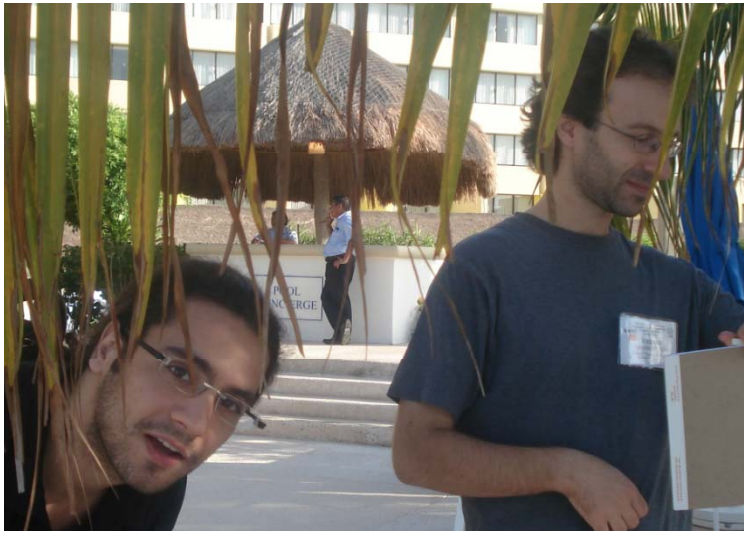
Template-Based Face Detection

CSE 6367 – Computer Vision
Vassilis Athitsos
University of Texas at Arlington

Face Detection

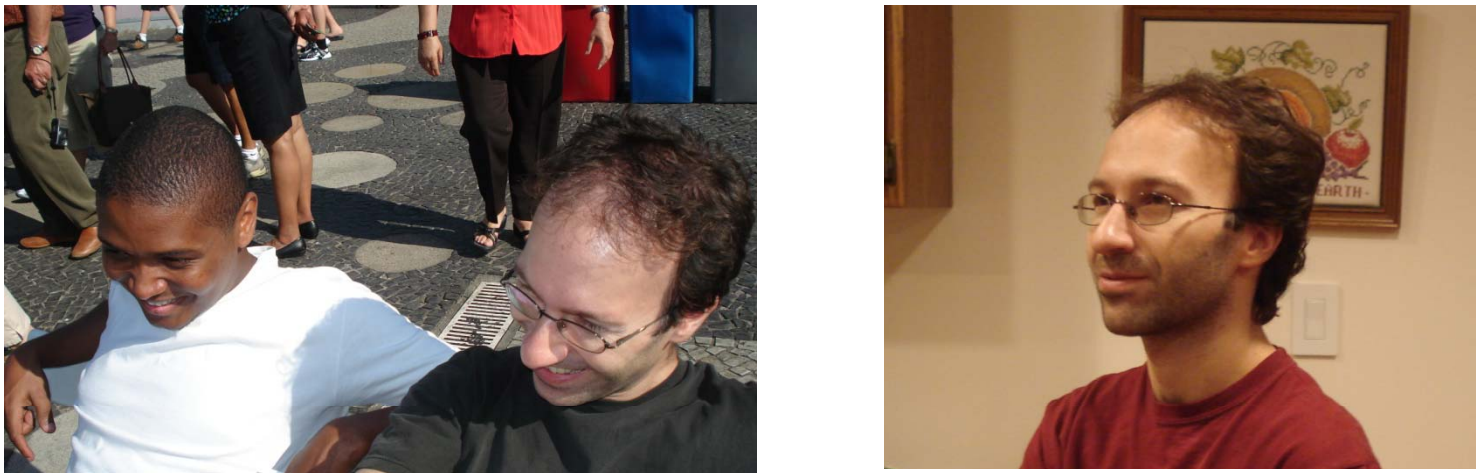
- We will make a few assumptions, to simplify the problem:
  - The face is fully visible.



Example cases where the face is not fully visible

Face Detection

- We will make a few assumptions, to simplify the problem:
  - The face is fully visible.
  - We see a *frontal* view of the face, i.e., the face is facing towards the camera.



Example cases where the view of the face is not frontal

Face Detection

- We will make a few assumptions, to simplify the problem:
  - The face is fully visible.
  - We see a *frontal* view of the face, i.e., the face is facing towards the camera.
  - The face is more or less in an upright orientation.

Example of a face not in an upright orientation

What is a Template?

- A template is an example of how an object looks.
- What example would be appropriate if we are looking for a face?

What is a Template?

- A template is an example of how an object looks.
- What example would be appropriate if we are looking for a face?
  - An *average* face.

Computing an Average Face

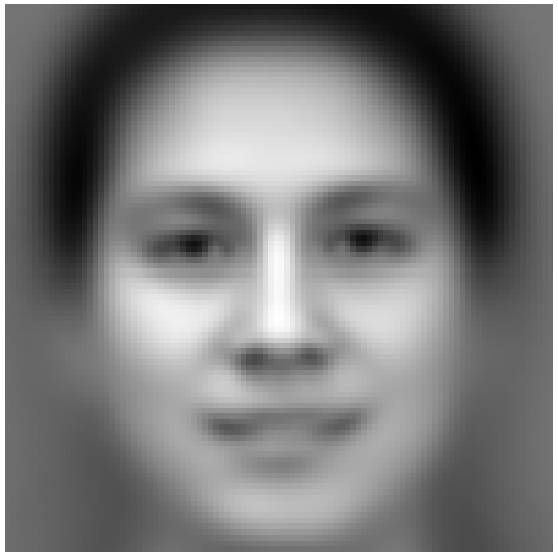
- We need *aligned* face images:
- In this case, *aligned* means:
  - same size
  - significant features (eyes, nose, mouth), to the degree possible, are in similar pixel locations.

an example set of aligned face images

Computing an Average Face

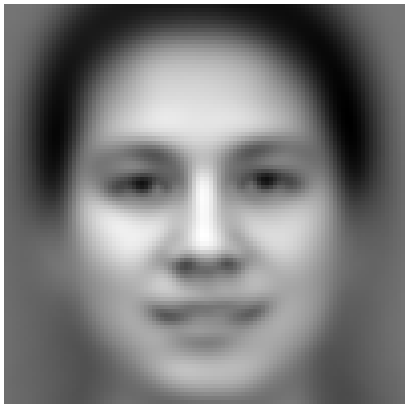
```
% Note that filenames is a cell variable (look it up!).
% Using cells is a good way to define a set of strings.
filenames = {
'4846d101.bmp'
'4848d101.bmp'
'4851d101.bmp'
'4853d101.bmp'
'4854d101.bmp'
};

number = prod(size(filenames));
image_vertical = 100;
image_horizontal = 100;
total = zeros(image_vertical, image_horizontal);

for index = 1: number
    image = read_gray(filenames{index});
    total = total + image;
    disp(index);
end

average_face = total / number;
```

Result: Average Face



Defining a Face Template

- Keep the parts of the average face that are most likely to be present in all faces:
  - Exclude background.
  - Exclude forehead (highly variable appearance, due to hair).
  - Exclude lower chin.



average face

face template

Using a Template to Find Faces

- How can we use a face template to perform face detection in an image?

Finding Matches for the Template

- How can we find good matches for a given template?

Using Normalized Correlation

```
function result = vector_normalized_correlation(v1, v2)

% function result = vector_normalized_correlation(v1, v2)

centered_v1 = v1 - mean(v1);
centered_v2 = v2 - mean(v2);
norm1 = norm(centered_v1);
norm2 = norm(centered_v2);

result = centered_v1 .* centered_v2 / (norm1 * norm2);
```

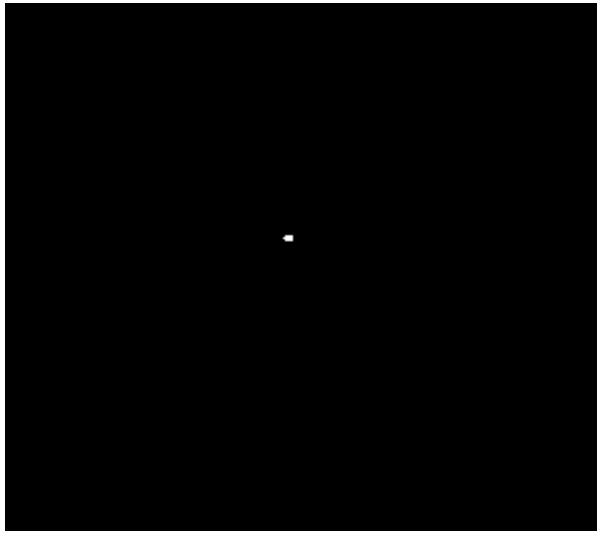
- Function normxcorr2(template, image) returns a matrix of normalized correlation scores between the template and each template-sized subwindow of the image.

Invoking normxcorr2

```
photo = read_gray('vassilis1g.bmp');
result = normxcorr2(face_filter, photo);
```

photo

result



result > 0.6

- It found the face!
- The result of normxcorr2 has larger size than the input.
- The face in photo matched the scale of the template.

normalized_correlation

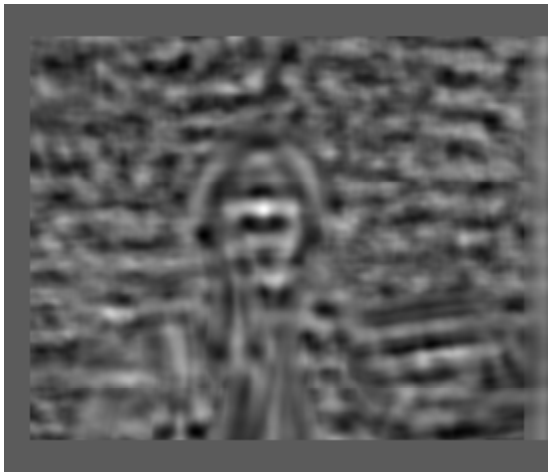
- A wrapper around normxcorr2.
- The result has the same size as the image.
- Border values are zero.

```
function result = normalized_correlation(image, template)

% function result = normalized_correlation(image, template)
%
% Returns a matrix containing normalized correlation results between
% template and all template-sized subwindows of image.

[image_rows, image_columns] = size( image);
[template_rows, template_columns] = size(template);
row_start = floor(template_rows / 2) + 1;
row_end = row_start + image_rows - 1;
col_start = floor(template_columns / 2) + 1;
col_end = col_start + image_columns - 1;

result = normxcorr2(template, image);
[result_rows, result_columns] = size(result);
result(1:template_rows, :) = 0;
result((result_rows-template_rows+1):result_rows, :) = 0;
result(:, 1:template_columns) = 0;
result(:, (result_columns-template_columns+1):result_rows, :) = 0;


result = result(row_start:row_end, col_start:col_end);
```

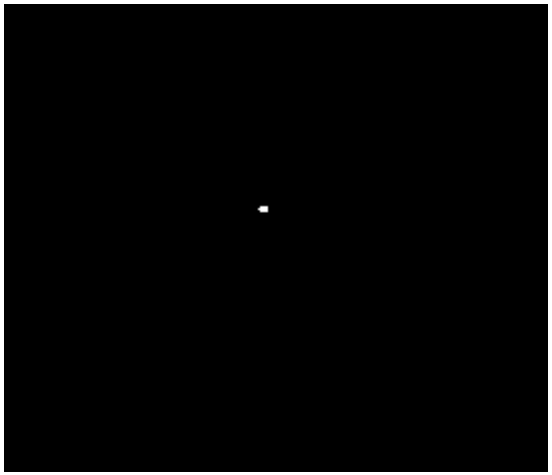
Invoking normalized_correlation

```
photo = read_gray('vassilis1g.bmp');
result = normalized_correlation(photo, face_filter);
```

photo



result

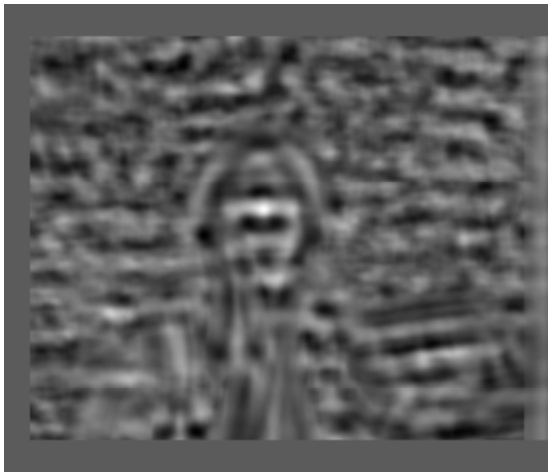


result > 0.6

A Trick for Visualizing Results

```
photo = read_gray('vassilis1g.bmp');
result = normalized_correlation(photo, face_filter);
visualization = max((result > 0.6)*255, photo * 0.7);
```

photo



result

visualization

Problem: Different Scales

```
photo = read_gray('vassilis1e.bmp');
result = normalized_correlation(photo, face_filter);
visualization = max((result > 0.35)*255, photo * 0.7);
```

photo

result

visualization

Solution: Multiscale Search

```
function ...
[result, max_scales] = multiscale_correlation(image, template, scales)

% function result = multiscale_correlation(image, template, scales)
%
% for each pixel, search over the specified scales, and record:
% - in result, the max normalized correlation score for that pixel
%   over all scales
% - in max_scales, the scale that gave the max score
```

Solution: Multiscale Search

```
function ...
[result, max_scales] = multiscale_correlation(image, template, scales)

result = ones(size(image)) * -10;
max_scales = ones(size(image)) * -10;

for scale = scales;
    % for efficiency, we either downsize the image, or the template,
    % depending on the current scale
    if scale >= 1
        scaled_image = imresize(image, 1/scale, 'bilinear');
        temp_result = normalized_correlation(scaled_image, template);
        temp_result = imresize(temp_result, size(image), 'bilinear');
    else
        scaled_image = image;
        scaled_template = imresize(template, scale, 'bilinear');
        temp_result = normalized_correlation(image, scaled_template);
    end

    higher_maxes = (temp_result > result);
    max_scales(higher_maxes) = scale;
    result(higher_maxes) = temp_result(higher_maxes);
end
```

Results of multiscale_correlation

```
photo = read_gray('vassilis1e.bmp');
scales = 0.5:0.1:3;
[result2, max_scales] = ...
        multiscale_correlation(photo, face_filter, scales);
visualization2 = max((result2 > 0.6)*255, photo * 0.7);
```

photo

result2

visualization2

Handling Rotations

```
load face_filter;
photo = read_gray('vassilis2b.bmp');
[result, boxes] = ...
     template_detector_demo(photo, face_filter, 0.5:0.1:3., 0, 1);
```

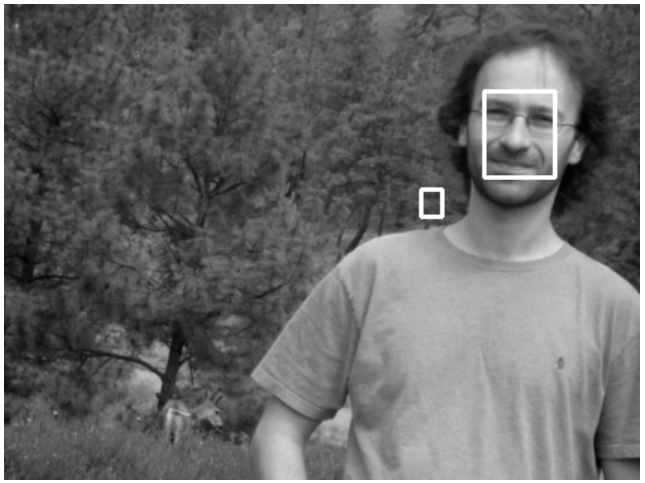
photo

result

Handling Rotations

```
load face_filter;
photo = read_gray('vassilis2b.bmp');
[resultb, boxes] = ...
     template_detector_demo(photo, face_filter, 0.5:0.1:3., 0, 2);
```

photo



resultb

Code for Template Search

- Useful functions:
  - template_search
  - find_template
  - template_detector_demo

```
function [max_responses, max_scales, max_rotations] = ...
    template_search(image, template, scales, rotations, result_number)

% function [result, max_scales, max_rotations] = ...
%    template_search(image, template, scales, rotations, result_number)
%
% for each pixel, search over the specified scales and rotations,
% and record:
% - in result, the max normalized correlation score for that pixel
%   over all scales
% - in max_scales, the scale that gave the max score
% - in max_rotations, the rotation that gave the max score
%
% clockwise rotations are positive, counterclockwise rotations are
% negative.
% rotations are specified in degrees
```

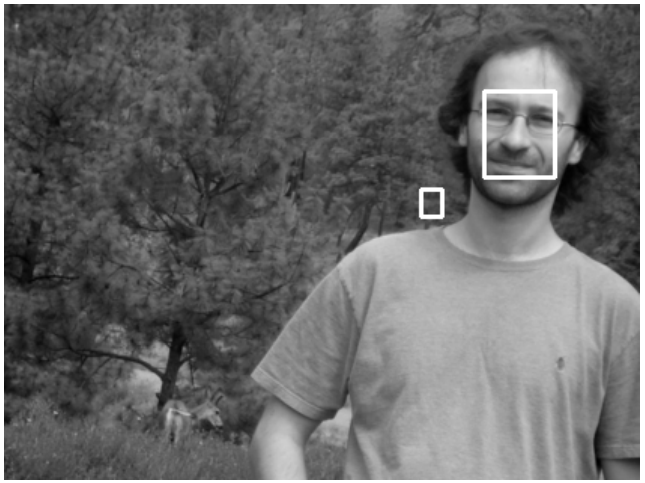
```
function [max_responses, max_scales, max_rotations] = ...
    template_search(image, template, scales, rotations, result_number)



max_responses = ones(size(image)) * -10;
max_scales = zeros(size(image));
max_rotations = zeros(size(image));

for rotation = rotations
    rotated = imrotate(image, -rotation, 'bilinear', 'crop');
    [responses, temp_max_scales] = ...
                 multiscale_correlation(rotated, template, scales);
    responses = imrotate(responses, rotation, 'nearest', 'crop');
    temp_max_scales = imrotate(temp_max_scales, rotation, ...
                                'nearest', 'crop');
    higher_maxes = (responses > max_responses);
    max_responses(higher_maxes) = responses(higher_maxes);
    max_scales(higher_maxes) = temp_max_scales(higher_maxes);
    max_rotations(higher_maxes) = rotation;
end
```

```
function [result, boxes] =  ...
      template_detector_demo(image, template, ...
                             scales, rotations, result_number)

% function [result, boxes] =
%       template_detector_demo(image, template, ...
%                              scales, rotations, result_number)
%
% returns an image that is a copy of the input image, with
% the bounding boxes drawn for each of the best matches for
% the template in the image, after searching all specified
% scales and rotations.

boxes = find_template(image, template, scales, ...
                      rotations, result_number);
result = image;

for number = 1:result_number
    result = draw_rectangle1(result, boxes(number, 1), ...
                             boxes(number, 2), ...
                             boxes(number, 3), boxes(number, 4));
end
```

Handling Rotations

```
load face_filter;
photo = read_gray('vassilis2b.bmp');
[resultb, boxes] = ...
     template_detector_demo(photo, face_filter, 0.5:0.1:3., 0, 2);
```

photo

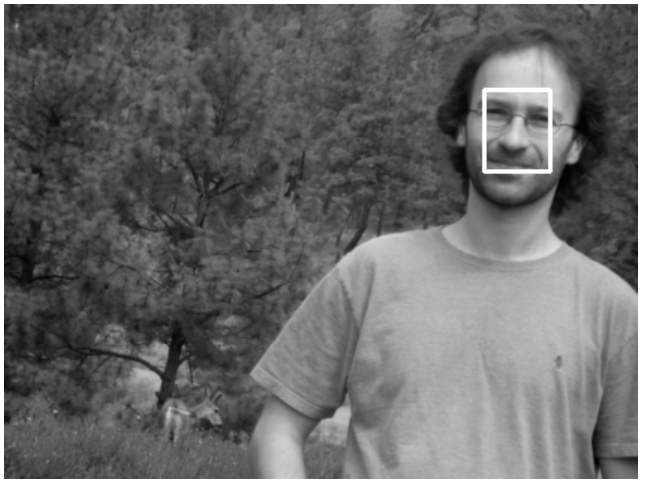


resultb

Handling Rotations

```
load face_filter;
photo = read_gray('vassilis2b.bmp');
scales = 0.5:0.1:3;
rotations = -10:5:10;
[result2, boxes] = template_detector_demo(photo, face_filter, ...,
                                          scales, rotations, 1);
```

photo



result2