Getting Christmassy with ARTIK and OpenCV

In our previous blog post, we talked about how to build a photo booth with ARTIK. To make your holiday a little merrier, we are going to extend our photo booth code to put Santa's hat on people. We will do this by using one of my favorite open source libraries OpenCV (Open Source Computer Vision) and its Python bindings.

Pre-requsites:

- An ARTIK 710 or ARTIK 1020 board
- An HDMI display and cable for your ARTIK
- A USB camera or MIPI camera for your ARTIK
- A keyboard for your HDMI display
- A Santa's hat image with a transparent background, saved to *hat.png*

1. I am using an ARTIK 710 and a USB camera for this project. Connect my ARTIK 710 to the HDMI output of a monitor. Plug in a USB keyboard and a USB camera.

2. Install X Windows on ARTIK 710

3. Install required OpenCV Python packages.

```
#dnf install python-devel opencv-python
```

4. Capture a photo

In the code below,

- We connect to the camera using *VideoCapture* () function. In my environment, I only have a USB camera connected so the index parameter of the function is 0.
- We can grab a frame from the camera with *read*().
- *cv2.imshow*() displays the frame in a pop-up window until we hit 'q' key.
- *release*() function releases your hold of the camera.
- The captured frame is saved into a file in *cv2.imwrite*().

```
import cv2

cap = cv2.VideoCapture(0)
while True:
  # Capture frame-by-frame
  ret, frame = cap.read()
  # Display the resulting frame
  cv2.imshow('Video', frame)
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break
cap.release()

cv2.imwrite("mockup.jpg", frame)
```

5. Face Detection

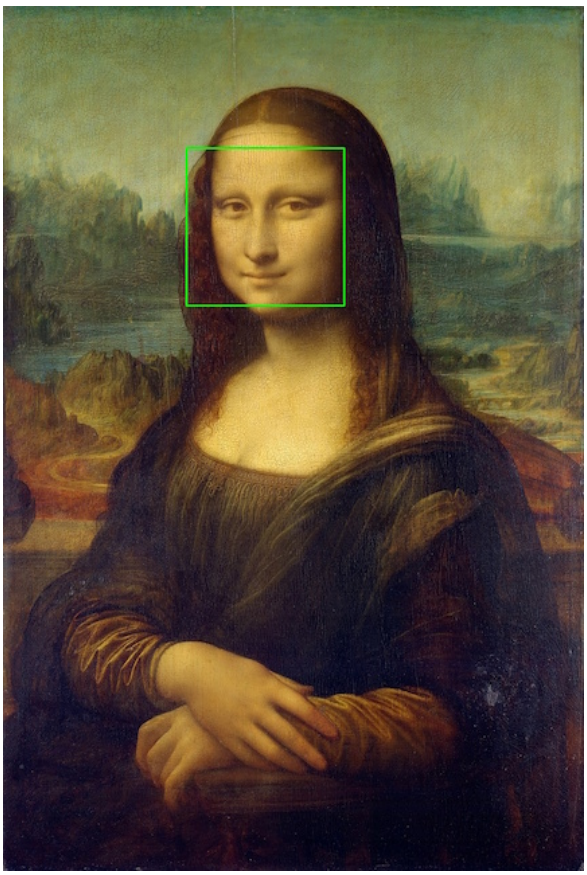
There are a lot of great online tutorials on how to use Haar Cascades for face detection, here is one you can find on OpenCV website. First, we have to download the Haar Cascade frontal face Classifier onto ARTIK.

```
# wget https//github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt.xml
```

We extended our code from step 4 to the block below:

- We create *faceCascade* object from *haarcascade_frontalface_alt.xml* file, then use this object to find faces by using the *detectMultiScale*() function.
- The parameters passed to the *detectMultiScale*() function determine the scaling factor, minimum size of faces that can be detected, and the minimum distances between faces etc.
- Once we have the identified faces, we draw a rectangle around each face.
- The mocked up image will show in the pop-up window, and our program ends when we hit 'q' key again.

```
import cv2

# Build cv2 Cascade Classifiers
faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
cap = cv2.VideoCapture(0)
while True:
  # Capture frame-by-frame
  ret, frame = cap.read()
  # Display the resulting frame
  cv2.imshow('Video', frame)
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break
cap.release()

# Create greyscale image from the video feed
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Detect faces in input video feed
faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.3,
        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)
  cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.imwrite("mockup.jpg", frame)
cv2.imshow('Video', frame)
while True:
  if cv2.waitKey(1) & 0xFF == ord('q'):
    cv2.destroyAllWindows()
    exit()
```



6. Place the hat

Now, we have a way to find face, we will place the hat to the top of the face.

6.1 Load the hat

- We load our hat image by using *cv2.imread*(). The -1 parameter tells the function to load the image as is, with alpha channel. hat.png we are using consists of 4 channels: Blue, Green, Red and an Alpha transparency channel(BGR-A). Alpha transparency channel indicates which pixels in the image are transparent.
- We take the alpha channel and create a single-layer image as our mask, also create an inverted mask. *orig_mask* defines the region of the hat, and *orig_mask_inv* defines the region around the hat.
- Convert hat image into a 3-channel BGR image.

```
# Load our overlay image: hat.png
imgHat = cv2.imread('hat.png',-1)
# Create the mask for the hat
orig_mask = imgHat[:,:,3]
# Create the inverted mask for the hat
orig_mask_inv = cv2.bitwise_not(orig_mask)

# Convert hat image to BGR
# and save the original image size (used later when re-sizing the image)
imgHat = imgHat[:,:,0:3]
orighatHeight, origHatWidth = imgHat.shape[:2]
```

6.2 Calculate the size and location of the hat

In OpenCV, common image operations are performed using Region of Interest(ROI). A ROI allows us to operate a rectangular region of an image. Normally, we create a ROI on the image, perform operations in this smaller region of the image rather than the entire image.

In this section of the code, we re-size the hat to make it fit into the frame, also stay proportional to the size of the face.

- In our example, we re-size the original hat image to 1.2 times the width of the face. Height should be proportional to the width. We can tweak the parameter based on the shape of your hat.
- Place the hat centered vertically to the top of the face.
- Make sure the hat image does not fall outside the frame boundary.
- Re-size the hat image, and masks to the hat sizes calculated.

```
 # The hat should be 1.2 times the width of the face
  hatWidth =  w + int(w * 0.2)
  hatHeight = hatWidth * origHatWidth / origHatWidth

  hat = cv2.resize(imgHat, (hatWidth,hatHeight), interpolation =
cv2.INTER_AREA)

  # Center the hat the top of face
  x1 = x - (hatWidth * 0.1)
  x2 = x + w + (hatWidth * 0.1)
  y1 = y - hatHeight + int(hatHeight * 0.2)
  y2 = y + int(hatHeight * 0.2)
  cropImg = 0

  # Check for clipping
  if x1 < 0:
    x1 = 0
  if x2 > frame.shape[0]:
    x2 = w
  if y1 < 0:
    y1 = 0
    cropImg = hat.shape[1] - y2
    hat = hat[cropImg:]
  if y2 > frame.shape[1]:
    y2 = h

  # Find region of interest
  roi_gray = gray
  roi_color = frame

  # Re-size the original image and the masks to the hat sizes
  # calcualted above
  mask = cv2.resize(orig_mask, (hatWidth,hatHeight), interpolation =
cv2.INTER_AREA)
  mask = mask[cropImg:]
  mask_inv = cv2.resize(orig_mask_inv, (hatWidth,hatHeight),
interpolation = cv2.INTER_AREA)
  mask_inv = mask_inv[cropImg:]
```

6.3 Draw the hat

- We take a ROI from the background image, with the size and location that we will place our hat image.
- The two *cv2.bitwise_and*() functions select the pixels in the background where the hat is not in the region, as well as the pixels that make up the hat.
- Add the two images and place the joined image back to the original image

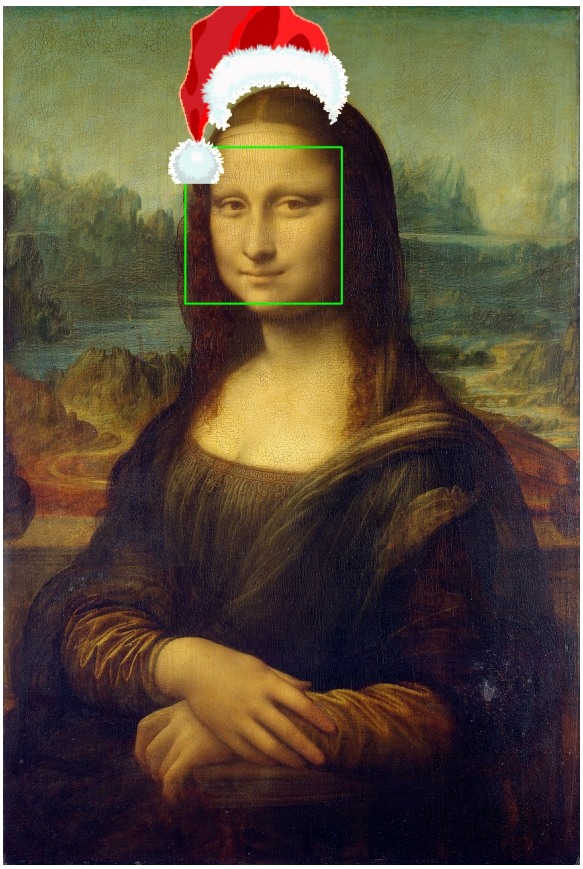
```
  # Take ROI for hat from background equal to size of hat image
  roi = roi_color[y1:y2, x1:x1+hatWidth]
  # roi_bg contains the original image only where the hat is not
  # in the region that is the size of the hat.
  roi_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)
  # roi_fg contains the image of the hat only where the hat is
  roi_fg = cv2.bitwise_and(hat,hat,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:x1+hatWidth] = dst
```

You can find the final code *hat.py* here.

Start X-window on your HDMI display. Use Alt+F1 to launch a terminal window, and run our fun photo app

```
#python hat.py
```

This is how Mona Lisa looks like with a Santa's hat.



ARTIK's processing power provides the capability to run OpenCV algorithms locally. For further readings, please check out OpenCV-Python Tutorials.

Have Fun and Merry Christmas!