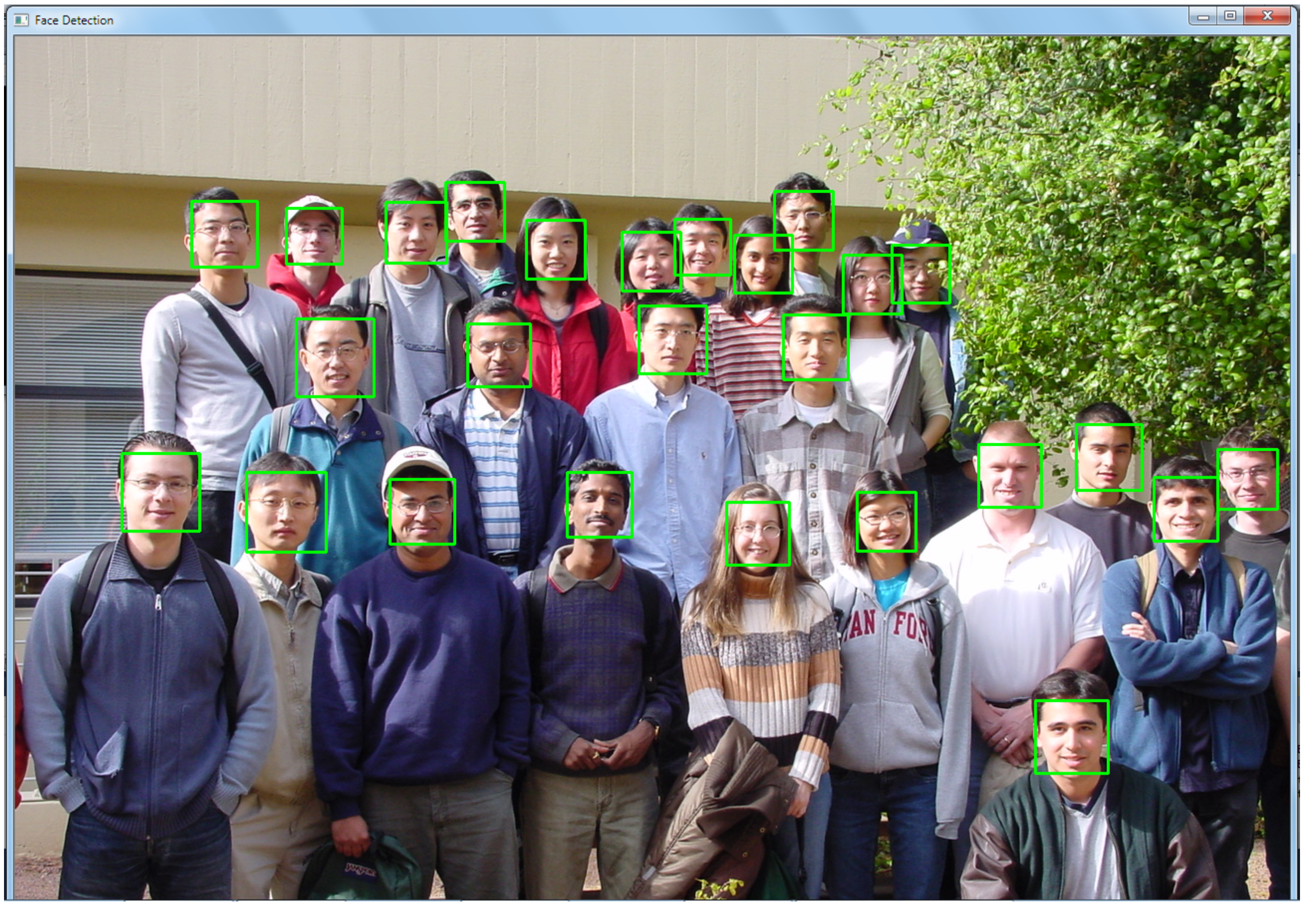
Face and Eye Detection Using OpenCV: Step by Step

Mahdi Rezaei
Department of Computer Science, the University of Auckland
m.rezaei@auckland.ac.nz
www.MahdiRezaei.com

Abstract:
The tutorial provides a detailed discussion on OpenCV library for face & eye detection based on Haar-like classifiers, which is the most common technique in computer-vision for face and eye detection. This paper is designed as part of course 775- Advanced multimedia imaging; however, it is easy to understand for any CS student who is interested in the topic.

The following instructions are in accordance with the provided source-code for face detection, and the tutorial "Creating a Cascade of Haar-Like Classifiers: Step by Step" both available from www.cs.auckland.ac.nz/~m.rezaei/Downloads.html
Further technical and academic informations could be found via our published papers and articles [1-8] available via www.cs.auckland.ac.nz/~m.rezaei/Publications.html.

Final output of this tutorial will be *face detections* as figure below:



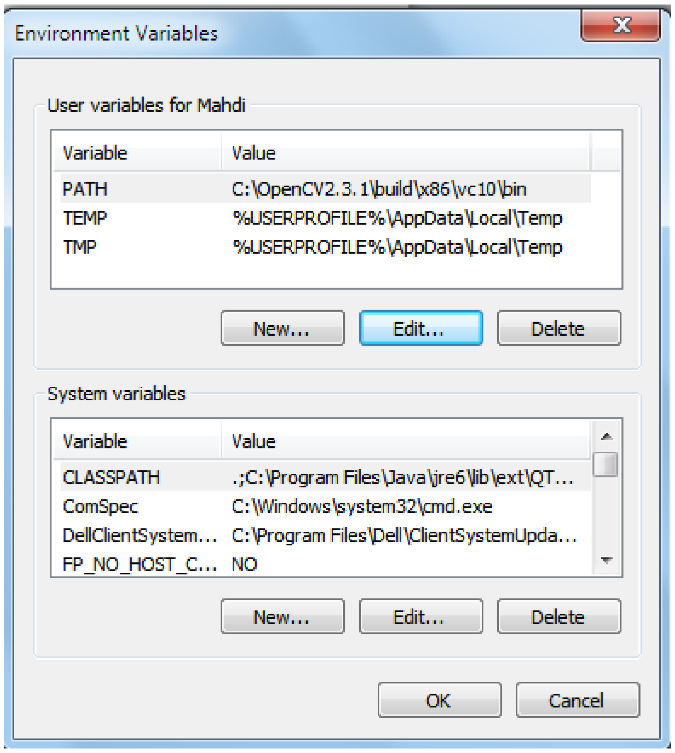
1. Prerequisites and Settings:

- *"Visual Studio 2010"* and "*OpenCV 2.3.1"* or higher
  Default Installation path:
  "C:\Program Files\Microsoft Visual Studio 10.0\"
  "C:\OpenCV2.3.1\"
- *"OpenCV Binary folder"* should be defined in Windows environmental path

Check your openCV to make sure if where the OpenCV *'bin'* folder is located. Then add it to Widows path under the "system variables" as follows:

1. Right-click on **My Computer** icon and then click on Properties.
2. From Advanced systems settings, click on **Environment Variables** button.
3. Finally, in the Environment Variables window (as shown below), highlight **Path** in *User Variables* section, and click on **Edit button**. Add or modify the path lines with the *bin* path or other paths that you want the computer to access. If you need to use multiple directories, separate them by semicolons as below example.

C:\Program Files; C:\OpenCV2.3.1\build\x86\vc10\bin


Environment Variables

User variables for Mahdi

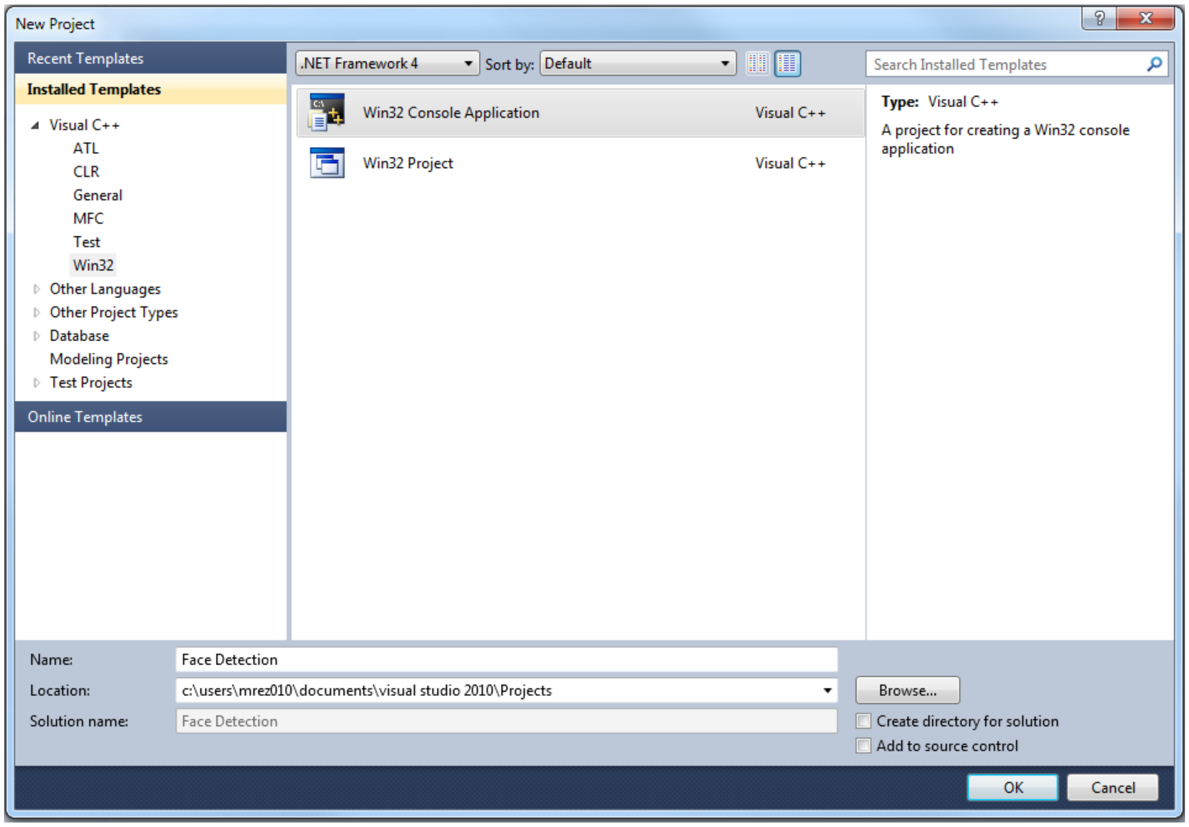
Variable	Value
PATH	C:\OpenCV2.3.1\build\x86\vc10\bin
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
CLASSPATH	.;C:\Program Files\Java\jre6\lib\ext\QT...
ComSpec	C:\Windows\system32\cmd.exe
DellClientSystem...	C:\Program Files\Dell\ClientSystemUpda...
FP_NO_HOST_C...	NO

2. Create a New Project:

- In Visual Studio 2010 C++ developer, create new application by following one of three ways below:
  - Click: New Project... or
  - Menu: File → New… → Project or
  - Shortcut: Ctrl + Shift + N
- Choose "Visual C++" Template → "Win 32" → "Win32 Console Application"



New Project

Recent Templates

Installed Templates

- Visual C++
  - ATL
  - CLR
  - General
  - MFC
  - Test
  - Win32
- Other Languages
- Other Project Types
- Database
- Modeling Projects
- Test Projects

Online Templates

.NET Framework 4 | Sort by: Default | Search Installed Templates

Win32 Console Application	Visual C++
Win32 Project	Visual C++

Type: Visual C++

A project for creating a Win32 console application

Name: Face Detection

Location: c:\users\mrez010\documents\visual studio 2010\Projects

Solution name: Face Detection

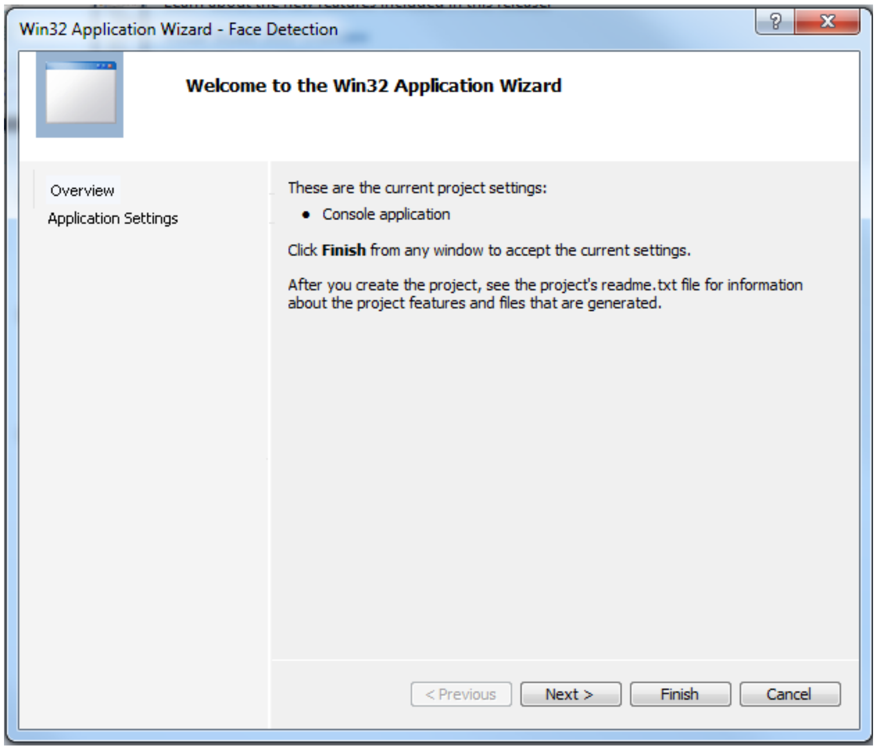
Browse...

☐ Create directory for solution

☐ Add to source control

OK | Cancel

- Select a "Location" and a "Name" for your program (e.g. Face Detection)
- Deselect "Create directory for solution" and Click OK
- Win32 Application Wizard Menu appears, Click Next.


Win32 Application Wizard - Face Detection

Welcome to the Win32 Application Wizard

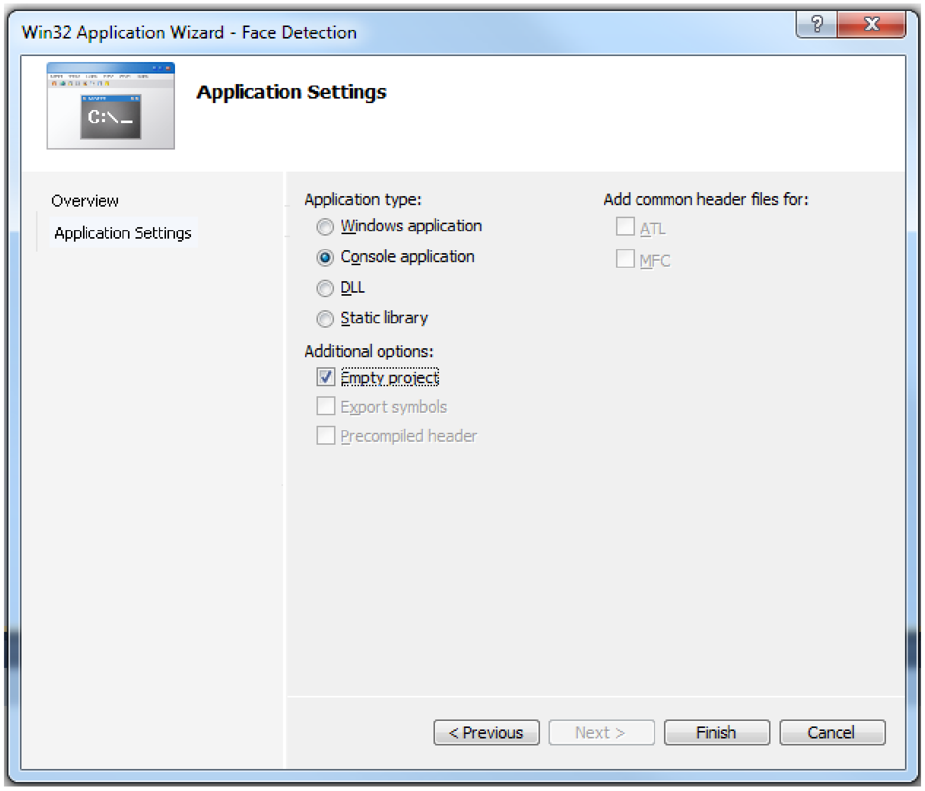
These are the current project settings:

- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

- In application setting, Additional options, tick “Empty project”


Win32 Application Wizard - Face Detection

Application Settings

Application type:

( ) Windows application
(●) Console application
( ) DLL
( ) Static library

Additional options:

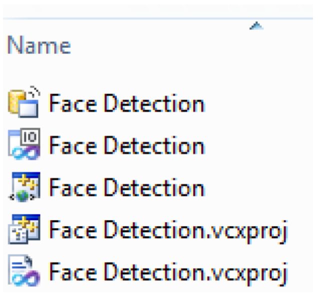
[✓] Empty project
[ ] Export symbols
[ ] Precompiled header

Add common header files for:

[ ] ATL
[ ] MFC

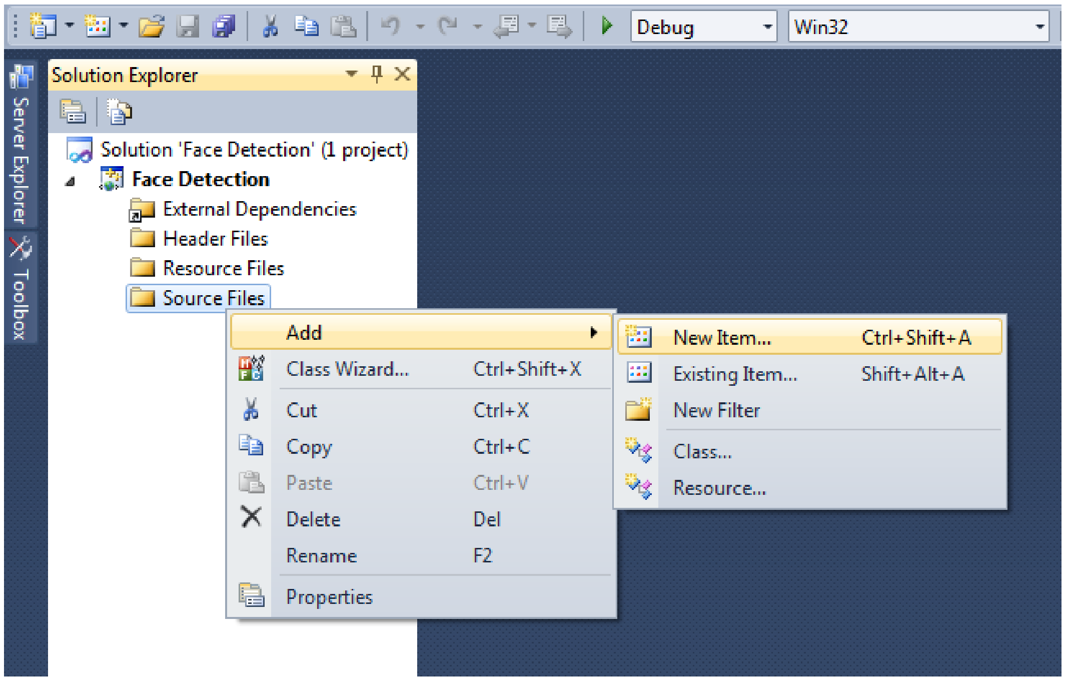
After these steps, the application wizard will create a folder with the same project name and in the same path that you already defined.

Following above steps, a few files like below will be created in your project folder:

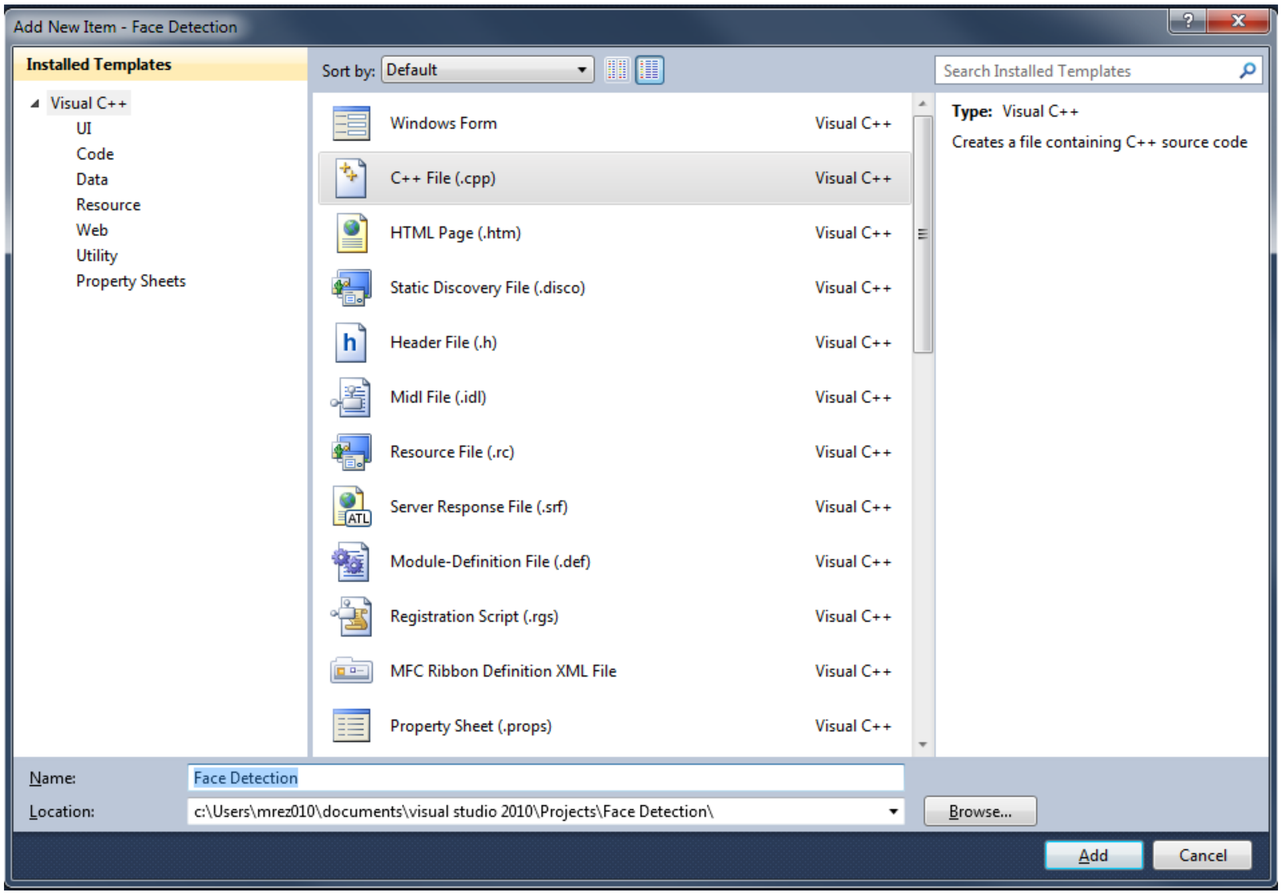

Name
Face Detection
Face Detection
Face Detection
Face Detection.vcxproj
Face Detection.vcxproj

Ok, now back to the Visual Studio developing window.

- Right Click on the **Source Files** → **Add** → **New Item**…



- A new window appears (see below). Select **C++ File**, Add the Name (e.g. **Face Detection**) and click **Add**.


Add New Item - Face Detection
Installed Templates
Visual C++
UI
Code
Data
Resource
Web
Utility
Property Sheets
Sort by: Default
Search Installed Templates
Windows Form Visual C++
C++ File (.cpp) Visual C++
HTML Page (.htm) Visual C++
Static Discovery File (.disco) Visual C++
Header File (.h) Visual C++
Midl File (.idl) Visual C++
Resource File (.rc) Visual C++
Server Response File (.srf) Visual C++
Module-Definition File (.def) Visual C++
Registration Script (.rgs) Visual C++
MFC Ribbon Definition XML File Visual C++
Property Sheet (.props) Visual C++
Browse...
Add
Cancel

Type: Visual C++

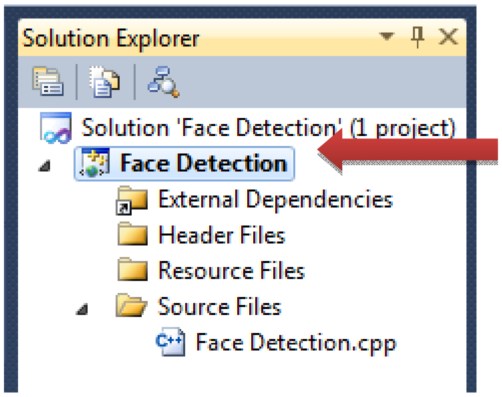
Creates a file containing C++ source code

Name: Face Detection

Location: c:\Users\mrez010\documents\visual studio 2010\Projects\Face Detection\

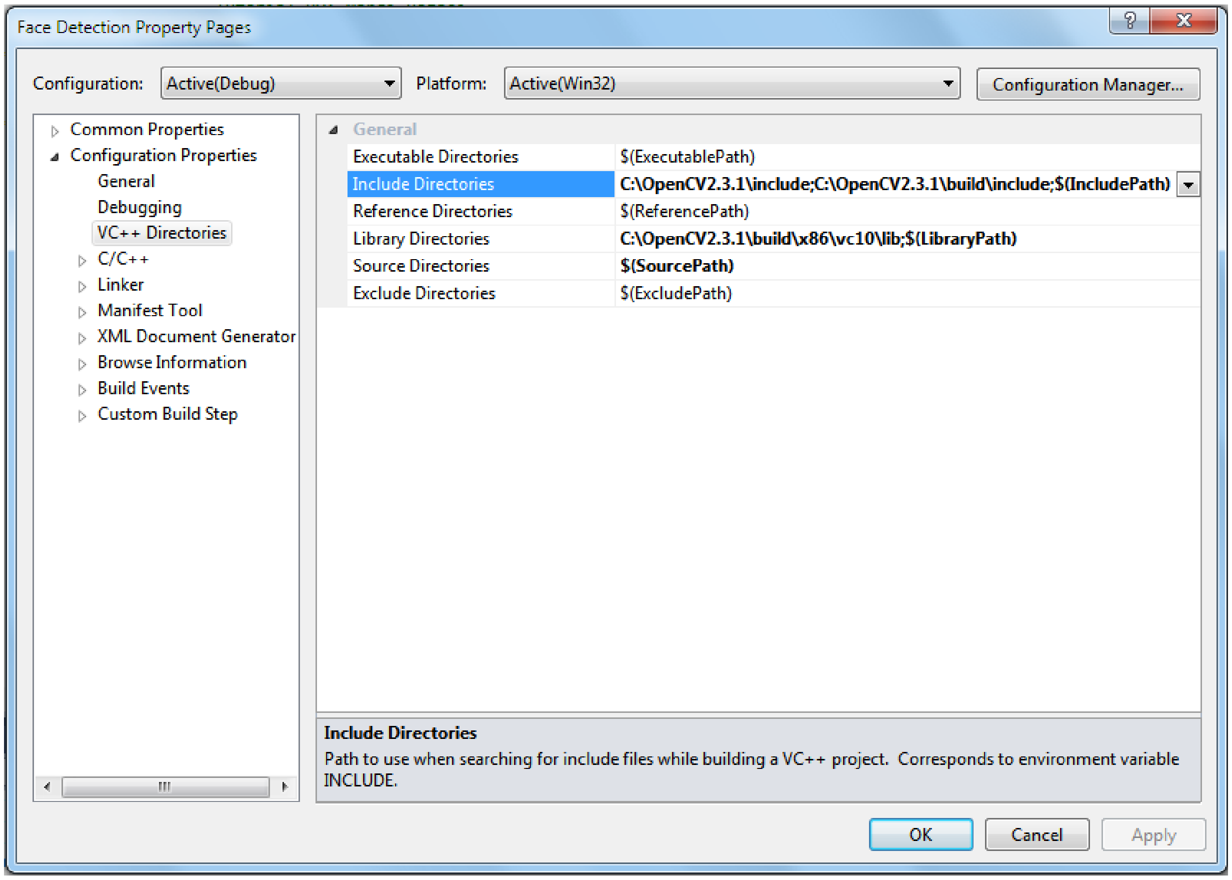
- Write your code and when finished, build your solution by pressing F7. You may expect to see some errors! To resolve the problem, you need to put some additional setting as follows:

- From Solution Explorer, right click on **Face Detection** or press **Alt + F7**



- Select **Configuration Properties → VC++ Directories** tab

  Make sure to put the paths **C:\OpenCV2.3.1\include**; **C:\OpenCV2.3.1\build\include**; for *Include Directories* as well as **C:\OpenCV2.3.1\build\x86\vc10\lib** for *Library Directories* (see below).



General

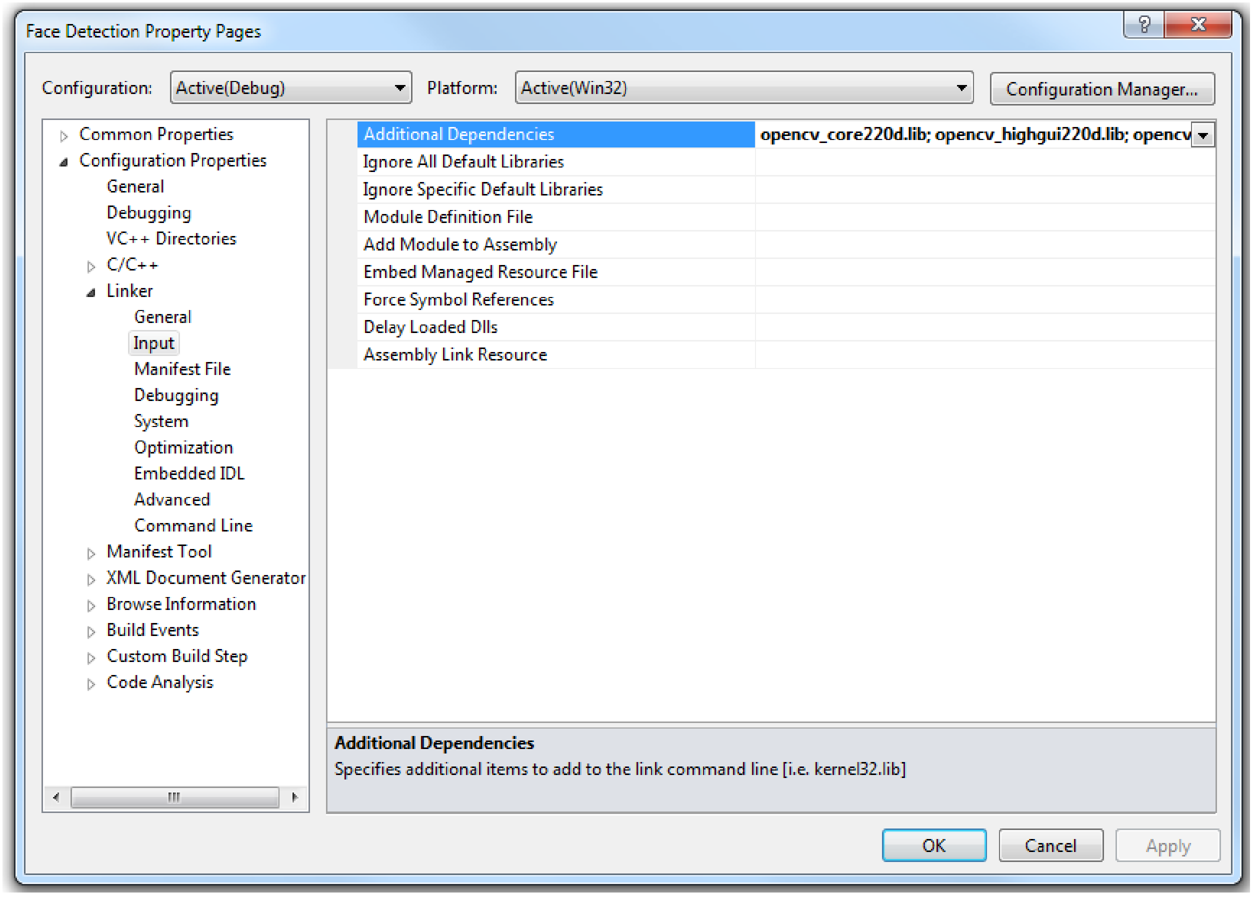
Executable Directories	$(ExecutablePath)
Include Directories	C:\OpenCV2.3.1\include;C:\OpenCV2.3.1\build\include;$(IncludePath)
Reference Directories	$(ReferencePath)
Library Directories	C:\OpenCV2.3.1\build\x86\vc10\lib;$(LibraryPath)
Source Directories	$(SourcePath)
Exclude Directories	$(ExcludePath)

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

- From **Linker** tab → **Input** Category → **Additional Dependencies**

  In front of Additional Dependencies put the paths for all needed libraries such as:

  opencv_core231d.lib; opencv_highgui231d.lib; opencv_imgproc231d.lib; opencv_objdetect231d.lib;



Additional Dependencies: opencv_core220d.lib; opencv_highgui220d.lib; opencv

Additional Dependencies
Specifies additional items to add to the link command line [i.e. kernel32.lib]

- Click Apply to save your settings
- That's it! Now, you can compile, run and enjoy your program ☺ - Press F5

3. Brief Review on the Provided Source Code

Preliminary settings before running the program:

1- In this program, there is a classifier which uses data stored in an XML file to decide how to classify input images. In order to initiate the classifier in your program, you need to identify where the XML data file is located on your computer.

The one you can use is **haarcascade_frontalface_default.xml**. In OpenCV version 2.3.1 it's located in: C:\OpenCV2.3.1\data\haarcascades\

It's a good idea to copy this XML file into your project folder to make you program more portable. Make sure the path you are defining is correct.

2- You'll also need a few images to test your program. The image names should be sequential with a same image extension; such as P0000.bmp, P0001.bmp, ..., P0124.bmp

In the provided source code the program loads images from path **\Face Detection\MyImages**.

The classifier file(s) are located in path **\Face Detection\MyCascade**
So, make sure to put these two folders into your project folder.

Initializing the Detector:

`CvHaarClassifierCascade *cascade;`
Defining `cascade` as a pointer to the structure - a trained classifier from a XML file

`CvMemStorage *storage;`
It is used to define a growing memory structure when you need to store dynamically growing data structures such as loading image sequences in your program.

(1), (2):
Both doing the same job in different ways. The function `cvLoad()`, loads the XML file that `CascadeFile` point to (i.e. `haarcascade_frontalface_alt2.xml`).

`cvLoad()` is a general-purpose function for loading data from files. It can take up to four input parameters. For this example, you'll only need the first parameter which is the path to an XML file containing a valid Haar Cascade file. Here, we are loading a frontal face detectors `MyCascade/haarcascade_frontalface_alt2.xml`

(3):
Before loading and detecting faces in images, you'll also need to initiate *`CvMemStorage`* object. This is a memory buffer that expands automatically, as needed. The face detector will put the list of detected faces into this buffer. Since the buffer is expandable, you won't need to worry about overflowing. All you need is to create a dynamic memory storage as below:

`storage = cvCreateMemStorage(0);`
0 is the default value for initial memory block size.

(4):
This line loads an input image in order to perfom face detection.

`IplImage,` is a data structure which is inherited from Intel Image Processing Library (IPL).

```
IplImage *inImage = cvLoadImage("GT.jpg",1);
```

Loads image data “GT.jpg” into `inImage`. 1 meand load the image as a 3-channel image
Parameters:

`>0 or CV_LOAD_IMAGE_COLOR`	the loaded image is forced to be a 3-channel color image
`=0 or CV_LOAD_IMAGE_GRAYSCALE`	the loaded image is forced to be grayscale
`<0 or CV_LOAD_IMAGE_UNCHANGED`	the loaded image will be loaded as is

(5):
Since it's easy to get a path wrong, lack of system memory, or mis-adressing the input file. So a good idea is to insert a quick check before going furture for face detection. With this we make sure everything loaded and initialized properly.

This module does a simple error check, shows a diagnostic message, and exit if the initialil check fails

```
        if( !cascade  || !storage || !inImage)
            {
                printf("Initialization Failed:%s\n",
                (!cascade)? " Cascade file not found !\n":
                (!storage)?  " Not memmory allocated or not enough memory !\n":
                             " The input file can not be found!\n");
                 system("pause");
                 return 0;
            }
```

(6):
Calls the function `detecFaces()` for face detection. The function shows the image `GT.jpg` which already was loaded into `inImage` along with detected faces, and finally realses the input image from memory.

(7):
This part follows a similar manner, but loads a sequence of multiple images P0000.bmp, P0001.bmp, and P0002.bmp … via a *for* loop. The loop loads images one by one, calls the function `detecFaces()` and shows detected faces for each image, seperately.

(8):
We are close to the end of `main()` and now we should clean up the occupied memory by release functions: `cvReleaseHaarClassifierCascade(&cascade1)` and `cvReleaseMemStorage(&storage)`

Now let's go for detail of function `detectFaces` in section (9)

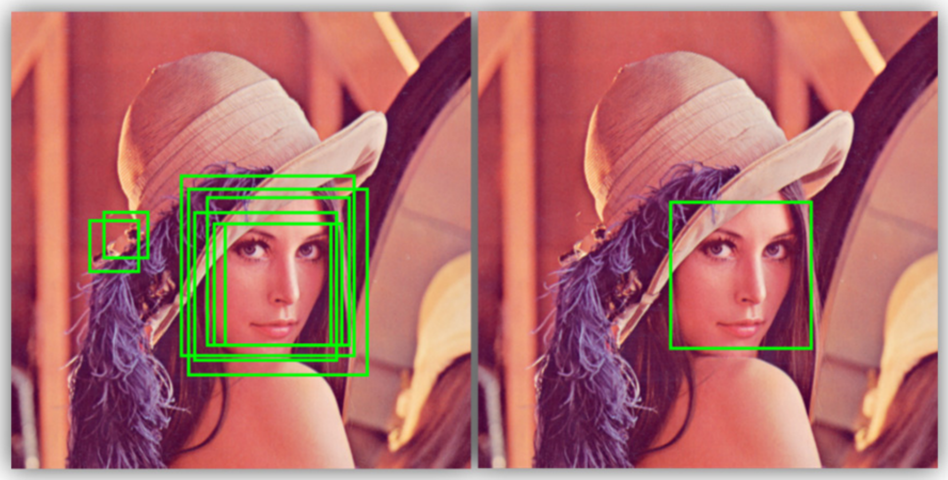
(9):
```
CvSeq * faces = cvHaarDetectObjects( newframe, cascade, storage,
                                     1.4, 1, 0, cvSize( 100, 100 ) );
```

`cvHaarDetectObjects()` runs the face detector. The function takes up to 7 parameters:

1- **Image pointer:** declares the input file here, `newframe`.
2- **Classifier:** Identifies the cascade classifier `cascade`, that we introduced it at the beginning of the program.
3- **Buffer:** The memory buffer (`storage`) that also previously allocated to handle face detection process.
7- yes 7!, **Window Size:** the parameter `cvSize(W,H)`, defines the size of smallest face to be searched within the input image. Actually this is the size of initial sliding-window. The default size in OpenCV is w=24 and h=24. However based on input image a tiny sub-window of `24x24` may not be meaningful as a face. So you may increase the size of initial search windows e.g. to `100x100`.
4- **Scale Rate:** The fourth input parameter specifies how quickly OpenCV should increase the scale for face detections, within search iterations. Higher values for this parameter will cause the detector run faster (by running fewer number of sliding window in each iteration), but if it's too high, it may jump too quickly between scales and miss some of the faces. Default

value for this parameter in OpenCV is 1.1 that means the scale increases by a factor of 10% on each pass.

5- **Minimum Neighbors Threshold**: when you call the face detector, for each positive face region actually it may generate many hits from the Haar detector. However, in face region there would be a large cluster of rectangles with largely overlaps. In addition, there may be some scattered detections which may appear around the face region. Usually, isolated detections are false detections, so it makes sense to discard these detections. It also makes sense to somehow merge the multiple detections for each face region into a single detection. The fifth parameter does both above actions before returning the final detected face. This merge step at first groups rectangles that contain a large amount of overlap, then finds the average rectangle for the group. It then replaces all rectangles in the group with the average rectangle. For example the minimum-neighbors threshold 3 means to merge groups of three or more and discard groups with fewer rectangles. If you find that your face detector is missing a lot of faces, you might try lowering this threshold; or increase it, if you find multiple detections for a single face.



6- **Flag Variable:** is the sixth parameter to `cvHaarDetectObjects()`. For this flag there are 6 available options. However, we discuss here on two most important ones.
The flag can be set either `0`, what means no change in the input image or as `CV_HAAR_DO_CANNY_PRUNING`. The second option means by running canny edge detection, the detector skips image regions that are flat (no edge) which are unlikely to contain a face. This can reduce overall computational time and possibly eliminating some false detections. However depend on your application this may also lead to missing some existing faces.

After face detection, second part of module (9) draws a rectangle to show the location of detected face. The code below draws *n* rectangle (i=n) for the *n* detected face as below:
Number of repeat in loop depends on the number of detected face in previous step which is stored in `faces`.

The structure `CvSeq` is growable sequence of elements (here detected faces) and is a base for all of OpenCV dynamic data structures)

The first line inside the loop

```
CvRect *r = ( CvRect *)cvGetSeqElem( faces, i );
```
Get and returns three Square elements of the detected face:

- top left position of the face as (x, y)
- width
- height

`CvRectangle()` draws a rectangle based on above information.

```
cvRectangle(CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color, int
            thickness=1, int lineType=8, int shift=0)
```

Parameters in `cvRectangle`:

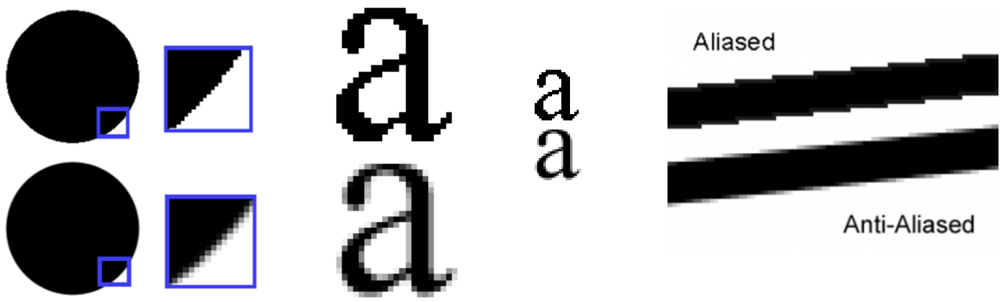
- *img* – Image
- *pt1* – top left rectangle's vertices
- *pt2* – opposite rectangle vertex
- *color* – Line color (RGB) or brightness (in gray-scale image)
- *thickness* – Thickness of lines that make up the rectangle. Negative values, or CV_FILLED, cause the function to draw a filled rectangle.
- *lineType* – Type of the line
- *shift* – Number of fractional bits in the point coordinates

Parameters in our code:

```
for( int i = 0 ; i < ( faces ? faces->total : 0 ) ; i++ )
      {
        CvRect* r = ( CvRect* )cvGetSeqElem( faces, i );
        cvRectangle( newframe,
                     cvPoint( r->x, r->y ),
                     cvPoint( r->x + r->width, r->y + r->height ),
                     CV_RGB( 0, 255, 0 ), 2, 8, 0 );
      }
```

- `Newframe:` The image in which the rectangle should be drawn on.
- `cvPoint( r->x, r->y ):`determines the top-left coner of the rectangle which is `x, y`
- `cvPoint( r->x + r->width, r->y + r->height ):` defines the right-bottom corner of the rectangle which is "x+width" and "y+height".
- `CV_RGB( 0, 255, 0 ):` Defines the rectangle color (here green).
- `2:` Tichness of the rectangle in pixel (here 2 pixel).
- `8:` Linetype (8-connected, 4- connected, CV_AA*).
- `0:` No Shift (each shift value devides the rectangle size and location by 2). e.g. Shift=3 means deviding the rectangule size and location by 8 .

*CV_AA: means Anti-Aliased but it is not applicable in rectangle drawing



As discussed earlier, finally in module (8) all the resources release and the program exits.

Eye Detection: For eye detection you just need to replace the classifier with an eye-detection cascade file, plus modifing cvSize(100, 100) to a smaller sliding windows. That is because an “eye” is smaller than a “face”, so we need to start with an smaller sliding window.

Next tutorial will be about how to create your own classifier (for face or eye detection).

Lab Practices:

1- Setup and run provided source code as described in tutorial.
2- Adjust face detection parameters to detect the highest rate of True Positives and less number of False Detection.

Further Practices:
Instead of using a green rectangle to outline the detected faces, try to use other types of OpenCv “drawing functions” such as cvCircle or cvEllipse.

Key: 115503030

More information: You can fined more advanced information via our published academic papers with various applications for face detection [1], [3], open or closed eye state detection [4], [6], vehicle detections and driver assistance systesm [5], and even facial feature improvements in artistic painting and rendering [7],[8].

References:

[1] Mahdi Rezaei, Reinhard Klette, “3D cascade of classifiers for open and closed eye detection in driver distraction monitoring”, *In Proc. Computer Analysis of Images and Patterns*, pp. 171-179, 2011.

[2] Mahdi Rezaei, Reinhard Klette, “Simultaneous Analysis of Driver Behaviour and Road Condition for Driver Distraction Detection”, International Journal of Image and Data Fusion, **2**, 217-236, 2011.

[3] Mahdi Rezaei, Hossein Ziaei Nafchi, Sandino Morales, “Global Haar-like Features: A New Extension of Classic Haar Features for Efficient Face Detection in Noisy Images”, *6th Pacific-Rim Symposium on Image and Video Technology*, PSIVT 2013.

[4] Mahdi Rezaei, Reinhard Klette, “Novel Adaptive Eye Detection and Tracking for Challenging Lighting Conditions”, *Asian Conference on Computer Vision Workshops (ACCV)*, pp. 427-440, 2013.

[5] Mahdi Rezaei, Mutsuhiro Terauchi, Vehicle Detection Based on Multi-feature Clues and Dempster-Shafer Fusion Theory, *6th Pacific-Rim Symposium on Image and Video Technology*, PSIVT 2013.

[6] Mahdi Rezaei, Reinhard Klette, "Adaptive Haar-like Classifier for Eye Status Detection Under Non-ideal Lighting Conditions", Proceedings of the 27th Conference on Image and Vision Computing New Zealand (IVCNZ), pp. 521- 526, 2012.

[7] Mahdi Rezaei, "Artistic Rendering of Human Portraits Paying Attention to Facial Features", Arts and Technology, **101**, pp. 90-99, 2012.

[8] Mahdi Rezaei, Juan Lin, Reinhard Klette, "Hybrid Filter Blending to Maintain Facial Expressions in Rendered Human Portraits", International Journal of Arts and Technology, 2014.