**Skin Detection: A Step-by-Step Example using Python and OpenCV**

**<https://www.pyimagesearch.com/2014/08/18/skin-detection-step-step-example-using-python-opencv/>**

by [**Adrian Rosebrock**](https://www.pyimagesearch.com/author/adrian/) on August 18, 2014 in [**Tutorials**](https://www.pyimagesearch.com/category/tutorials/)

So last night I went out for a few drinks with my colleague, James, a fellow computer vision researcher who I have known for years.

You see, James likes to party. He’s a “go hard” type of guy. He wanted to hit every single bar on Washington St. (there’s a lot of them)…and then hit ’em again on the way back.

It’s safe to say that by the end of the night that *James was blotto*. Wasted. Lights out.

Anyway, it was getting late. 1:47am, to be exact. I’m tired. I just want to go to bed. We’re at the last bar. I’m signing my name on the tab and heading for the door, looking forward to getting some sleep.

And James, in his clearly drunken and intoxicated state, walks (or rather, stumbles like a newborn foal) up to this blonde who looks and dresses like she’s straight out of South Beach and says: *“You know…if you take your shirt off I have a Python script that can detect how much skin you’re showing. Wanna see?”*

Now tell me,*what do you think happens next?*

That’s right. James gets a top shelf Manhattan thrown in his face by the South Beach girl. Honestly, he’s just lucky that he didn’t catch a *well deserved* fist to the nose.

Shocked, and fairly appalled, I led the drunken James back to my apartment to sleep it off. Not to mention have a conversation with him regarding how to treat women the next morning.

So here’s the deal…

In this blog post I’m going to show you ***how to detect skin in images using computer vision.***

But use these superpowers for good, okay? Don’t be like James. Play it cool.

Read on and find out how ***simple*** it really is to detect skin in images using Python and OpenCV.

**OpenCV and Python versions:**  
This example will run on**Python 2.7/Python 3.4+** and **OpenCV 2.4.X/OpenCV 3.0+**.

You know the drill. Open up your favorite editor, create a new file, name it skindetector.py , and let’s get to work:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # import the necessary packages  from pyimagesearch import imutils  import numpy as np  import argparse  import cv2    # construct the argument parse and parse the arguments  ap = argparse.ArgumentParser()  ap.add\_argument("-v", "--video",  help = "path to the (optional) video file")  args = vars(ap.parse\_args())    # define the upper and lower boundaries of the HSV pixel  # intensities to be considered 'skin'  lower = np.array([0, 48, 80], dtype = "uint8")  upper = np.array([20, 255, 255], dtype = "uint8") |

On **Lines 2-5** we import the packages that we’ll need. We’ll use NumPy for some numerical processing, argparse  to parse our command line arguments, and cv2  for our OpenCV bindings.

We’ll also use a package called imutils  which contains a bunch of “convenience” image processing functions for resizing, rotating, etc. You can read more about theimutils  package in my [Basics of Image Manipulations](https://www.pyimagesearch.com/2014/01/20/basic-image-manipulations-in-python-and-opencv-resizing-scaling-rotating-and-cropping/) post.

From there, **Lines 8-11** parse our command line arguments. We have only a single (optional) switch, --video , which allows you to detect skin in a pre-supplied video. However, you could just as easily omit this switch from the command and use your webcam to detect skin in images. If you don’t have a webcam attached to your system, then you’ll have to supply a video.

From there, we define the *lower* and *upper* boundaries for pixel intensities to be considered skin on **Lines 15 and 16**. It’s important to note that these boundaries are for the *HSV color space,****NOT****the RGB color space****.***

If you haven’t had a chance to take a look at my post on [Color Detection Using Python and OpenCV](https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/), then now would be a good time to do so — we’ll be building off the fundamentals presented in the color detection post and extending our method to detect skin in images.

Alright, now let’s write some code to grab a reference to the webcam or video:

|  |  |
| --- | --- |
| 18  19  20  21  22  23  24  25 | # if a video path was not supplied, grab the reference  # to the gray  if not args.get("video", False):  camera = cv2.VideoCapture(0)    # otherwise, load the video  else:  camera = cv2.VideoCapture(args["video"]) |

On **Line 20**we make a check to see if the --video  switch was supplied. If it was not, then we grab reference to the webcam by passing a value of *0*  to the cv2.VideoCapture  function.

However, if the --video  switch was supplied, then we’ll pass the path to the video tocv2.VideoCapture  on **Lines 24 and 25**.

Let’s start reading frames from our video:

|  |  |
| --- | --- |
| 27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | # keep looping over the frames in the video  while True:  # grab the current frame  (grabbed, frame) = camera.read()    # if we are viewing a video and we did not grab a  # frame, then we have reached the end of the video  if args.get("video") and not grabbed:  break    # resize the frame, convert it to the HSV color space,  # and determine the HSV pixel intensities that fall into  # the speicifed upper and lower boundaries  frame = imutils.resize(frame, width = 400)  converted = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)  skinMask = cv2.inRange(converted, lower, upper)    # apply a series of erosions and dilations to the mask  # using an elliptical kernel  kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (11, 11))  skinMask = cv2.erode(skinMask, kernel, iterations = 2)  skinMask = cv2.dilate(skinMask, kernel, iterations = 2)    # blur the mask to help remove noise, then apply the  # mask to the frame  skinMask = cv2.GaussianBlur(skinMask, (3, 3), 0)  skin = cv2.bitwise\_and(frame, frame, mask = skinMask)    # show the skin in the image along with the mask  cv2.imshow("images", np.hstack([frame, skin]))    # if the 'q' key is pressed, stop the loop  if cv2.waitKey(1) & 0xFF == ord("q"):  break    # cleanup the camera and close any open windows  camera.release()  cv2.destroyAllWindows() |

We start looping over the frames in the video on **Line 28**and grab the next frame on **Line 30**using the camera.read()  method.

The camera.read()  function returns a tuple, consisting of grabbed  and frame . Thegrabbed  variable is simply a boolean flag, indicating if the frame was successfully read or not. The frame  is the frame itself.

From there, we make a check on **Line 34** to see if the frame  was not read. If it was not, and we are reading a video from file, we assume we have reached the end of the video and we can safely break from the loop. Otherwise, we keep on looping.

In order to speed up the skin detection process, we use our imutils.resize  convenience function to resize our frame to have a width of 400 pixels on **Line 40**.

The actual skin detection takes place on **Line 41 and 42**. First, we convert the image from the RGB color space to the HSV color space. Then, we apply the cv2.inRange  function, supplying our HSV frame, and our lower and upper boundaries as arguments, respectively.

The output of the cv2.inRange  function is our mask This mask is a single channel image, has the same width and height as the frame, and is of the 8-bit unsigned integer data type.

Pixels that are *white* (255) in the mask represent areas of the frame ***that are skin***. Pixels that are *black* (0) in the mask represent areas ***that are not******skin***.

However, we may detect many small false-positive skin regions in the image. To remove these small regions, take a look at **Lines 46-48**. First, we create an elliptical structuring kernel. Then, we use this kernel to perform two iterations of erosions and dilations, respectively. These erosions and dilations will help remove the small false-positive skin regions in the image.

From there, we smooth the mask slightly using a Gaussian blur on **Line 52**. This smoothing step, while not critical to the skin detection process, produces a much cleaner mask.

We then apply the skin mask to our frame on **Line 53**.

**Line 56** shows us a side-by-side view of the original frame along with the frame with skin detected in it.

We wait for a keypress on **Line 59**, and if it’s the q  key, then we break from the loop.

Finally, **Lines 63 and 64** release the reference to our webcam/video and close any open windows.

**Running our Skin Detector**

To run our skin detector, open up a terminal and navigate to where our source code is stored.

If you are using the example video provided with the code downloads for this post (or an example video of your own), then issue the following command:

|  |  |
| --- | --- |
| 1 | $ python skindetector.py --video video/skin\_example.mov |

Otherwise, if you want to use your own webcam, execute this command:

|  |  |
| --- | --- |
| 1 | $ python skindetector.py |

If all goes well, you should see my detected skin, as shown by the following figure:

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/skin_detection_face.jpg)

**Figure 1:** Detecting the skin of my face using Python and OpenCV.

And here is another example image from the video supplied with the code download:

[](https://www.pyimagesearch.com/wp-content/uploads/2014/08/skin_detection_hand.jpg)

**Figure 2:** Detecting both the skin of my and and the skin of my face using Python and OpenCV.

Pretty cool, right?

Using this basic approach we were able to build a fairly rudimentary skin detection algorithm — that’s still able to obtain decent results!

**Limitations**

There are some pretty obvious limitations and drawbacks to this approach.

The main drawback is that we are framing skin detection as a “color detection” problem. This assumes that we can easily specify the HSV values for ranges of pixel intensities that are considered skin.

However, under different lighting conditions, this approach might not perform as well — and we would likely have to continue to tweak the HSV value ranges.

The HSV value ranges we supplied worked fairly well for this example post…but what about for other ethnicities?

For example, I’m a white male. And surely the same HSV values used to detect my skin could not be used to detect someone from Asia or Africa. This implies that we have some *a priori*knowledge regarding the skin tone of who we want to detect.

Of course, more robust approaches can be applied. A (highly simplified) example would be to perform face detection to an image, determine the color of the skin on their face, and then use that model to detect the rest of the skin on their body.

Not a bad approach, but as you can imagine, it’s definitely a little more complicated.

In the meantime, as long as you can specify the HSV values of the skin tone you want to detect, framing skin detection as a color detection problem should work quite well for you.

**Summary**

In this blog post I showed you how to detect skin in images using Python and OpenCV.

To accomplish our skin detection, we framed *skin detection*as an extension to [*color detection*](https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/).

We were able to supply upper and lower ranges of pixel intensities in the HSV color space to detect skin in images.

While this is not a perfect or robust approach, the simplicity of our skin detection algorithm makes it a very good starting point to build more robust solutions.

But remember our promise, okay?

Don’t be like James. Use our skin detection superpowers for good, not evil.