HOW THE CODE WORKS

# Get user supplied values

imagePath = sys.argv[1]

cascPath = sys.argv[2]

First pass in the image and cascade names as command-line arguments. We’ll use the image as well as the default cascade for detecting faces provided by OpenCV

# Create the haar cascade

faceCascade = cv2.CascadeClassifier(cascPath)

Now we create the cascade and initialize it with our face cascade. This loads the face cascade into memory so it’s ready for use. Remember, the cascade is just an XML file that contains the data to detect faces.

# Read the image

image = cv2.imread(imagePath)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

Here we read the image and convert it to grayscale. Many operations in OpenCV are done in grayscale.

# Detect faces in the image

faces = faceCascade.detectMultiScale(

gray,

scaleFactor=1.1,

minNeighbors=5,

minSize=(30, 30),

flags = cv2.cv.CV\_HAAR\_SCALE\_IMAGE

)

This function detects the actual face and is the key part of our code, so let’s go over the options:

1. The [detectMultiScale function](http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html" \l "cascadeclassifier-detectmultiscale) is a general function that detects objects. Since we are calling it on the face cascade, that’s what it detects.
2. The first option is the grayscale image.
3. The second is the scaleFactor. Since some faces may be closer to the camera, they would appear bigger than the faces in the back. The scale factor compensates for this.
4. The detection algorithm uses a moving window to detect objects. minNeighbors defines how many objects are detected near the current one before it declares the face found. minSize, meanwhile, gives the size of each window.

The function returns a list of rectangles in which it believes it found a face. Next, we will loop over where it thinks it found something.

print "Found {0} faces!".format(len(faces))

# Draw a rectangle around the faces

for (x, y, w, h) in faces:

cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

This function returns 4 values: the x and y location of the rectangle, and the rectangle’s width and height (w , h).

We use these values to draw a rectangle using the built-in rectangle() function.

cv2.imshow("Faces found", image)

cv2.waitKey(0)

In the end, we display the image and wait for the user to press a key.