We're going to cover how to create a sort of filter, revisiting the bitwise operations, where we will filter for specifically a certain color, attempting to just show it. Alternatively, you could also specifically filter out a specific color, and then replace it with a scene, like we did with replacing a ROI (region of image) with something else, much like how a green screen works.

In order to filter like this you have a few options. Generally, you are probably going to convert your colors to HSV, which is "Hue Saturation Value." This can help you actually pinpoint a more specific color, based on hue and saturation ranges, with a variance of value, for example. If you wanted, you could actually produce filters based on BGR values, but this would be a bit more difficult. If you're having a hard time visualizing HSV, don't feel silly, check out the Wikipedia page on [HSV](https://en.wikipedia.org/wiki/HSL_and_HSV), there is a very useful graphic there for you to visualize it. Hue for color, saturation for the strength of the color, and value for light is how I would best describe it personally. Now let's hop in.

https://upload.wikimedia.org/wikipedia/commons/1/16/Hsl-hsv\_models\_b.svg

import cv2

import numpy as np

cap = cv2.VideoCapture(0)

while(1):

\_, frame = cap.read()

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

lower\_red = np.array([30,150,50])

lower\_red = np.array([30,150,50])

upper\_red = np.array([255,255,180])

mask = cv2.inRange(hsv, lower\_red, upper\_red)

res = cv2.bitwise\_and(frame,frame, mask= mask)

cv2.imshow('frame',frame)

cv2.imshow('mask',mask)

cv2.imshow('res',res)

k = cv2.waitKey(5) & 0xFF

if k == 27:

break

cv2.destroyAllWindows()

cap.release()



This is just an example, with reds as the target. The way this works is what we see will be anything that is between our ranges here, basically 30-255, 150-255, and 50-180. This is a for a red, but feel free to try to find your own colors. The reason why HSV works best here is because we want a range of colors, and we generally want the same-ish color in this case. Many times, a typical red will still have some green and blue, so we would have to allow for some green and some blue, but then we'd want pretty much full red. This means we'd get lower-light mixes of all colors still at this point.

To figure the HSV range, I believe the best method is to just trial and error. There are built in methods to OpenCV to convert BGR to HSV. If you wanted to pick just a single color, then the BGR to HSV would be great to use. For the sake of teaching, here's an example of that code at work:

dark\_red = np.uint8([[[12,22,121]]])

dark\_red = cv2.cvtColor(dark\_red,cv2.COLOR\_BGR2HSV)

The result here will be an HSV value that is identical to the dark\_red value. This is great... but again... you run into the fundamental problem with ranges in colors vs ranges in HSV. They are just fundamentally different. You may have a legitimate use for BGR ranges, they will still work, but for detecting one "color," it wont work well.

Back to the main code, however, we have first that we convert the frames to HSV. Nothing too special there. Next, we specify some HSV values for the color red. We create a mask, which uses an "inRange" statement, for our specific range. This is true or false, black or white. Next, we "restore" our red-ness by running a bitwise operation. Basically, we show color where there is the frame AND the mask. The white part of the mask will be red range, that was converted to pure white, while everything else became black. Finally we show it all. I chose to show the original frame, the mask, and the final result, so you can get a better idea of what is happening.