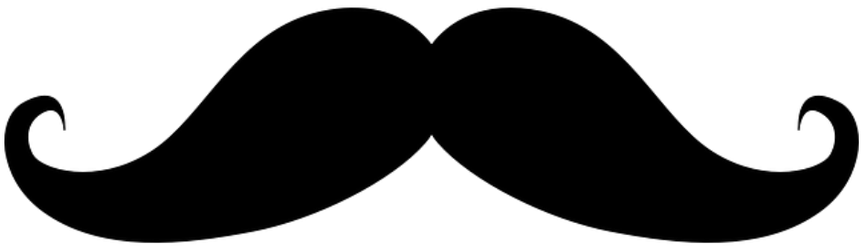

The Code

Below is the complete program code. You can copy and save this code into a new python file in your projects directory, and it should run without issue assuming that you have the following:

- Python working on your system.
- OpenCV python bindings working on your system.
- The Haar cascade classifier xml files installed as part of the OpenCV install.
- The mustache.png file saved to your project directory.
- Your webcam works (test with Cheese or any other webcam program).

The full code:

```
import cv2  # OpenCV Library

#-------------------------------------------------------------------------------
#       Load and configure Haar Cascade Classifiers
#-------------------------------------------------------------------------------

# location of OpenCV Haar Cascade Classifiers:
baseCascadePath = "/usr/local/share/OpenCV/haarcascades/"

# xml files describing our haar cascade classifiers
faceCascadeFilePath = baseCascadePath + "haarcascade_frontalface_default.xml"
noseCascadeFilePath = baseCascadePath + "haarcascade_mcs_nose.xml"

# build our cv2 Cascade Classifiers
faceCascade = cv2.CascadeClassifier(faceCascadeFilePath)
noseCascade = cv2.CascadeClassifier(noseCascadeFilePath)

#-------------------------------------------------------------------------------
#       Load and configure mustache (.png with alpha transparency)
#-------------------------------------------------------------------------------

# Load our overlay image: mustache.png
imgMustache = cv2.imread('mustache.png',-1)

# Create the mask for the mustache
orig_mask = imgMustache[:,:,3]

# Create the inverted mask for the mustache
orig_mask_inv = cv2.bitwise_not(orig_mask)

# Convert mustache image to BGR
# and save the original image size (used later when re-sizing the image)
imgMustache = imgMustache[:,:,0:3]
origMustacheHeight, origMustacheWidth = imgMustache.shape[:2]

#-------------------------------------------------------------------------------
#       Main program loop
#-------------------------------------------------------------------------------

# collect video input from first webcam on system
video_capture = cv2.VideoCapture(0)

while True:
    # Capture video feed
    ret, frame = video_capture.read()

    # Create greyscale image from the video feed
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in input video stream
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.cv.CV_HAAR_SCALE_IMAGE
    )

   # Iterate over each face found
    for (x, y, w, h) in faces:
        # Un-comment the next line for debug (draw box around all faces)
        # face = cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)

        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]

        # Detect a nose within the region bounded by each face (the ROI)
        nose = noseCascade.detectMultiScale(roi_gray)

        for (nx,ny,nw,nh) in nose:
            # Un-comment the next line for debug (draw box around the nose)
            #cv2.rectangle(roi_color,(nx,ny),(nx+nw,ny+nh),(255,0,0),2)

            # The mustache should be three times the width of the nose
            mustacheWidth =  3 * nw
            mustacheHeight = mustacheWidth * origMustacheHeight / origMustacheWidth

```

```
            # Center the mustache on the bottom of the nose
            x1 = nx - (mustacheWidth/4)
            x2 = nx + nw + (mustacheWidth/4)
            y1 = ny + nh - (mustacheHeight/2)
            y2 = ny + nh + (mustacheHeight/2)

            # Check for clipping
            if x1 < 0:
                x1 = 0
            if y1 < 0:
                y1 = 0
            if x2 > w:
                x2 = w
            if y2 > h:
                y2 = h

            # Re-calculate the width and height of the mustache image
            mustacheWidth = x2 - x1
            mustacheHeight = y2 - y1

            # Re-size the original image and the masks to the mustache sizes
            # calcualted above
            mustache = cv2.resize(imgMustache, (mustacheWidth,mustacheHeight), interpolation = cv2.INTER_AREA)
            mask = cv2.resize(orig_mask, (mustacheWidth,mustacheHeight), interpolation = cv2.INTER_AREA)
            mask_inv = cv2.resize(orig_mask_inv, (mustacheWidth,mustacheHeight), interpolation = cv2.INTER_AREA)

            # take ROI for mustache from background equal to size of mustache image
            roi = roi_color[y1:y2, x1:x2]

            # roi_bg contains the original image only where the mustache is not
            # in the region that is the size of the mustache.
            roi_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)

            # roi_fg contains the image of the mustache only where the mustache is
            roi_fg = cv2.bitwise_and(mustache,mustache,mask = mask)

            # join the roi_bg and roi_fg
            dst = cv2.add(roi_bg,roi_fg)

            # place the joined image, saved to dst back over the original image
            roi_color[y1:y2, x1:x2] = dst

            break

    # Display the resulting frame
    cv2.imshow('Video', frame)

    # press any key to exit
    # NOTE;  x86 systems may need to remove: " 0xFF == ord('q')"
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

Examining the Code in Depth

We will now walk through the code a section at a time. If you have any difficulties understanding the code, work through the tutorials above, which should provide a solid foundation for this code.

Load the Haar Cascade Classifiers

The Haar Cascade Classifiers are xml files installed with OpenCV that define specific features (such as a face or nose) to be detected in an image. The default folder for these files on my system is **/usr/local/share/OpenCV/haarcascades/**. Since we are interested in detecting faces in the image (a webcam video is just a series of images we will process and modify one frame at a time), in the snippet of code below we load the **haarcascade_frontalface_default.xml** and the **haarcascade_mcs_nose.xml** classifier files, which are used to identify the frontal view of a face and a nose, respectively.

```
1  import cv2  # OpenCV Library
2
3  #-----------------------------------------------------------------------------
4  #       Load and configure Haar Cascade Classifiers
5  #-----------------------------------------------------------------------------
6
7  # location of OpenCV Haar Cascade Classifiers:
8  baseCascadePath = "/usr/local/share/OpenCV/haarcascades/"
9
10 # xml files describing our haar cascade classifiers
11 faceCascadeFilePath = baseCascadePath + "haarcascade_frontalface_default.xml"
12 noseCascadeFilePath = baseCascadePath + "haarcascade_mcs_nose.xml"
13
14 # build our cv2 Cascade Classifiers
15 faceCascade = cv2.CascadeClassifier(faceCascadeFilePath)
16 noseCascade = cv2.CascadeClassifier(noseCascadeFilePath)
```

Line 11: This is the full path to the .xml file for the front-face Haar classifier.
Line 12: This is the full path to the .xml file for the nose Haar classifier.

Line 15: Build the Haar cascade classifier for the face (used at line 51).
Line 16: Build the Haar cascade classifier for the nose (used at line 68).

Side Note: we are doing facial **detection**, rather than facial **recognition**. The difference is that detection merely tells us that it has found a face (or a region of the image that looks like a face), while facial recognition is when a detected faces is compared against a database of faces to specifically identify one individual from that database.

Load the Mustache

In the next code snippet below, we load the mustache image and create our image masks. The image masks are used to select sections from an image that we want to display. When we overlay the image of a mustache over a background image, we need to identify which pixels from the mustache image should be displayed, and which images from the background image should be displayed. The masks are used to identify which pixels should be used when applied to an image.

```
18 #-----------------------------------------------------------------------------
19 #       Load and configure mustache (.png with alpha transparency)
```

```
20 #-----------------------------------------------------------------------------
21
22 # Load our overlay image: mustache.png
23 imgMustache = cv2.imread('mustache.png',-1)
24
25 # Create the mask for the mustache
26 orig_mask = imgMustache[:,:,3]
27
28 # Create the inverted mask for the mustache
29 orig_mask_inv = cv2.bitwise_not(orig_mask)
30
31 # Convert mustache image to BGR
32 # and save the original image size (used later when re-sizing the image)
33 imgMustache = imgMustache[:,:,0:3]
34 origMustacheHeight, origMustacheWidth = imgMustache.shape[:2]
```

Line 23: We load the mustache with -1 (negative one) as the second parameter to load all the layers in the image. The image is made up of 4 layers (or channels): Blue, Green, Red, and an Alpha transparency layer (knows as BGR-A). The alpha channel tell us which pixels in the image should be transparent (show the image underneath) or should be non-transparent (made up of a combination of the other 3 layers).
Line 26: Here we take just the alpha layer and create a new single-layer image that we will use for masking.
Line 29: We take the inverse of our mask. The initial mask will define the area for the mustache, and the inverse mask will be for the region around the mustache).
Line 33: Here we convert the mustache image to a 3-channel BGR image (BGR rather than BGR-A is required when we overlay the mustache image over the webcam image later).
Line 34: Save the original mustache image sizes, which we will use later when re-sizing the mustache image.

Main Program Loop

Now we begin capturing images from the webcam and detecting faces in the stream of webcam frames.

```
36 #-----------------------------------------------------------------------------
37 #       Main program loop
38 #-----------------------------------------------------------------------------
39
40 # collect video input from first webcam on system
41 video_capture = cv2.VideoCapture(0)
42
43 while True:
44     # Capture video feed
45     ret, frame = video_capture.read()
46
47     # Create greyscale image from the video feed
48     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Line 41: Set the video capture device to the first webcam on the system.
Line 43: Begin an endless loop (we break out later if we detect a key press)
Line 45: Capture a frame from the webcam, save it in **frame** variable.
Line 48: Haar Cascade Classifiers operate on greyscale images, so save a grayscale image to the **grey** variable.

Detecting Faces and Noses

This is where we use the Haar cascade classifiers to detect the face and nose.

```
50 # Detect faces in input video stream
51  faces = faceCascade.detectMultiScale(
52      gray,
53      scaleFactor=1.1,
54      minNeighbors=5,
55      minSize=(30, 30),
56      flags=cv2.cv.CV_HAAR_SCALE_IMAGE
57  )
58
59 # Iterate over each face found
60  for (x, y, w, h) in faces:
61      # Un-comment the next line for debug (draw box around all faces)
62      # face = cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
63
64      roi_gray = gray[y:y+h, x:x+w]
65      roi_color = frame[y:y+h, x:x+w]
66
67      # Detect a nose within the region bounded by each face (the ROI)
68      nose = noseCascade.detectMultiScale(roi_gray)
69
70      for (nx,ny,nw,nh) in nose:
71          # Un-comment the next line for debug (draw box around the nose)
72          #cv2.rectangle(roi_color,(nx,ny),(nx+nw,ny+nh),(255,0,0),2)
```

Line 51: This line is where faces are identified in the image. The **faces** variable holds a list of the faces, defined by the x and y coordinates of the upper left corner of the face, and the width and height of the face.
We use the **faceCascade** cascade classifier created at line 15, and use the **detectMultiScale** function to detect the faces. The parameters passed to **detectMultiScale** help determine the minimum size of faces that can be detected in the image, how close faces can be together before they can't be detected, and other options. These parameters can be tweaked to help improve detection, and to decrease load on the system.
Line 60: We want to iterate through each face found in the webcam frame. Each face is defined by an x and y starting coordinate, and the width and height. Un-comment line 61 if you want to see boxes drawn around all discovered faces.
Line 64: We create a greyscale ROI for the area where the face was discovered (remember that we will be looking for a nose within this face, and Haar cascade classifiers operate on greyscale images.
Line 64: We also keep a color ROI for the area where the face is, as we will draw our mustache over the color ROI. (Remember that the x and y co-ordinates are backwards when selecting a ROI.
Line 68: Here we detect all noses found within the ROI that describes each face. Un-comment line 72 to draw a rectangle around the nose.
Line 70: Once we know were the nose is (x,y, width, height) in **relation** to the ROI bounded by the face…

Calculate the Size and Location of the Mustache

This section of the code is where the mustache is re-sized to remain proportional to the size of the face. What this means that as the face gets closer to the camera (gets larger), the mustache will also get larger, consistantly being 3 times the width of the nose, and scaling vertically proportionally.

In OpenCV we will often talk about **Regions of Interest**, or ROI. When modifying an image, we often will only modify a small section of the image. Rather than making changes to the entire image, we will select a region from the original image, where the coordinate system for the region is relative to the region, rather than the image. We can also take a ROI from a ROI if needed (you'll see this below). It's important to remember which ROI you are working in, as you can't edit outside a ROI without editing the parent of that ROI.

Second side note: When creating a ROI, the co-ordinates are backwards. You would write [y,x], rather than [x,y].

```
74  # The mustache should be three times the width of the nose
75  mustacheWidth =  3 * nw
76  mustacheHeight = mustacheWidth * origMustacheHeight / origMustacheWidth
77
78  # Center the mustache on the bottom of the nose
79  x1 = nx - (mustacheWidth/4)
80  x2 = nx + nw + (mustacheWidth/4)
81  y1 = ny + nh - (mustacheHeight/2)
82  y2 = ny + nh + (mustacheHeight/2)
83
84  # Check for clipping
85  if x1 < 0:
86      x1 = 0
87  if y1 < 0:
88      y1 = 0
89  if x2 > w:
90      x2 = w
91  if y2 > h:
92      y2 = h
93
94  # Re-calculate the width and height of the mustache image
95  mustacheWidth = x2 - x1
96  mustacheHeight = y2 - y1
97
98  # Re-size the original image and the masks to the mustache sizes
99  # calcualted above
100 mustache = cv2.resize(imgMustache, (mustacheWidth,mustacheHeight), interpolation = cv2.INTER_AREA)
101 mask = cv2.resize(orig_mask, (mustacheWidth,mustacheHeight), interpolation = cv2.INTER_AREA)
102 mask_inv = cv2.resize(orig_mask_inv, (mustacheWidth,mustacheHeight), interpolation = cv2.INTER_AREA)
```

Line 75: Re-size our original mustache image width to be 3 times the width of the nose (I felt this size looked good, you can play with this number without any issue).
Line 76: The height of the mustache with relation to the nose should be proportional to the width of the mustache and the original image (maintain proportional to the original image).
Line 79 – 82: We now place the mustache centered vertically at the bottom of the area bounded by the nose (in the ROI of the face), and vertically along the center of the nose.
Line 85 – 93: We need to ensure that the image of the mustache does not extend outside the ROI bounded by the face. We simply clip the mustache co-ordinates if this happens. Failing to do this will cause an error.
Line 95 – 96: Now that we know the x an y coordinates of the mustache (x1,y1) designates the upper left, while (x2,y2) for the lower right, in relation to the ROI for the face, we can calculate the width and height of the mustache we want to draw over the face.
Line 100: In this line, we create a new image, **mustache** which is a re-sized copy of our original BGR mustache (imgMustache).
Line 101: Here we create a new mask for the re-sized mustache.
Line 102: Here we create an inverted mask for the re-sized mustache.

Draw The Mustache Over the Frame

```
104 # take ROI for mustache from background equal to size of mustache image
105 roi = roi_color[y1:y2, x1:x2]
106
107 # roi_bg contains the original image only where the mustache is not
108 # in the region that is the size of the mustache.
109 roi_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)
110
111 # roi_fg contains the image of the mustache only where the mustache is
112 roi_fg = cv2.bitwise_and(mustache,mustache,mask = mask)
113
114 # join the roi_bg and roi_fg
115 dst = cv2.add(roi_bg,roi_fg)
116
117 # place the joined image, saved to dst back over the original image
118 roi_color[y1:y2, x1:x2] = dst
119
120 break
```

Line 105: We save a color ROI from the background image that is the size and location of where we will place our re-sized mustache.
Line 109: We use the **bitwise_and** function with our inverted mask to select the pixels in the background region where the mustache is not.
Line 112: We use the **bitwise_and** function again with our mask and the mustache image to select the pixels from our mustache that make up the mustache.
Line 115: Here we merge the two ROI images we created (since the masks are inverse of each other, the pixels selected in the two above **bitwise_and** operations won't overlap, but together will make up a full image, the size of our ROI.
Line 118: here we place the combination of our mustache and the area around the mustache back on the main image for display.
Line 120: We only want to detect one nose (multiple can be found by the classifier). We break to prevent other mustaches from being drawn on this face (other faces will still have mustaches drawn on them). This isn't an elegant solution, but it mostly works.

Show the Image

```
122     # Display the resulting frame
123     cv2.imshow('Video', frame)
124
125     # press any key to exit
126     # NOTE;  x86 systems may need to remove: if cv2.waitKey(1) & 0xFF == ord('q'):
127     if cv2.waitKey(1) & 0xFF == ord('q'):
128         break
129
130 # When everything is done, release the capture
131 video_capture.release()
132 cv2.destroyAllWindows()
```

Line 123: Display our frame with the mustache drawn over it.
Line 127: If the user presses **q**, then exit.
Line 131 & 132: Clean up and end.

Wrapping Things Up