

Facial Blemish Removal using Canny Edge Detection and Gaussian Blurring

**ECE533 Project
Fall 2005**

**By:
Joseph Marino
Gregory Yoblin**

Problem Statement:

The goal of this project was to design a product capable of automatically removing facial blemishes. It would do this by locating “blemishes” within an image via Canny Edge Detection. After locating the “blemishes,” the program would remove them and fill them in with a value that would render the removed “blemish” undetectable to the human eye.

Motivation:

Every person in the world has days where they don't look as good as they want to. Usually, those days occur at the least opportune moments: days when pictures are being taken. Currently, there are several methods of removing these “blemishes.” Some people simply take a patch of skin that looks similar to the region where there is a blemish and paste it over the blemish. Some programs allow the user to “erase” the blemish using some sort of blurring function of varying sizes. However these methods are very time consuming for large numbers of blemishes, and they require a very artistic eye to remove them properly. By creating a program that will work with larger areas to remove multiple blemishes at once, we hope to greatly reduce the amount of effort required to clean up images after they have been taken. Our program is not designed to be the only tool, rather it is very useful as a preprocessing step in the touch-up process.

Unfortunately cleaning up an image is a very aesthetic process, and there is no way to quantify what an ideal output would be. As such, our program will not produce ideal results all of the time. But it does a very good job at removing batches of blemishes in open areas.

Approach:

The project was implemented incrementally to allow for isolated testing and debugging, and to maintain a functional code set at all points during the project. This process began with determining a method for “removing” a blemish from a picture. Initial research had revealed two

potentially effective methods, linear interpolation via Laplace's equation and filling with the regional average value. Both would be smoothed with a Gaussian filter. The next step was selecting a set of edge detection and morphological operators to produce a blemish mask, a binary image with ones over the blemish pixels to be removed, from a manually selected sub-image. Those pieces were combined to form a program capable of removing blemishes from a manually selected region. Finally, the previous components would be modified for full automation in a block-processing or sliding-window type algorithm.

Work Performed:

This project was implemented in Matlab functions. Coding began with the `makeGaussianMask()` function, as either blemish removal method would require smoothing afterward. From there, informal experiments were performed on a sampling of target images where blemishes were manually selected and removed by the two methods described above in the approach. Filling with the average value was unanimously chosen as the more aesthetically pleasing option, and this decision allowed moving forward to write the blemish removal code.

The next obstacle was selecting a method of edge detection and the subsequent morphological operators to extract a blemish mask from a sub-image containing a blemish. This called for further experiments, which began with a comparison of Sobel, Prewitt, Canny, and Laplacian of Gaussian edge detection results when used on typical sub-images. Of these methods, Canny consistently gave “good” results, in which there would be a connected or nearly connected set of pixels around the boundary of the blemish along with some stray lines unrelated to the blemish. Utilizing contrast stretching as a preprocessing step helped improve the Canny edge detection results further.

Finding the right set of morphological operations was a guess and check type procedure. The best result was obtained using closing to connect any nearly connected blemish boundaries, followed by region filling on the background which when inverted gives the interior pixels of the blemish(s), and finished by opening to remove the stray lines. The results of these steps are shown in Figure 1. During this phase, manual sub-image selection was assumed which allowed the region filling algorithm to use the upper left corner as the starting pixel inside the region to fill. However, this choice would require reconsideration during the automation step. To combine the previously completed segments into a manual selection based blemish removal function, a sub-image selection function `getLittleOne()` was created and some glue code was added into the `mblem()` function allowing the processed sub-image to be reinserted into the original picture.

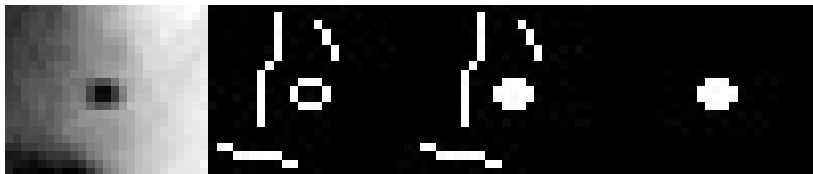


Figure 1: Contrast Stretching, Canny Edges, Closing & Filling, Opening

Steps toward automation began with creating a “no touch” masking option, so features like eyes, nostrils, and lips could be manually marked as non-blemish regions the algorithm would ignore. Next, several methods of block processing were compared, and the poor results showed a need for adjustment to the blemish detection and removal algorithm for use in an automated setting.

Results:

Blemish removal in manually selected sub-images was successfully implemented. By choosing a sub-image such that no blemishes touch the edge, the manual method works well and can be

repeated for removing multiple blemishes in spatially distant regions. Results of such a use can be seen in Figures 2 and 3 where blemishes on the cheek and chin area have been removed separately. However, when used to block process an entire picture the inability to determine blemish from natural feature greatly diminishes the resulting picture. Blemishes are removed along with other features meant to be left untouched.



Figure 2: Original



Figure 3: Corrected

Discussion:

After testing, many things became apparent. First and foremost, the program that we designed is not something that can be used as the sole means of removing “blemishes.” However, it can be used as a wonderful preprocessing step. Given more time, many of the limitations that stop this program from being what we envisioned it to be would be able to be fixed. We will discuss several of them here. To start out with, the program was supposed to be a fully automatic process. However, when we attempted to use this program in a block processing style on a full image we ended up with a checkerboard pattern on the image. Additionally, when we tried using half steps to alleviate the checkerboarding we ended up with more instead of less. At this point we determined that in order to get the sort of automatic processing that we were aiming for, we would need to design a program with the ability to determine when there was a “blemish” in the

frame of removal. Unfortunately, this would require some sort of classification system similar to a Support Vector Machine (SVM) or an ADABOOST algorithm.

Another issue that came up in the process of testing was how to fill the background of the image to create the blemish mask. Our original idea was to use just the top left corner. However when there was part of a blemish in that corner, the algorithm broke down. From there we decided that we needed to have multiple points as the initial point for filling, and then compare the results. We thought of using multiple random points in the image, but quickly realized that there was a good probability of getting all the points inside a “blemish.” This problem, though, has absolutely no good solution. This is due to the fact that the region can be selected by the user. There is no direct control over how the user selects the region. They may decide to select the blemish and have almost no background. This eliminates any pixel counting threshold. The user may decide to pick a region that has blemishes along the edges, which would invalidate the method that we are using currently. The only solution that we were able to propose would be to design a system with intelligence enough to determine if the point was inside of a “blemish” when it was selected as the seed for our flood filling. Since this is far beyond the scope of this class, we decided to simply make it a requirement that there be no blemishes along the edge of the region selected.

The other major problem that we could not solve given our time constraints was the problem of slowly developing blemishes. Our program works wonderfully for “blemishes” with a very definite demarcation. However, if someone has a mole that is the same color as the rest of their skin our program will have a major problem with it. Again, this is a problem that would be

easily solved by using an artificially intelligent agent in our program. It would be a simple matter to train the agent with examples of skin colored moles and other gradually changing “blemishes.” These problems, however, are all the result of designing a system that is supposed to make things visually appealing. There is no simple scientific explanation or equation to describe what people find appealing. Also, these issues should not take anything away from the success of this program. It works far better than we thought it would when we started out. The fact that it does have some issues only leaves the project open for further work by others in addition to ourselves. All in all, this project was a success. It has improved upon the blemish removal methods that exist in programs like PaintShop. It has also opened the door for further research.

```

% function [mask] = bfill(image, debug)
%   mask:  returns binary mask with 4-connected regions filled
%
%   image: binary image with region edges
%   debug: true turns on debugging output
%
%   Author: Joseph Marino (C) 2005
%   Class: ECE533
%
%   Notes: may fail when (1) pixel (1,1) is in a contained region, or (2)
%   when the subimage is fully divided by a region boundry.
%
%   ed:[12/12/05] renamed function from fill to bfill to avoid overloading
%               and enable debugging input. (Joseph Marino)

function [mask] = bfill(image, debug)
if (nargin < 2), debug = 0; end

i = logical(image);
n = ~image;
b = logical([0 1 0;
            1 1 1;
            0 1 0]);

[x,y] = size(i);
point = [1 1 x x; %4 corners
        1 y 1 y];

m = zeros(x,y,4);
for u = 1:4
    o = logical(zeros(x,y));
    ol = 0;
    o(point(1,u),point(2,u)) = 1; % assumes corners are points outside
    % any region
    % this assumption should hold for our use as the
    % while subimages are selected manually..may need
    % revision for automation

    % Region Filling
    while(any(any(o-ol)))
        ol = o;
        tmp = logical(imdilate(o,b));
        o = tmp & n;
    end
    m(:, :, u) = o;
end
i = (sum(m,3) > 2);

mask = logical(image) | ~i;

% display intermediate steps for debugging
if (debug)
    figure(1), subplot(1,3,1), imshow(image);
    subplot(1,3,2), imshow(i);
    subplot(1,3,3), imshow(mask);
end

```

```

%This function should only be used on BW images (*.pgm)
%[image2 rect] = getLittleOne(img)
%Code adapted from Saad Ali
%(University of Central Florida Computer Vision Lab)
%2005
% Author: Greg Yoblin (C) 2005
% Class: ECE533
%
% ed:[12/7/05] added rect to return original subimage coordinates and
% changed input to image matrix instead of filename (Joseph Marino)
%
% ed:[12/7/05] added bounds checking on subimage coordinates (Joseph Marino)
%
% ed:[12/9/05] added code to adjust for color images (Greg Yoblin)

function [image2, rect] = getLittleOne(img)

figure(5), subplot(1,1,1), imshow(img);
[w h]=size(img);
r(1).val=getrect;close(5);
x=round(r(1).val(1));
y=round(r(1).val(2));
width=round(r(1).val(3));
height=round(r(1).val(4));
rect = [y y+height;x x+width];

% bounds checks
rect(:,1) = (rect(:,1) >= [1;1]) .* rect(:,1) + (rect(:,1) < [1;1]);
rect(:,2) = (rect(:,2) <= size(img(:, :, 1))') .* rect(:,2) + (rect(:,2) >
size(img(:, :, 1))') ...
.* size(img(:, :, 1))';

image2=img(rect(1,1):rect(1,2), rect(2,1):rect(2,2), :);

```

```
%This will make a 2-dimensional Gaussian Mask for Smoothing
%mask = makeGaussianMask(sigma)
% Author: Greg Yoblin (C) 2005
% Class: ECE533
function mask = makeGaussianMask(sigma)

loLim=-ceil(3*sigma);
hiLim=ceil(3*sigma);

x=loLim:hiLim;
y=loLim:hiLim;

for i=1:(hiLim-loLim)+1
    for j=1:(hiLim-loLim)+1
        mask(i,j)=(1/(2*pi*sigma^2))*exp(-(x(i)^2+y(j)^2)/(2*sigma^2));
    end
end
end
```

```

function image2 = makeMaskings(image)
image2=zeros(size(image));

%0 - finish mask
%1 - continue to get mask
%2 - remove last piece

userInput = 1;
rect=[0 0 0 0];
while userInput ~= 0
    if userInput == 1
        [img rect] = getLittleOne(image);

        %Put individual pixels into the mask
        for i=rect(1,1):rect(1,2)
            for j=rect(2,1):rect(2,2)
                if sum(image2(i,j,:)) == 0
                    image2(i,j,:)=image(i,j,:);
                end
            end
        end
    else

        %remove pixels that were in the last iteration
        for i=rect(1,1):rect(1,2)
            for j=rect(2,1):rect(2,2)
                image2(i,j,:)=0;
            end
        end
    end
end

masked= image2 == 0;
masked = uint8(double(image).*masked);

figure(12);
subplot(121);imshow(image);subplot(122);imshow(masked);

    userInput = input('0 - finish the mask\n1 - get more for the mask\n2 -
remove the last piece\nSelection: ');
end

image2 = ~image2;

```

```

% function [picture] = smallblem(image, notouch, debug)
%   picture:  returns matrix with blemishes "corrected"
%
%   image: [x,y] grayscale image of type uint8
%
% Authors: Joseph Marino, Greg Yoblin (C) 2005
% Class: ECE533
%
% ed:[12/10/05] Implemented notouch mask (Joseph Marino)
% ed:[12/12/05] Added debugging switch, fixed some casting errors (Joseph
Marino)

function [picture] = smallblem(image, notouch, debug)
if (nargin < 2)
    notouch = ones(size(image));
    debug = 0;
elseif (nargin < 3)
    debug = 0;
end

layers = size(image,3);
if (layers>1)
    i = rgb2gray(image);
else
    i = image;
end

i = double(i);
i = i - min(min(i));
i = uint8(i * (255/max(max(i))));

e1 = edge(i,'canny');
e1 = imclose(e1,strel('disk',1));
%e3 = imopen(e2,strel('disk',1));
%mask = e3;
mask = bfill(e1,debug);
mask = imopen(mask, strel('disk',1));
% notouch masking
mask = mask & notouch;

% Show intermediate steps for debugging
if (debug)
    figure(1)
    subplot(2,2,1),imshow(i); title('Range Expanded Image');
    subplot(2,2,2),imshow(e1); title('Closed Canny Edges');
    subplot(2,2,3),imshow(mask); title('Mask');
end

i = image;
mask = repmat(mask, [1 1 layers]);
i2 = uint8(double(i) - double(mask*255));
if (debug)
    subplot(2,2,4),imshow(i2); title('Comparison of Mask and Original Image');
end

% Calculate Local Average Color
count = repmat(size(i,1)*size(i,2),[1 1 layers]) - sum(sum(mask));

```

```

avg = round(sum(sum(i2))./count);
i = uint8(double(i2) + (double(mask) .* repmat(avg, [size(i,1) size(i,2)
1])));

z = makeGaussianmask(1);

for u = 1:layers
    t(:, :, u) = uint8(conv2(double(i(:, :, u)), z, 'valid'));
end

x = (size(i,1) - size(t,1)) / 2;
y = (size(i,2) - size(t,2)) / 2;

picture = i;
picture(x+1:size(picture,1)-x,y+1:size(picture,2)-y,:) = t(:, :, :);

% Comparison of original subimage to corrected subimage
if (debug)
    figure(2), subplot(1,2,1), imshow(image);title('Original');
    subplot(1,2,2), imshow(picture);title('Corrected');
end

```

```

% function [picture] = mblem(filename, debug)
%   picture:  returns a corrected picture, by allowing manual selection
%             of a blemished region, and correcting the blemish.
%
%   filename: filename of full image to select region from.
%   debug:    true turns on debugging output
%
%   Author: Joseph Marino (C) 2005
%   Class: ECE533
%
%   ed:[12/9/05] added code for full color and BW images (Greg Yoblin)
%   ed:[12/10/05] finished notouch masking code (Joseph Marino)

function [ picture ] = mblem( filename, debug )
if (nargin < 2), debug = 0;end

image = imread(filename);
%if (size(image,3)>1), image = rgb2gray(image);end

userInfo = 1;

picture = image;

in = input('Would you like to mask regions to prevent removal? (y/n) ','s');
if upper(in) == 'Y'
    mask = makeMaskings(image);
else
    mask = ones(size(image));
end
while (userInfo ~= 0)

    disp('Use your mouse to select a region for blemish removal.')
    [little rect] = getLittleOne(picture);

    newlittle = smallblem(little, mask(rect(1,1):rect(1,2),...
        rect(2,1):rect(2,2)), debug);
    picture(rect(1,1):rect(1,2), rect(2,1):rect(2,2),:)=newlittle(:,:,:);

    % Compare original to corrected for debugging
    figure(3), subplot(1,2,1), imshow(image);title('Original')
    subplot(1,2,2), imshow(picture);title('Corrected');

    userInfo = input('Press 1 for more regions, or Press 0 to end: ');
end

```

Breakdown of Work:

	Greg Yoblin	Joseph Marino
Project Idea	100%	0%
Resource Gathering	50%	50%
Project Proposal	50%	50%
MATLAB Coding	40%	60%
Testing	35%	65%
PowerPoint Presentation	80%	20%
Oral Presentation	50%	50%
Written Report	50%	50%
Overall	50%	50%

Approval Signatures:

Greg Yoblin

Date

Joseph Marino

Date