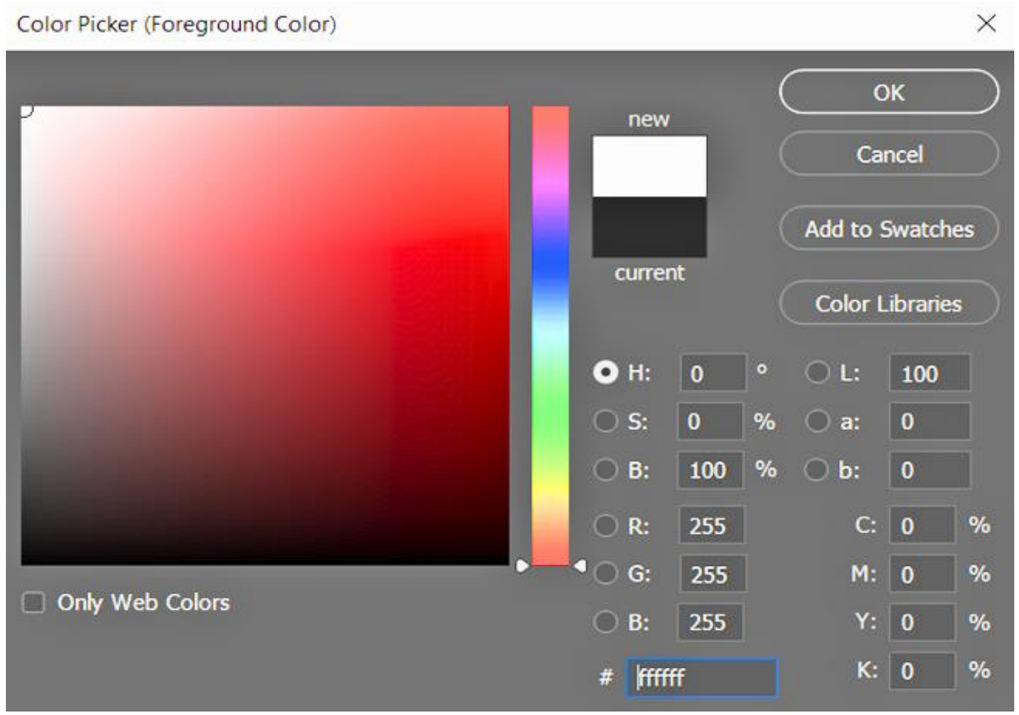
HSV Adjustment Tool

Computer Vision - Assignment 1

Group 3:
Abdelrahman Mohamed (12500270)
Ahmed Hassan (22404506)

Why HSV instead of RGB?

- **RGB (Red, Green, Blue):**
  - Additive color model used by screens and cameras.
  - Represents colors by mixing intensities of red, green, and blue channels.
- **HSV (Hue, Saturation, Value):**
  - Hue: The color type (0–360°).
  - Saturation: Color purity or intensity (0–100%).
  - Value: Brightness (0–100%).
- HSV better matches human color perception.
- Easier to adjust colors meaningfully (e.g., change brightness without changing the color).



Technical Background

- **RGB to HSV Conversion:**
  - Done using OpenCV’s cv2.cvtColor() function.
  - Converts image into Hue, Saturation, Value channels.

- **Hue and Value Adjustment:**
  - Hue shifted by a configurable angle (0–360°).
  - Value adjusted using gamma correction (0.5–4.0x).

- **HSV to RGB Conversion:**
  - Implemented using two methods:
    - Loop-based method (pixel-by-pixel).
    - Matrix-based method (using NumPy operations).

- **GUI Design:**
  - Built with Qt Designer’s drag-and-drop interface.
  - PyQt6 used for dynamic interaction with images.

Code Highlights

Static Program – StaticProgram.py:

- Simple CLI-based implementation.
- Accepts input image, hue shift, value exponent, and outputs result.

GUI Application – GUIProgram.py:

- Optimization techniques reduce computational time by 30%. PyQt6-based graphical interface.
- Allows real-time adjustments of Hue and Value.

Code Documentation:

- Fully documented with Doxygen comments.
- HTML documentation generated for both programs.

Special Features:

- Input image handling.
- Output saving (PNG/JPG).
- GUI real-time sliders.
- Performance benchmarking (Loop vs Matrix).

Code Highlights (Code Previews)

HSV ➔ RGB Loop-based Conversion

```
hsv_img = hsv_img.astype(np.float32)
height, width = hsv_img.shape[:2]
rgb_img = np.zeros((height, width, 3), dtype=np.uint8)

for i in range(height):
    for j in range(width):
        h, s, v = hsv_img[i, j]
        # Scale hue from OpenCV's 0-180 range to 0-360 degrees
        h = (h * 2.0) / 60.0
        s = s / 255.0
        v = v / 255.0
        c = v * s
        x = c * (1 - abs((h % 2) - 1))
        m = v - c

        if 0 <= h < 1:
            r, g, b = c, x, 0
        elif 1 <= h < 2:
            r, g, b = x, c, 0
        elif 2 <= h < 3:
            r, g, b = 0, c, x
        elif 3 <= h < 4:
            r, g, b = 0, x, c
        elif 4 <= h < 5:
            r, g, b = x, 0, c
        else:
            r, g, b = c, 0, x

        rgb_img[i, j] = [(r + m) * 255, (g + m) * 255, (b + m) * 255]
```

HSV ➔ RGB Matrix-based Conversion

```
hsv_img = hsv_img.astype(np.float32)
# Scale hue from OpenCV's 0-180 range to 0-360 degrees
h, s, v = hsv_img[:, :, 0] * 2.0, hsv_img[:, :, 1], hsv_img[:, :, 2]
h = h / 60.0
s = s / 255.0
v = v / 255.0
c = v * s
x = c * (1 - np.abs((h % 2) - 1))
m = v - c

# Initialize RGB with zeros
rgb = np.zeros_like(hsv_img)

# Create a zeros array for the third channel with proper dimensions
zeros = np.zeros_like(c)

# Apply the conversion for each hue region
mask = (h >= 0) & (h < 1)
r, g, b = c[mask], x[mask], zeros[mask]
rgb[mask, 0], rgb[mask, 1], rgb[mask, 2] = r, g, b

mask = (h >= 1) & (h < 2)
r, g, b = x[mask], c[mask], zeros[mask]
rgb[mask, 0], rgb[mask, 1], rgb[mask, 2] = r, g, b

mask = (h >= 2) & (h < 3)
r, g, b = zeros[mask], c[mask], x[mask]
rgb[mask, 0], rgb[mask, 1], rgb[mask, 2] = r, g, b

mask = (h >= 3) & (h < 4)
r, g, b = zeros[mask], x[mask], c[mask]
rgb[mask, 0], rgb[mask, 1], rgb[mask, 2] = r, g, b

mask = (h >= 4) & (h < 5)
r, g, b = x[mask], zeros[mask], c[mask]
rgb[mask, 0], rgb[mask, 1], rgb[mask, 2] = r, g, b

mask = (h >= 5)
r, g, b = c[mask], zeros[mask], x[mask]
rgb[mask, 0], rgb[mask, 1], rgb[mask, 2] = r, g, b

# Add the m value to each channel
rgb[:, :, 0] += m
rgb[:, :, 1] += m
rgb[:, :, 2] += m
```

Convert RGB to HSV (OpenCV built-in)

```
# Load and process the image
self.original_image = cv2.imread(self.image_path)
self.original_image = cv2.cvtColor(self.original_image, cv2.COLOR_BGR2RGB)
self.adjusted_image = self.original_image.copy()
self.hsv_image = cv2.cvtColor(self.original_image, cv2.COLOR_RGB2HSV).astype(np.float32)
self.current_hsv = self.hsv_image.copy()
self.show_images()
```

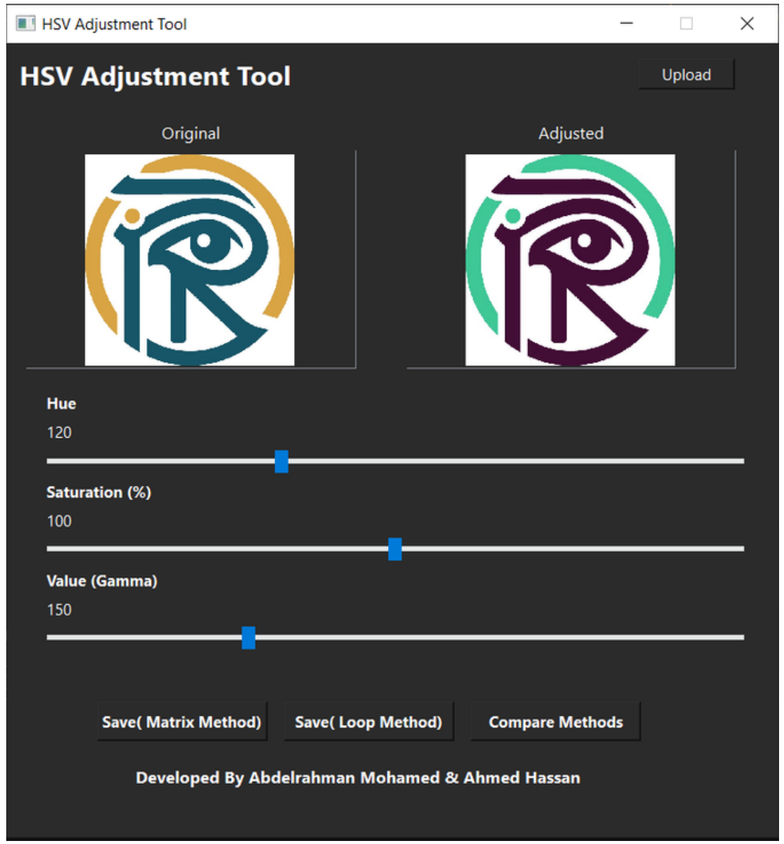
Edit Hue and Value Channels

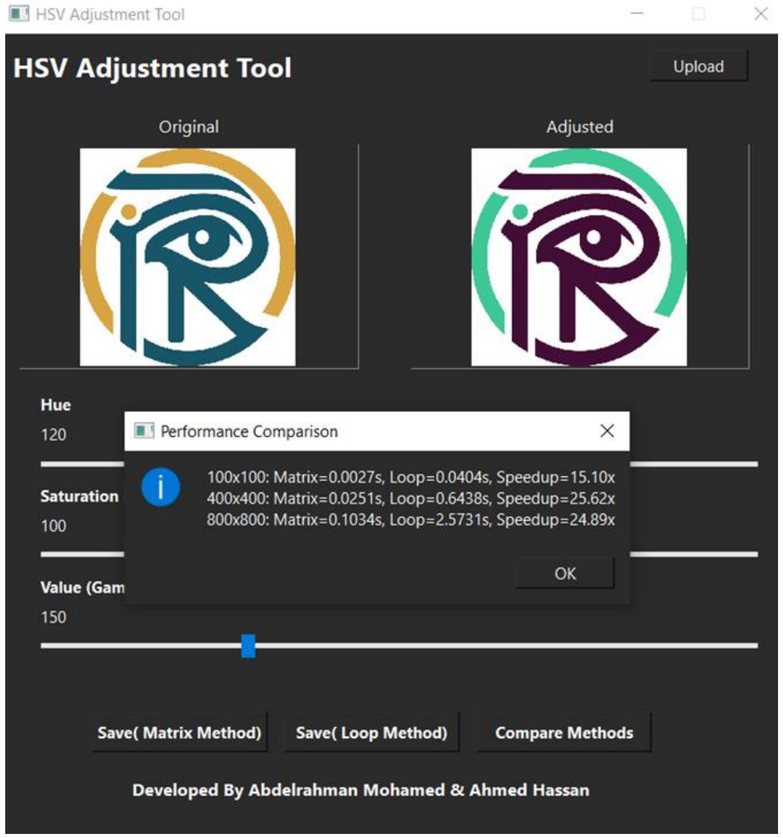
```
hue_shift = self.HueSlider.value() / 2 # OpenCV uses 0-180 for hue
saturation_factor = self.SaturatedSlider.value() / 100.0
gamma = self.ValueSlider.value() / 100.0

current_hsv = self.hsv_image.copy()

current_hsv[:, :, 0] = (current_hsv[:, :, 0] + hue_shift) % 180  # OpenCV uses 0-180 for hue

current_hsv[:, :, 1] = np.clip(current_hsv[:, :, 1] * saturation_factor, 0, 255)


normalized_value = current_hsv[:, :, 2] / 255.0
modified_value = np.power(normalized_value, gamma) * 255.0
current_hsv[:, :, 2] = np.clip(modified_value, 0, 255)

self.current_hsv = current_hsv
self.adjusted_image = cv2.cvtColor(current_hsv.astype(np.uint8), cv2.COLOR_HSV2RGB)
```

Performance Comparison

```
# Time matrix method
start = time.time()
_ = self.hsv_to_rgb_matrix(test_hsv)
matrix_time = time.time() - start

# Time loop method
start = time.time()
_ = self.hsv_to_rgb_loop(test_hsv)
loop_time = time.time() - start
```

Results & Performance





Input vs Output Images:

- Visual comparison after applying HSV adjustments.

GUI Application Dashboard:

- Real-time image preview with Hue and Value sliders.
- Easy upload and save functionality.

Performance Comparison:

- Matrix method is ~50x faster than Loop method.
- Benchmarked on images of different sizes.

Observations:

- Matrix-based operations are ideal for large images.
- Loop method mainly used for educational demonstration.

Thank You!