Day 8: Color detection using HSV

Outcomes:

- Detect a specific color in an image
- Use **HSV ranges** to isolate colors
- Create and apply masks
- Understand how color-based segmentation works

Why HSV for color detection?

- In BGR, color changes with lighting.
- In HSV,
    - Hue represents actual color,
    - more stable under lighting changes.

That’s why most color detection tasks are done in HSV

Color Detection

Converting Image to HSV:

```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

Define Color Range (example: Blue)

```
import numpy as np


lower_blue = np.array([100, 150, 50])

upper_blue = np.array([140, 255, 255])
```

- Hue: 100 – 140, pixels within this range are considered blue
- Saturation: 150 – 255, ignores gray/faded colors
- Value: 50 – 255, ignores very dark pixels

This helps detect all pixels that fall between these ranges and is visibly blue enough.

Picture this: A pixel whose hue value is 100 but low saturation and value means it is a weak blue or simply is not blue enough.

Beginner Rule of Thumb

- **Hue** → choose the color
- **Saturation** → remove gray/weak colors
- **Value** → remove dark noise

Create a Mask

```
mask = cv2.inRange(hsv, lower_blue, upper_blue)
```

Mask is a binary image.

- White (255) → color present
- Black (0) → color absent

Apply Mask to Original Image

```
result = cv2.bitwise_and(img, img, mask=mask)
```

- Keeps only detected color
- Everything else becomes black

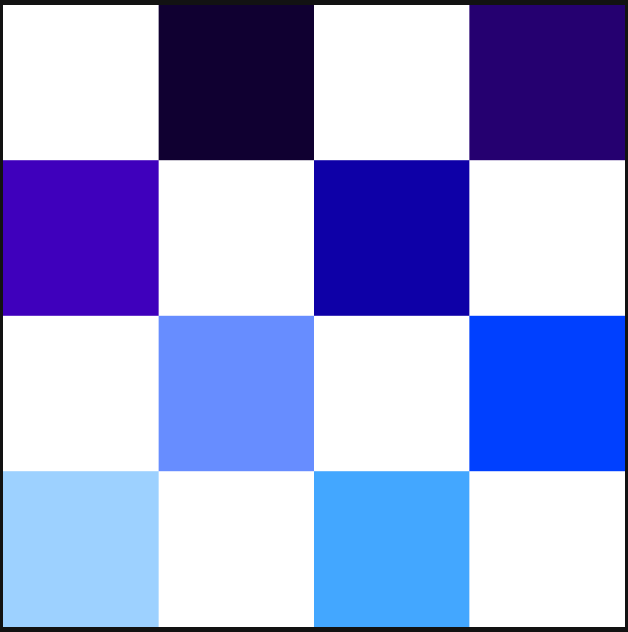
Display Results

```
cv2.imshow("Original", img)

cv2.imshow("Mask", mask)

cv2.imshow("Result", result)


cv2.waitKey(0)

cv2.destroyAllWindows()
```

13-colorDetection.py

```python
import cv2 as cv

import numpy as np


img1 = cv.imread(“OpenCV-codes/images/image2.png”)

img = cv.resize(img1, (480,480))


hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)


lower_blue = np.array([100, 150, 50])

upper_blue = np.array([140, 255, 255])


mask = cv.inRange(hsv, lower_blue, upper_blue)


result = cv.bitwise_and(img, img, mask=mask)


cv.imshow(“Original”, img)

cv.imshow(“Mask”, mask)

cv.imshow(“Result”, result)


cv.waitKey(0)

cv.destroyAllWindows()
```

Original: (image2.png)



Details

Shape of the image: (980, 980, 3)

Purpose: To illustrate how color detection detects color using a range.

This image contains very light to very dark shades of blue.

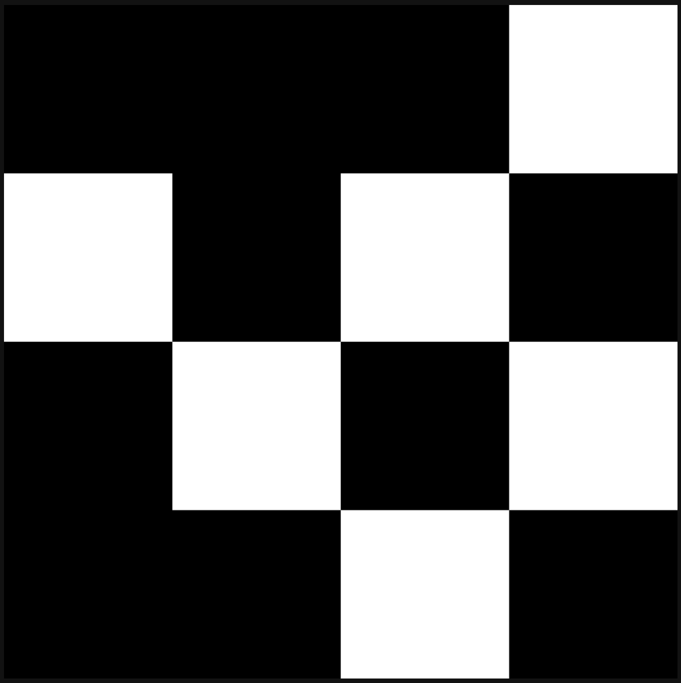
We want to see if OpenCV detects all or some of these shades.

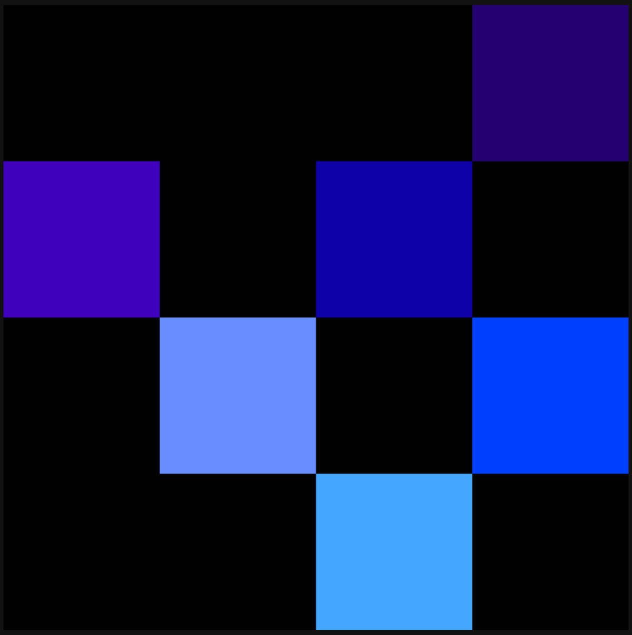
Output:

Mask:

The mask is a binary image where pixels falling within the blue HSV range are shown in white, and all other pixels are black.

Notice how it shows exactly which parts of the image OpenCV considers as blue based on the chosen HSV thresholds.





Result:

The result image keeps only the blue regions from the original image using the mask, while all other areas are turned black.

This visually confirms how accurately the chosen HSV range isolates the target color.

Functions used in the program

`cv2.inRange()` – Create a mask

It is a **thresholding** function.

It classifies pixels based on whether their values fall inside a specified range.

Syntax:

```
mask = cv2.inRange(img, lower_bound, upper_bound)
```

It looks at the `img`

```
if pixel value >= lower_bound AND
pixel value <=upper_bound
→ output pixel = white
else output pixel = black
```

Hence the result is always a binary image.

`cv2.bitwise_and()` – logical operation

- cv2.bitwise_and() is a pixel-wise logical operation.
- It performs an **AND** operation on two images.

Syntax:

```
cv2.bitwise_and(imgA, imgB, mask=…)
```

- In a simple language:

  "Keep pixel data only where a condition is true."

- That condition is usually provided via a mask (which is optional)
- What it does internally:
  - Takes pixel from image A
  - Takes pixel from image B
  - Applies **AND** operation
  - If a mask is given, the operation only applies where mask ≠ 0
  - Else the output is black
- Input & Output:
  - Input: Two images + mask (optional)
  - Output: Image after logical filtering.

HSV Range Tuning Tip

If detection is bad:

- Increase Saturation lower bound
- Adjust Hue range slightly
- Lighting matters

Core insight:

Color Detection = Convert → Threshold → Mask → Apply

This pipeline is used in

- Object tracking
- Traffic signal detection
- Gesture recognition