

```

*****
*
*          IMAGE PROCESSING (e-Yantra 2014)
*          =====
*   This software is intended to teach image processing concepts
*
*   MODULE: Functions
*   Filename: Masking.pdf
*   Version: 1.0.0
*   Date: November 3, 2014
*
*   Author: Arun Mukundan, e-Yantra Project, Department of Computer Science
*   and Engineering, Indian Institute of Technology Bombay.
*
*   Software released under Creative Commons CC BY-NC-SA
*
*   For legal information refer to:
*   http://creativecommons.org/licenses/by-nc-sa/4.0/legalcode
*
*   This software is made available on an "AS IS WHERE IS BASIS".
*   Licensee/end user indemnifies and will keep e-Yantra indemnified from
*   any and all claim(s) that emanate from the use of the Software or
*   breach of the terms of this agreement.
*
*   e-Yantra - An MHRD project under National Mission on Education using
*   ICT (NMEICT)
*
*****

```

Masking

Masking is similar to Thresholding, but uses both upper and lower thresholds on all the channels, i.e. we are not constrained to use a grayscale image. Further, the pixels that lie within the range specified are set to 255 and the rest to 0. Once we have done this, suppose we need to retain the value of the pixels that lie within the range we want instead of setting them to 255, then we can use bitwise operations. An example is given below

Masking:

```
mask = cv2.inRange(hsv, lower, upper)
```

mask → Returned image

src → The source image

lower → The intended lower threshold

upper → The intended upper threshold

Bitwise And:

```
dst = cv2.bitwise_and(src1, src2, dst, mask)
```

dst → Returned image

src1 → The first source image

src2 → The second source image

mask → The intended mask

Typical Usage:

param1 = [80,100,150]

param2 = [130,200,255]

lower = numpy.array(param1) *##Convert to a form OpenCV can understand*

upper = numpy.array(param2)

mask = cv2.inRange(src, lower, upper)

res = cv2.bitwise_and(frame, frame, mask = mask)