Day 9: Trackbars for Real-time HSV tuning

Outcomes:

- Create trackbars in OpenCV
- Adjust HSV values in real time
- Tune color detection interactively
- Understand how real-time parameter control works

Trackbars

What are Trackbars?

Trackbars are **interactive sliders** used in programs (very common in OpenCV) to **change values in real-time** while your program is running.

They let you **tune numbers visually instead of hard coding them**.

In simple language,

> "A trackbar is a slider that lets you change a number while the program is running."

Why trackbars?

Hard-coding HSV values is slow and inaccurate.

Trackbars let you:

- Slide values
- See results instantly
- Find perfect HSV ranges

This is essential for real-world color detection.

When detecting colors in HSV:

- You need **6 numbers** (LH, LS, LV, UH, US, UV)
- These numbers **change with lighting**
- Writing numbers, running code, changing again is painful

Trackbars solve this by letting you **slide values and see changes instantly**.

Creating a Window

Syntax:

```
cv2.namedWindow("windowName")
```

Example:

```
cv2.namedWindow("Trackbars")
```

What this does:

- Creates an empty window
- This window is just a **container**
- Trackbars must be attached to a window (no window = no trackbars).

Creating ONE Trackbar (breakdown of `cv2.createTrackbar()`)

Syntax of `cv2.createTrackbar()`:

```
cv2.createTrackbar(trackbarName, windowName, value, maxValue, onChange)
```

`trackbarName`	Name shown on the slider
`windowName`	Window where the slider appears
`value`	Initial (starting value)
`maxValue`	Maximum value of the slider
`onChange`	Function called when slider moves

Purpose of the line:

Create a slider called LH that goes from `value` to `maxValue`.

The confusing `nothing(x)` function

```
def nothing(x):
    pass
```

Why does this exist?

OpenCV forces every trackbar to have a callback function.

But we don't need it.

So, we give it a **dummy function** that does nothing.

Creating All Trackbars

```
def nothing(x):
    pass

cv2.createTrackbar("LH", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("LS", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("LV", "Trackbars", 0, 255, nothing)

cv2.createTrackbar("UH", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("US", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("UV", "Trackbars", 255, 255, nothing)
```

`LH` → Hue minimum
`LS` → Saturation minimum
`LV` → Value minimum

`UH` → Hue maximum
`US` → Saturation maximum
`UV` → Value maximum

Reading ONE Trackbar (breakdown of `cv2.getTrackbarPos()`)

Syntax:

```
cv2.getTrackbarPos(trackbarName, windowName)
```

Return type: `int`

`Example:`

```
lh = cv2.getTrackbarPos("LH", "Trackbars")
```

What it does:

- Goes to the window called `Trackbars`
- Finds the slider `LH`
- Reads its current position

Why a Loop is Mandatory

```
while True:
    lh = cv2.getTrackbarPos("LH", "Trackbars")
```

Trackbars change continuously when you drag them.

Without the loop:

- Value is read only once
- Slider movement does nothing

Hence, we need a loop to **continuously check** the position of the trackbar.

Reading All Trackbar Values

```
lh = cv2.getTrackbarPos("LH", "Trackbars")
ls = cv2.getTrackbarPos("LS", "Trackbars")
lv = cv2.getTrackbarPos("LV", "Trackbars")

uh = cv2.getTrackbarPos("UH", "Trackbars")
us = cv2.getTrackbarPos("US", "Trackbars")
uv = cv2.getTrackbarPos("UV", "Trackbars")
```

Thresholding

```
lower = np.array([lh, ls, lv])
upper = np.array([uh, us, uv])
```

These values change the moment a slider is moved.

Apply Real-Time Masking

```
lower = np.array([lh, ls, lv])
upper = np.array([uh, us, uv])

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

It is kept inside a loop to update continuously.

Mental-Structure of the Loop

```
while True:
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    #read trackbars
    #create mask
    #apply mask

    cv2.imshow("Result", result)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

14-trackbars.py

```python
import cv2 as cv
import numpy as np

img = cv.imread("OpenCV-codes/images/image3.png")

# Convert into HSV
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)

# Create a Window
cv.namedWindow("Trackbars")

# Create Dummy Function
def nothing(x):
    pass

# Create 6 trackbars
cv.createTrackbar("LH", "Trackbars", 0, 179, nothing)
cv.createTrackbar("LS", "Trackbars", 0, 255, nothing)
cv.createTrackbar("LV", "Trackbars", 0, 255, nothing)

cv.createTrackbar("UH", "Trackbars", 179, 179, nothing)
cv.createTrackbar("US", "Trackbars", 255, 255, nothing)
cv.createTrackbar("UV", "Trackbars", 255, 255, nothing)

# Get Trackbar Position
while True:
    lh = cv.getTrackbarPos("LH", "Trackbars")
```

```
    ls = cv.getTrackbarPos("LS", "Trackbars")
    lv = cv.getTrackbarPos("LV", "Trackbars")

    uh = cv.getTrackbarPos("UH", "Trackbars")
    us = cv.getTrackbarPos("US", "Trackbars")
    uv = cv.getTrackbarPos("UV", "Trackbars")

    # Updating thresholds
    lower_bound = np.array([lh, ls, lv])
    upper_bound = np.array([uh, us, uv])

    # Masking
    mask = cv.inRange(hsv, lower_bound, upper_bound)
    result = cv.bitwise_and(img, img, mask=mask)

    # Displaying the results
    cv.imshow("mask", mask)
    cv.imshow("result", result)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break


cv.waitKey(0)
cv.destroyAllWindows()
```

Note: Image is static, so HSV conversion is done once, whereas while working with videos, it should be done inside the loop.

Original image (image3.png)

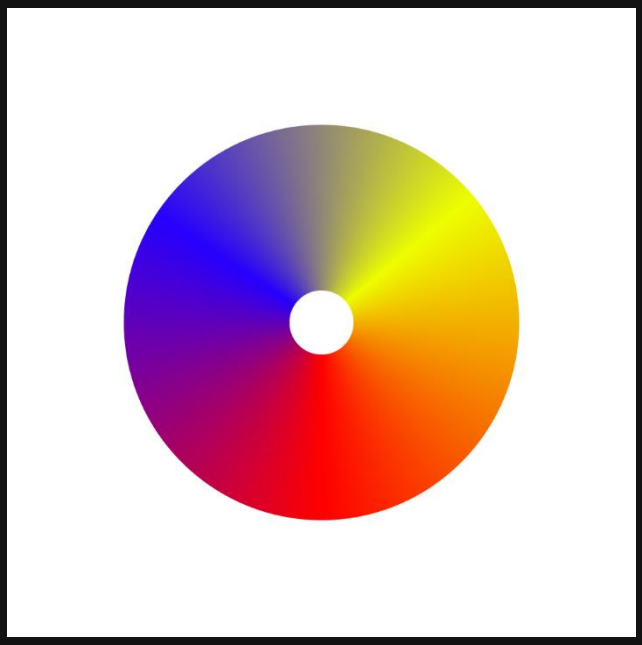


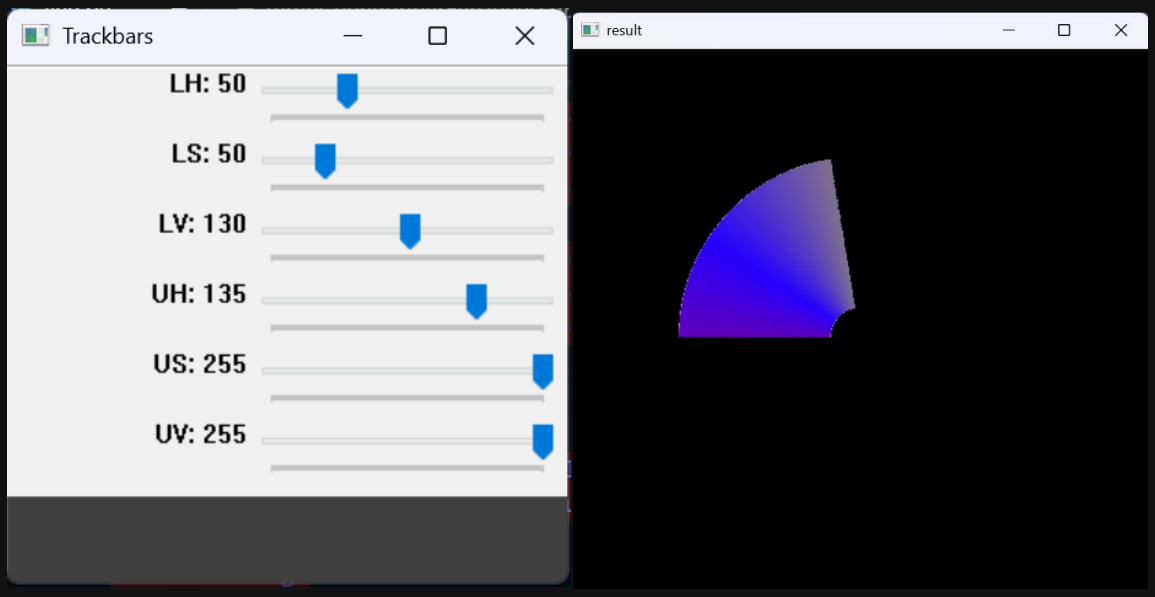
Image Details:

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

Purpose: To illustrate how thresholding can be done in real-time using trackbars.

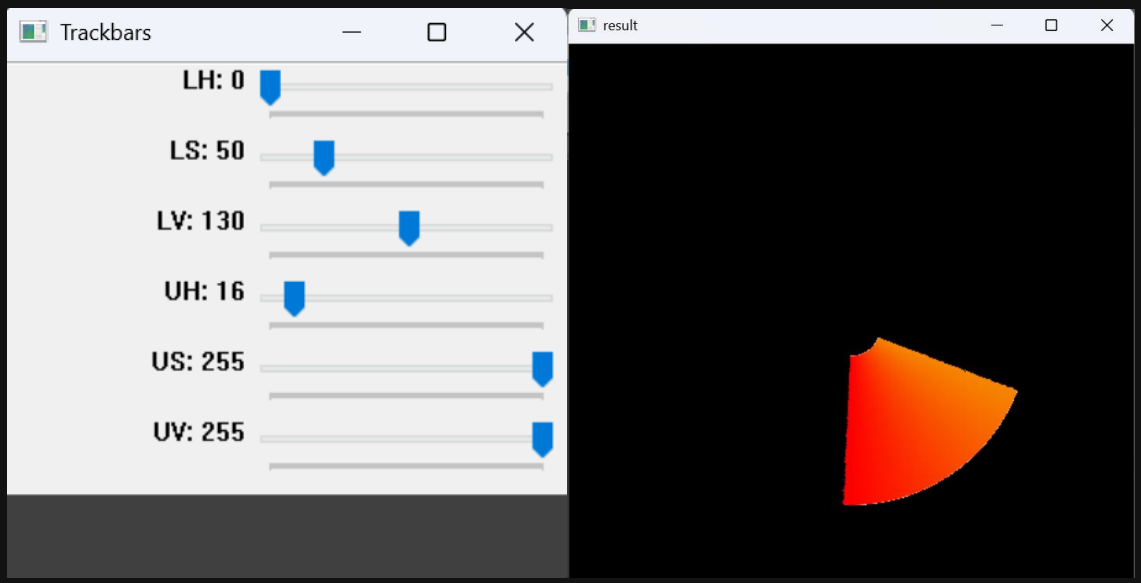
Output

Trackbar Values and the Corresponding Results



It seems that a Hue range of approximately **50-135** isolates blue regions in the image effectively.

Similarly, it seems that a Hue range of **0–16** isolates red regions in the image.



Note: These values are approximate, image-dependent observations rather than fixed color thresholds.
Reason: HSV ranges vary with lighting, camera characteristics, and exposure, so they are not globally applicable.

Conclusion

As the trackbar values are adjusted, the HSV thresholds change in real time.

This causes the mask and result image to update instantly, allowing us to visually find the most accurate HSV range for the target color under current lighting conditions.