\_rgb);  
  
 % Flatten image channels  
 R = double(reshape(I\_rgb(:,:,1), [], 1));  
 G = double(reshape(I\_rgb(:,:,2), [], 1));  
 B = double(reshape(I\_rgb(:,:,3), [], 1));  
  
 Hh = reshape(I\_hsv(:,:,1), [], 1);  
 Ss = reshape(I\_hsv(:,:,2), [], 1);  
 Vv = reshape(I\_hsv(:,:,3), [], 1);  
  
 gt = reshape(GT, [], 1) > 0;  
  
 % --------------------- RGB bin computation ---------------------  
 binR = floor(R / 256 \* nbinsRGB) + 1;  
 binG = floor(G / 256 \* nbinsRGB) + 1;  
 binB = floor(B / 256 \* nbinsRGB) + 1;  
  
 binR = min(max(binR,1), nbinsRGB);  
 binG = min(max(binG,1), nbinsRGB);  
 binB = min(max(binB,1), nbinsRGB);  
  
 idxRGB = sub2ind([nbinsRGB nbinsRGB nbinsRGB], binR, binG, binB);  
  
 % --------------------- HSV bin computation ---------------------  
 binH = floor(Hh \* nbinsHSV) + 1;  
 binS = floor(Ss \* nbinsHSV) + 1;  
 binV = floor(Vv \* nbinsHSV) + 1;  
  
 binH = min(max(binH,1), nbinsHSV);  
 binS = min(max(binS,1), nbinsHSV);  
 binV = min(max(binV,1), nbinsHSV);  
  
 idxHSV = sub2ind([nbinsHSV nbinsHSV nbinsHSV], binH, binS, binV);  
  
 % --------------------- Build RGB histograms ---------------------  
 idxFaceRGB = idxRGB(gt);  
 H\_face\_rgb = accumarray(idxFaceRGB, 1, [nbinsRGB^3 1]);  
 H\_face\_rgb = reshape(H\_face\_rgb, [nbinsRGB nbinsRGB nbinsRGB]);  
  
 idxBgRGB = idxRGB(~gt);  
 H\_bg\_rgb = accumarray(idxBgRGB, 1, [nbinsRGB^3 1]);  
 H\_bg\_rgb = reshape(H\_bg\_rgb, [nbinsRGB nbinsRGB nbinsRGB]);  
  
 % --------------------- Build HSV histograms ---------------------  
 idxFaceHSV = idxHSV(gt);  
 H\_face\_hsv = accumarray(idxFaceHSV, 1, [nbinsHSV^3 1]);  
 H\_face\_hsv = reshape(H\_face\_hsv, [nbinsHSV nbinsHSV nbinsHSV]);  
  
 idxBgHSV = idxHSV(~gt);  
 H\_bg\_hsv = accumarray(idxBgHSV, 1, [nbinsHSV^3 1]);  
 H\_bg\_hsv = reshape(H\_bg\_hsv, [nbinsHSV nbinsHSV nbinsHSV]);  
  
 % Ground truth labels for alpha tuning  
 pixelGT = gt;  
end  
  
% -------------------------------------------------------------------------  
% Helper: build 3D Gaussian kernel  
% -------------------------------------------------------------------------  
function G = gaussian3d\_kernel(sigma)  
 r = ceil(3\*sigma);  
 [x, y, z] = ndgrid(-r:r, -r:r, -r:r);  
 G = exp(-(x.^2 + y.^2 + z.^2) / (2\*sigma^2));  
 G = G / sum(G(:));  
end  
  
% Train and save  
model = train\_face\_models();  
save("face\_model.mat","model");

Training on 11 images (RGB + HSV)...  
Smoothing 3D histograms...  
Tuning alpha (RGB + HSV)...  
Best alpha = -2.000, training error = 0.0347  
Training complete. Faces accumulated for template: 242

## Online Detection, Skin Model Application and Visualization

% //Yizuo Chen  
function faces = detect\_faces\_image(imgFile, model)  
% Detect faces using:  
% • RGB+HSV skin model (trained)  
% • Morphological cleanup  
% • Strong first-pass classification + template matching  
% • Automatic row-band estimation  
% • Rescue pass for missed faces  
% • Duplicate-merging (neck/logo removal)  
% Final output: list of face bounding boxes + centers.  
  
 I = imread(imgFile);  
 if size(I,3) == 1  
 I = repmat(I,[1 1 3]); % handle grayscale input  
 end  
 Igray = im2double(rgb2gray(I));  
 [H, W, ~] = size(I);  
  
 % -------------------------------------------------------------  
 % 1. Skin segmentation using trained RGB+HSV likelihood model  
 % -------------------------------------------------------------  
 % produces:  
 % (a) skinMask — thresholded bright skin-likelihood  
 % (b) logRimg — visualization of log-likelihood  
 [skinMask, logRimg] = apply\_skin\_model(I, model);  
 debug\_show(logRimg, 'DEBUG: logR image');  
  
 % -------------------------------------------------------------  
 % 1a. Morphology: merge patches, fill holes, remove noise  
 % -------------------------------------------------------------  
 % rely on training stats for minimal blob size  
 minPix = max(round(model.areaStats.min \* 0.20), 60);  
  
 skinMask = bwareaopen(skinMask, minPix); % remove tiny blobs  
 skinMask = imclose(skinMask, strel('disk', 6)); % merge forehead/cheek regions  
 skinMask = imfill(skinMask, 'holes'); % remove holes from eyes/mouth  
 skinMask = imopen(skinMask, strel('disk', 2)); % knock off thin noise  
 skinMask = bwareaopen(skinMask, minPix); % final size filter  
 debug\_show(skinMask, 'DEBUG: final skin mask');  
  
 % -------------------------------------------------------------  
 % 2. Connected component analysis  
 % -------------------------------------------------------------  
 % Extract geometric properties for all blobs.  
 L = bwlabel(skinMask);  
 stats = regionprops(L, 'Area', 'BoundingBox', 'Centroid');  
  
 % Debug display of all raw blobs  
 if ~isempty(stats)  
 imgBlobs = insertShape(I, 'Rectangle', cat(1,stats.BoundingBox), ...  
 'Color','cyan','LineWidth',2);  
 else  
 imgBlobs = I;  
 end  
 debug\_show(imgBlobs, 'DEBUG: connected components');  
  
 % Empty output if nothing detected  
 faces = struct('BoundingBox', {}, 'Center', {}, 'Score', {});  
 if isempty(stats)  
 return;  
 end  
  
 % -------------------------------------------------------------  
 % Load trained priors (area, vertical distribution, template)  
 % -------------------------------------------------------------  
 meanA = model.areaStats.mean;  
 minA = 0.35 \* meanA; % conservative lower bound  
 maxA = 2.5 \* meanA; % reject very large artifacts  
  
 yCenters = model.yPosHist.centers;  
 yHist = model.yPosHist.hist;  
  
 tpl = model.tpl;  
 tplMask = model.tplMask;  
 tplSize = model.tplSize;  
  
 strongIdx = [];  
  
 % -------------------------------------------------------------  
 % 3. FIRST PASS — strong face candidates  
 % Filters:  
 % • size, aspect ratio, fill ratio  
 % • vertical prior from training  
 % • skin intensity ratio  
 % • template correlation score  
 % These survive as high-confidence faces.  
 % -------------------------------------------------------------  
 for i = 1:numel(stats)  
 a = stats(i).Area;  
 bb = stats(i).BoundingBox;  
 c = stats(i).Centroid;  
  
 w = bb(3);  
 h = bb(4);  
 ratio = h / w;  
  
 % --- geometric constraints ---  
 if a < minA || a > maxA  
 continue;  
 end  
 if ratio < 0.8 || ratio > 2.0  
 continue;  
 end  
 fillRatio = a / (w\*h);  
 if fillRatio < 0.30 || fillRatio > 0.90  
 continue;  
 end  
  
 % --- vertical position prior ---  
 yNorm = c(2) / H;  
 if yNorm < 0.10 || yNorm > 0.70  
 continue;  
 end  
 [~, idxY] = min(abs(yCenters - yNorm));  
 if yHist(idxY) < 0.01  
 continue;  
 end  
  
 % --- template correlation check ---  
 pad = 0.25;  
 x = bb(1); y = bb(2);  
 x1 = max(1, floor(x - pad\*w));  
 y1 = max(1, floor(y - pad\*h));  
 x2 = min(W, ceil(x + w + pad\*w));  
 y2 = min(H, ceil(y + h + pad\*h));  
  
 % skin density in region  
 patchLogR = logRimg(y1:y2, x1:x2);  
 hardRatio = nnz(patchLogR > 0) / numel(patchLogR);  
 if hardRatio < 0.07  
 continue;  
 end  
  
 % correlation with template  
 patch = Igray(y1:y2, x1:x2);  
 patchR = imresize(patch, tplSize);  
 patchR(~tplMask) = 0;  
  
 if var(patchR(:)) < 0.003  
 continue;  
 end  
  
 tplVec = tpl(:) - mean(tpl(:));  
 patchVec = patchR(:) - mean(patchR(:));  
 score = (tplVec' \* patchVec) / (norm(tplVec)\*norm(patchVec) + eps);  
  
 if score < 0.40  
 continue;  
 end  
  
 % --- accept strong candidate ---  
 cx = x1 + (x2 - x1)/2;  
 cy = y1 + (y2 - y1)/2;  
  
 faces(end+1).BoundingBox = [x1 y1 (x2-x1+1) (y2-y1+1)];  
 faces(end).Center = [cx cy];  
 faces(end).Score = score;  
  
 strongIdx(end+1) = i;  
 end  
  
 nStrong = numel(faces);  
  
 % -------------------------------------------------------------  
 % 4. Determine likely vertical row-band from strong detections  
 % Used to rescue missed faces and eliminate table noise.  
 % -------------------------------------------------------------  
 if ~isempty(strongIdx)  
 yStrongNorm = arrayfun(@(k) stats(k).Centroid(2) / H, strongIdx);  
 margin = 0.05;  
 rowBandMinNorm = max(0, min(yStrongNorm) - margin);  
 rowBandMaxNorm = min(1, max(yStrongNorm) + margin);  
 else  
 % fallback if no strong detections  
 rowBandMinNorm = 0.10;  
 rowBandMaxNorm = 0.70;  
 end  
  
 % convert to pixel indices  
 rowBandMinPix = max(1, floor(rowBandMinNorm \* H));  
 rowBandMaxPix = min(H, ceil(rowBandMaxNorm \* H));  
  
 % debug visualization of allowed rescue region  
 bandMask = false(H,W);  
 bandMask(rowBandMinPix:rowBandMaxPix, :) = true;  
 debug\_show(bandMask & skinMask, ...  
 sprintf('DEBUG: face row band [%.2f, %.2f]', ...  
 rowBandMinNorm, rowBandMaxNorm));  
  
 % -------------------------------------------------------------  
 % 4b. Adaptive LOWER bounds for rescued faces  
 % Prevents tiny noise from entering second pass, but allows  
 % large faces (edges of group) to be kept.  
 % -------------------------------------------------------------  
 if ~isempty(strongIdx)  
 strongAreas = [stats(strongIdx).Area];  
 strongBB = cat(1, stats(strongIdx).BoundingBox);  
 strongHeights = strongBB(:,4);  
  
 medA = median(strongAreas);  
 medH = median(strongHeights);  
  
 rescueMinA = 0.35 \* medA; % minimal area  
 rescueMinH = 0.40 \* medH; % minimal height  
 else  
 rescueMinA = minA;  
 rescueMinH = 0;  
 end  
  
 % -------------------------------------------------------------  
 % 5. SECOND PASS — rescue plausible blobs inside row band  
 % Used to recover faces missed in first pass.  
 % -------------------------------------------------------------  
 allIdx = 1:numel(stats);  
 extraIdx = setdiff(allIdx, strongIdx);  
  
 for ii = extraIdx  
 bb = stats(ii).BoundingBox;  
 c = stats(ii).Centroid;  
 a = stats(ii).Area;  
  
 % must fall inside row band  
 yNorm = c(2) / H;  
 if yNorm < rowBandMinNorm || yNorm > rowBandMaxNorm  
 continue;  
 end  
  
 % adaptive geometric filters  
 w = bb(3);  
 h = bb(4);  
 ratio = h / w;  
 fillRatio = a / (w\*h);  
  
 if a < rescueMinA, continue; end  
 if h < rescueMinH, continue; end  
 if ratio < 0.6 || ratio > 2.5, continue; end  
 if fillRatio < 0.25 || fillRatio > 0.95, continue; end  
  
 % accept rescued detection  
 pad = 0.25;  
 x = bb(1); y = bb(2);  
 x1 = max(1, floor(x - pad\*w));  
 y1 = max(1, floor(y - pad\*h));  
 x2 = min(W, ceil(x + w + pad\*w));  
 y2 = min(H, ceil(y + h + pad\*h));  
  
 cx = x1 + (x2 - x1)/2;  
 cy = y1 + (y2 - y1)/2;  
  
 faces(end+1).BoundingBox = [x1 y1 (x2-x1+1) (y2-y1+1)];  
 faces(end).Center = [cx cy];  
 faces(end).Score = NaN;  
 end  
  
 % -------------------------------------------------------------  
 % 6. Merge vertically stacked duplicates (logo + chin + head)  
 % Removes doubled detections for same person.  
 % -------------------------------------------------------------  
 if numel(faces) > 1  
 keep = true(1, numel(faces));  
  
 for i = 1:numel(faces)  
 if ~keep(i), continue; end  
 b1 = faces(i).BoundingBox;  
 c1 = faces(i).Center;  
 w1 = b1(3); h1 = b1(4);  
  
 for j = i+1:numel(faces)  
 if ~keep(j), continue; end  
 b2 = faces(j).BoundingBox;  
 c2 = faces(j).Center;  
 w2 = b2(3); h2 = b2(4);  
  
 % horizontal similarity + vertical proximity  
 wAvg = 0.5\*(w1 + w2);  
 dx = abs(c1(1) - c2(1));  
 dy = abs(c1(2) - c2(2));  
  
 if dx < 0.45\*wAvg && dy < 1.4\*max(h1,h2)  
 % pick stronger score as base  
 s1 = faces(i).Score; if isnan(s1), s1 = -Inf; end  
 s2 = faces(j).Score; if isnan(s2), s2 = -Inf; end  
  
 if s2 > s1  
 base = j; other = i; bbBase = b2; bbOther = b1;  
 else  
 base = i; other = j; bbBase = b1; bbOther = b2;  
 end  
  
 % merge bounding boxes  
 xMin = min(bbBase(1), bbOther(1));  
 yMin = min(bbBase(2), bbOther(2));  
 xMax = max(bbBase(1)+bbBase(3), bbOther(1)+bbOther(3));  
 yMax = max(bbBase(2)+bbBase(4), bbOther(2)+bbOther(4));  
  
 bbNew = [xMin, yMin, xMax-xMin, yMax-yMin];  
  
 faces(base).BoundingBox = bbNew;  
 faces(base).Center = [bbNew(1)+bbNew(3)/2, ...  
 bbNew(2)+bbNew(4)/2];  
  
 keep(other) = false;  
 end  
 end  
 end  
  
 faces = faces(keep);  
 end  
  
 % -------------------------------------------------------------  
 % 7. Visualization of final detections  
 % -------------------------------------------------------------  
 outImg = I;  
 for k = 1:numel(faces)  
 outImg = insertShape(outImg,'Rectangle',faces(k).BoundingBox, ...  
 'Color','yellow','LineWidth',3);  
 end  
  
 debug\_show(outImg, 'DEBUG: final detected faces');  
end

function [mask, logRimg] = apply\_skin\_model(I, model)  
% Apply RGB+HSV log-likelihood model:  
% • compute skin probability using trained histograms  
% • convert to log-likelihood image  
% • apply bright-pixel threshold (~195–255)  
% • apply HSV gate to remove false positives  
  
 I\_rgb = im2uint8(I);  
 [H, W, ~] = size(I\_rgb);  
  
 nbinsRGB = model.nbinsRGB;  
 nbinsHSV = model.nbinsHSV;  
  
 % ----------------- RGB flattening + binning -----------------  
 R = double(reshape(I\_rgb(:,:,1), [], 1));  
 G = double(reshape(I\_rgb(:,:,2), [], 1));  
 B = double(reshape(I\_rgb(:,:,3), [], 1));  
  
 binR = floor(R / 256 \* nbinsRGB) + 1;  
 binG = floor(G / 256 \* nbinsRGB) + 1;  
 binB = floor(B / 256 \* nbinsRGB) + 1;  
 binR = min(max(binR,1), nbinsRGB);  
 binG = min(max(binG,1), nbinsRGB);  
 binB = min(max(binB,1), nbinsRGB);  
 idxRGB = sub2ind([nbinsRGB nbinsRGB nbinsRGB], binR, binG, binB);  
  
 % ----------------- HSV flattening + binning -----------------  
 I\_hsv = rgb2hsv(im2double(I\_rgb));  
 Hh = reshape(I\_hsv(:,:,1), [], 1);  
 Ss = reshape(I\_hsv(:,:,2), [], 1);  
 Vv = reshape(I\_hsv(:,:,3), [], 1);  
  
 binH = floor(Hh \* nbinsHSV) + 1;  
 binS = floor(Ss \* nbinsHSV) + 1;  
 binV = floor(Vv \* nbinsHSV) + 1;  
 binH = min(max(binH,1), nbinsHSV);  
 binS = min(max(binS,1), nbinsHSV);  
 binV = min(max(binV,1), nbinsHSV);  
  
 idxHSV = sub2ind([nbinsHSV nbinsHSV nbinsHSV], binH, binS, binV);  
  
 % ----------------- Combined log-likelihood -----------------  
 scores = model.logR\_rgb(idxRGB) + model.logR\_hsv(idxHSV) + model.alpha;  
 logRimg = reshape(scores, H, W);  
  
 % normalize to 0–255  
 logRnorm = mat2gray(logRimg);  
 logR8 = uint8(255 \* logRnorm);  
  
 % brightness thresholding (your measured skin limit)  
 brightMask = logR8 >= 195;  
  
 % HSV gate to reject bright non-skin (lights, shirts, monitors)  
 Simg = reshape(Ss, H, W);  
 Vimg = reshape(Vv, H, W);  
 hsvGate = (Simg > 0.08) & (Simg < 0.95) & ...  
 (Vimg > 0.05) & (Vimg < 0.98);  
  
 mask = brightMask & hsvGate;  
end

A group of people standing in a room

AI-generated content may be incorrect.

A group of people in a line

AI-generated content may be incorrect.

A group of people standing in a room with computers

AI-generated content may be incorrect.

A black screen with white dots

AI-generated content may be incorrect.

A group of people standing in a room with computers

AI-generated content may be incorrect.

function show\_faces(imgFile, faces)  
% Display detected faces with bounding boxes and center points.  
  
 I = imread(imgFile);  
 figure; imshow(I); hold on;  
  
 for i = 1:numel(faces)  
 bb = faces(i).BoundingBox;  
 rectangle('Position', bb, 'EdgeColor', 'y', 'LineWidth', 2);  
 plot(faces(i).Center(1), faces(i).Center(2), 'r+', ...  
 'MarkerSize', 8, 'LineWidth', 2);  
 end  
  
 title(sprintf('Detected faces: %d', numel(faces)));  
end

A group of people standing in a room with computers

AI-generated content may be incorrect.

function debug\_show(img, titleText)  
% Helper to show intermediate results with pause.  
  
 figure('Name', titleText);  
 imshow(img, []);  
 title(titleText,'Interpreter','none');  
 drawnow;  
 uiwait(msgbox('OK to continue','DEBUG','modal'));  
end

load face\_model.mat  
faces = detect\_faces\_image("person2.jpg", model);  
show\_faces("person2.jpg", faces);

## Function for making training images

% //Yizuo Chen  
% Clean ref8.png so that all non-white pixels become pure black.  
% Output: ref8\_clean.png (binary mask)  
  
function clean\_ref8()  
  
 % --- Load image ---  
 I = imread('ref4.png');  
  
 % Convert to grayscale if needed  
 if size(I,3) == 3  
 I = rgb2gray(I);  
 end  
  
 % --- Convert to binary mask ---  
 % White (255) → 255  
 % Anything else → 0  
  
 mask = I == 255; % logical mask of pure white pixels  
 out = uint8(mask) \* 255;  
  
 % --- Save cleaned version ---  
 imwrite(out, 'ref8\_clean.png');  
  
 % --- Optional debug display ---  
 figure;  
 subplot(1,2,1); imshow(I, []); title('Original ref8.png');  
 subplot(1,2,2); imshow(out,[]); title('Cleaned ref8 mask');  
  
 fprintf('Saved cleaned mask as ref8\_clean.png\n');  
end

Saved cleaned mask as ref8\_clean.png