符昕宇 202364810311

To access my code more easily, you can find them at my github repository:

https://github.com/Fluorine-Brian/Code-for-Digital-Image-Processing

Homework1: Implement Figure 6.29 by programming with Python

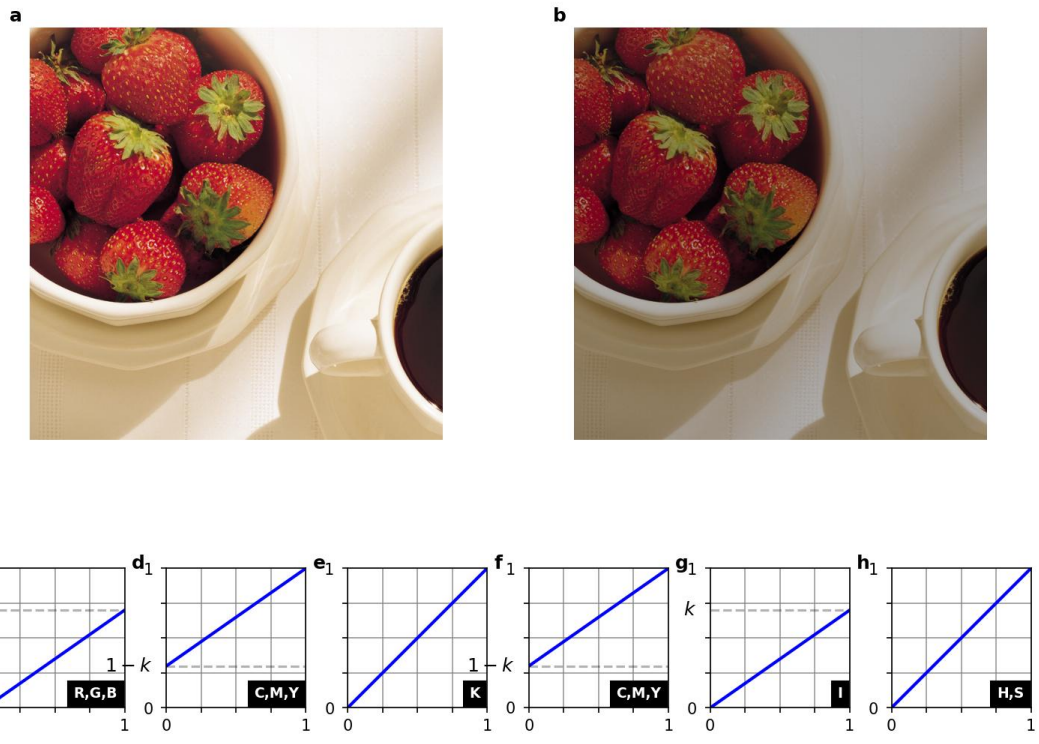
Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os
import matplotlib.gridspec as gridspec

def adjust_intensity_rgb(image, k):
    img_float = image.astype(float) / 255.0
    processed = img_float * k
    processed = np.clip(processed, 0, 1)
    return (processed * 255).astype(np.uint8)

def plot_mapping_function(ax, slope=None, intercept=None, label_text="", k_val=0.7,
subplot_label=""):
    x = np.linspace(0, 1, 100)
    if slope is not None and intercept is not None:
        y = slope * x + intercept
    elif slope is not None:
        y = slope * x
    else:
        y = x
    y = np.clip(y, 0, 1)

    ax.plot(x, y, 'b-', linewidth=2)
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.set_xticks(np.linspace(0, 1, 5))
    ax.set_yticks(np.linspace(0, 1, 5))
    ax.tick_params(axis='both', which='major', labelsize=10)
    ax.grid(True, linestyle='-', alpha=0.8, color='gray', which='major')
    ax.set_aspect('equal')
    if k_val is not None:
        if "R,G,B" in label_text or label_text == "I":
            ax.plot([0, 1], [k_val, k_val], 'k--', alpha=0.3)
            ax.text(-0.1, k_val, r'$k$', va='center', ha='right', fontsize=12)
        elif "C,M,Y" in label_text and intercept is not None and intercept > 0:
```

```
            val = 1 - k_val
            ax.plot([0, 1], [val, val], 'k--', alpha=0.3)
            ax.text(-0.1, val, r'$1-k$', va='center', ha='right', fontsize=12)
    ax.set_xticklabels([0, '', '', '', 1])
    ax.set_yticklabels([0, '', '', '', 1])
    ax.tick_params(axis='y', pad=5)
    ax.text(0.95, 0.05, label_text, transform=ax.transAxes,
            ha='right', va='bottom', color='white', backgroundcolor='black',
fontsize=9, fontweight='bold')
    ax.text(-0.25, 1.1, subplot_label, transform=ax.transAxes,
            ha='left', va='top', fontsize=12, fontweight='bold')

def plot_image_subplot(ax, image_data, subplot_label=""):
    ax.imshow(image_data)
    ax.axis('off')
    ax.text(-0.05, 1.05, subplot_label, transform=ax.transAxes,
            ha='left', va='top', fontsize=12, fontweight='bold')

if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)
    image_filename = "Fig0630(01)(strawberries_fullcolor).tif"
    image_path = os.path.join(input_dir, image_filename)
    base_name = os.path.splitext(image_filename)[0]

    original_image = imageio.imread(image_path)
    if len(original_image.shape) == 2:
        original_image = np.stack((original_image,) * 3, axis=-1)
    elif original_image.shape[2] == 4:
        original_image = original_image[:, :, :3]
    k = 0.7
    result_image = adjust_intensity_rgb(original_image, k)

    fig = plt.figure(figsize=(12, 8))
    gs = gridspec.GridSpec(2, 6, figure=fig, hspace=0.4, wspace=0.3,
height_ratios=[2.5, 1])

    ax_img1 = fig.add_subplot(gs[0, 0:3])
    plot_image_subplot(ax_img1, original_image, subplot_label="a")

    ax_img2 = fig.add_subplot(gs[0, 3:6])
    plot_image_subplot(ax_img2, result_image, subplot_label="b")
```

```
    ax_f1 = fig.add_subplot(gs[1, 0])
    plot_mapping_function(ax_f1, slope=k, intercept=0, label_text="R,G,B", k_val=k,
subplot_label="c")
    ax_f2 = fig.add_subplot(gs[1, 1])
    plot_mapping_function(ax_f2, slope=k, intercept=1 - k, label_text="C,M,Y",
k_val=k, subplot_label="d")
    ax_f3 = fig.add_subplot(gs[1, 2])
    plot_mapping_function(ax_f3, slope=1, intercept=0, label_text="K", k_val=k,
subplot_label="e")
    ax_f4 = fig.add_subplot(gs[1, 3])
    plot_mapping_function(ax_f4, slope=k, intercept=1 - k, label_text="C,M,Y",
k_val=k, subplot_label="f")
    ax_f5 = fig.add_subplot(gs[1, 4])
    plot_mapping_function(ax_f5, slope=k, intercept=0, label_text="I", k_val=k,
subplot_label="g")
    ax_f6 = fig.add_subplot(gs[1, 5])
    plot_mapping_function(ax_f6, slope=1, intercept=0, label_text="H,S", k_val=k,
subplot_label="h")
    save_path = os.path.join(output_dir, f"{base_name}_fig629_final.png")
    plt.savefig(save_path, bbox_inches='tight', dpi=150)
    plt.close(fig)
```

Processed Images:



Homework2: Implement Figure 6.31 by programming with Python

Python Code:

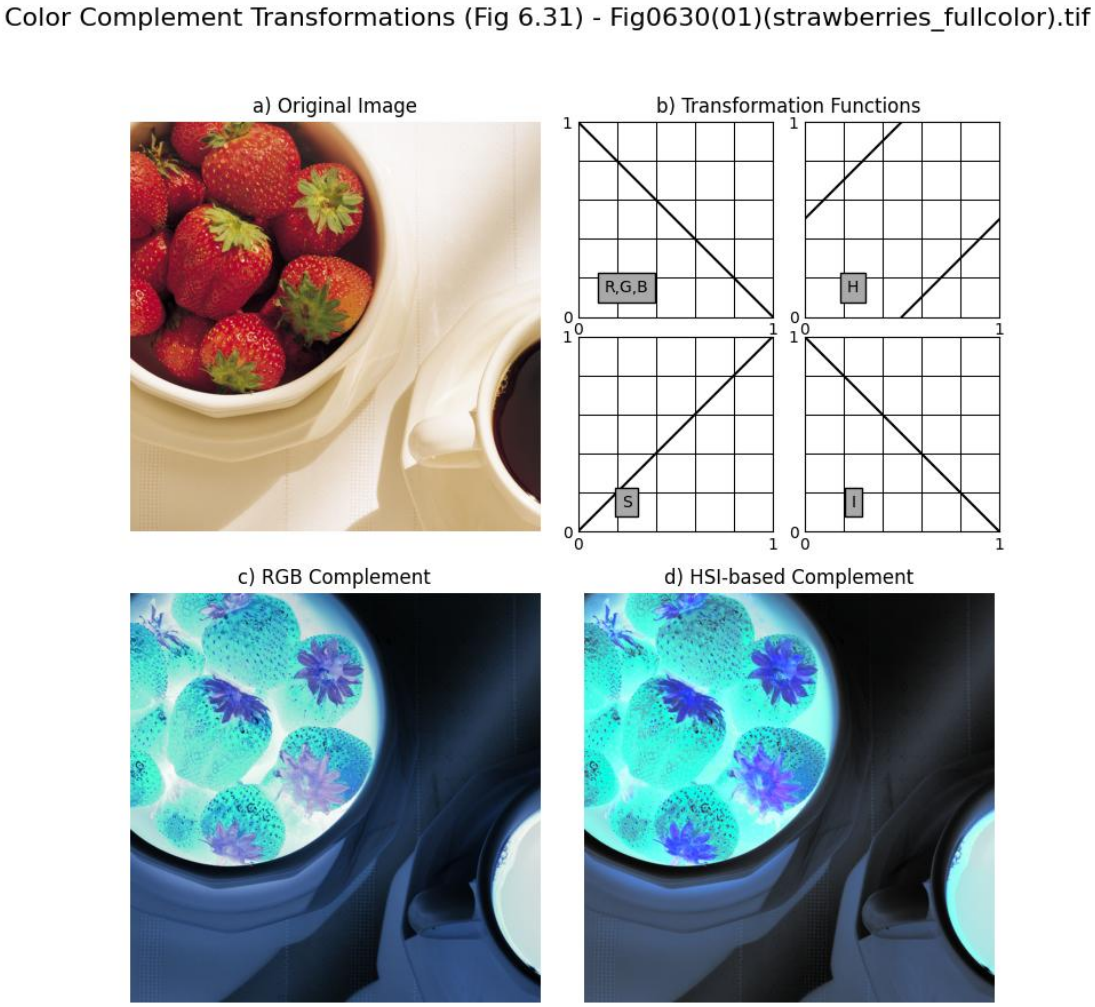
```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os
import matplotlib.gridspec as gridspec

def rgb_complement(image):
    return 255 - image

def rgb_to_hsi(rgb_image):
    img_float = rgb_image.astype(float) / 255.0
    R, G, B = img_float[:, :, 0], img_float[:, :, 1], img_float[:, :, 2]
    H = np.zeros_like(R)
    S = np.zeros_like(R)
    I = (R + G + B) / 3.0
    min_rgb = np.minimum(np.minimum(R, G), B)
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", RuntimeWarning)
        S = 1 - (3.0 / (R + G + B + 1e-6)) * min_rgb
    num = 0.5 * ((R - G) + (R - B))
    den = np.sqrt((R - G) ** 2 + (R - B) * (G - B))
    theta = np.arccos(np.clip(num / (den + 1e-6), -1, 1))
    theta = np.degrees(theta)
    H[B <= G] = theta[B <= G]
    H[B > G] = 360 - theta[B > G]
    H[S == 0] = 0
    hsi_image = np.stack([H, S, I], axis=2)
    return hsi_image

def hsi_to_rgb(hsi_image):
    H, S, I = hsi_image[:, :, 0], hsi_image[:, :, 1], hsi_image[:, :, 2]
    R, G, B = np.zeros_like(H), np.zeros_like(H), np.zeros_like(H)

    idx = (H >= 0) & (H < 120)
    B[idx] = I[idx] * (1 - S[idx])
    R[idx] = I[idx] * (1 + (S[idx] * np.cos(np.radians(H[idx]))) /
np.cos(np.radians(60 - H[idx])))
    G[idx] = 3 * I[idx] - (R[idx] + B[idx])
    idx = (H >= 120) & (H < 240)
    H_prime = H[idx] - 120
    R[idx] = I[idx] * (1 - S[idx])
```

```
    G[idx] = I[idx] * (1 + (S[idx] * np.cos(np.radians(H_prime))) /
np.cos(np.radians(60 - H_prime)))
    B[idx] = 3 * I[idx] - (R[idx] + G[idx])
    idx = (H >= 240) & (H < 360)
    H_prime = H[idx] - 240
    G[idx] = I[idx] * (1 - S[idx])
    B[idx] = I[idx] * (1 + (S[idx] * np.cos(np.radians(H_prime))) /
np.cos(np.radians(60 - H_prime)))
    R[idx] = 3 * I[idx] - (G[idx] + B[idx])
    rgb_image = np.stack([R, G, B], axis=2)
    rgb_image = np.clip(rgb_image * 255, 0, 255).astype(np.uint8)

    return rgb_image

def hsi_transform_from_book(image):
    hsi_img = rgb_to_hsi(image)
    H, S, I = hsi_img[:, :, 0], hsi_img[:, :, 1], hsi_img[:, :, 2]
    H_transformed = (H + 180) % 360
    S_transformed = S
    I_transformed = 1.0 - I
    hsi_transformed = np.stack([H_transformed, S_transformed, I_transformed],
axis=2)
    return hsi_to_rgb(hsi_transformed)

def plot_transform_functions(fig, spec):
    """
    Plots the four transformation functions
    """
    gs_nested = gridspec.GridSpecFromSubplotSpec(2, 2, subplot_spec=spec,
wspace=0.1, hspace=0.1)
    x = np.linspace(0, 1, 100)
    plots_def = {
        '00': {'type': 'rgb', 'label': 'R,G,B'},
        '01': {'type': 'h', 'label': 'H'},
        '10': {'type': 's', 'label': 'S'},
        '11': {'type': 'i', 'label': 'I'}
    }
    for i in range(2):
        for j in range(2):
            ax = fig.add_subplot(gs_nested[i, j])
            key = f'{i}{j}'
            p_def = plots_def[key]
            if p_def['type'] == 'rgb' or p_def['type'] == 'i':
                ax.plot([0, 1], [1, 0], 'k')
```

```
            elif p_def['type'] == 'h':
                ax.plot([0, 0.5], [0.5, 1], 'k')
                ax.plot([0.5, 1], [0, 0.5], 'k')
            elif p_def['type'] == 's':
                ax.plot([0, 1], [0, 1], 'k')

            ax.set_aspect('equal', adjustable='box')
            ax.set_xlim(0, 1)
            ax.set_ylim(0, 1)
            ticks = np.linspace(0, 1, 6)
            ax.set_xticks(ticks)
            ax.set_yticks(ticks)
            ax.set_xticklabels(['0', '', '', '', '', '1'])
            ax.set_yticklabels(['0', '', '', '', '', '1'])
            ax.grid(True, color='black', linewidth=0.75)
            ax.tick_params(length=0)
            ax.text(0.25, 0.15, p_def['label'], ha='center', va='center',
                    bbox={'facecolor': 'darkgray', 'edgecolor': 'black', 'pad': 4})

if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)
    image_filename = "Fig0630(01)(strawberries_fullcolor).tif"
    image_path = os.path.join(input_dir, image_filename)
    original_image = imageio.imread(image_path)
    base_name = os.path.splitext(image_filename)[0]
    rgb_comp_image = rgb_complement(original_image)
    hsi_comp_image = hsi_transform_from_book(original_image)

    fig = plt.figure(figsize=(10, 10))
    fig.suptitle(f'Color Complement Transformations (Fig 6.31) - {image_filename}',
fontsize=16)
    gs_main = gridspec.GridSpec(2, 2, figure=fig, wspace=0.05, hspace=0.15)

    ax_a = fig.add_subplot(gs_main[0, 0])
    ax_a.imshow(original_image)
    ax_a.set_title('a) Original Image')
    ax_a.axis('off')
    plot_transform_functions(fig, gs_main[0, 1])
    ax_b_title = fig.add_subplot(gs_main[0, 1])
    ax_b_title.set_title('b) Transformation Functions')
    ax_b_title.axis('off')
    ax_c = fig.add_subplot(gs_main[1, 0])
```

```
    ax_c.imshow(rgb_comp_image)
    ax_c.set_title('c) RGB Complement')
    ax_c.axis('off')
    ax_d = fig.add_subplot(gs_main[1, 1])
    ax_d.imshow(hsi_comp_image)
    ax_d.set_title('d) HSI-based Complement')
    ax_d.axis('off')

    combined_output_path = os.path.join(output_dir,
f"{base_name}_complement_full_figure.png")
    plt.savefig(combined_output_path, bbox_inches='tight')
    plt.close()
```

Processed Images:



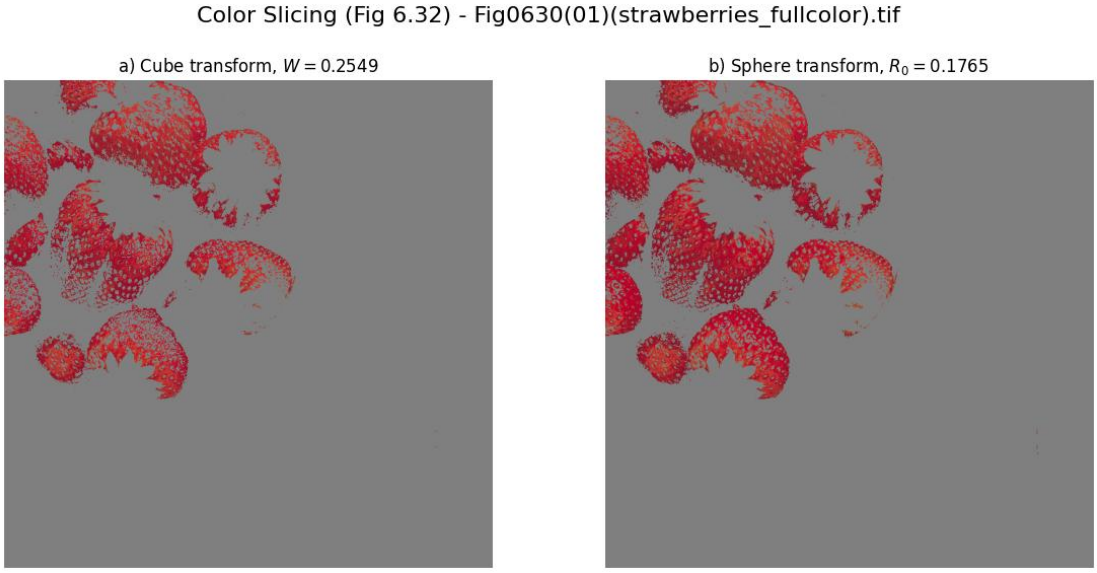
Homework3: Implement Figure 6.31 by programming with Python

Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os


def color_slicing_cube(image, center_color, width, neutral_color=(0.5, 0.5, 0.5)):
    img_float = image.astype(np.float64) / 255.0
    center = np.array(center_color)
    diff = np.abs(img_float - center)
    mask = np.any(diff > width / 2.0, axis=2)
    result_image = np.copy(image)
    neutral_pixel = (np.array(neutral_color) * 255).astype(np.uint8)
    result_image[mask] = neutral_pixel
    return result_image


def color_slicing_sphere(image, center_color, radius, neutral_color=(0.5, 0.5,
0.5)):
    img_float = image.astype(np.float64) / 255.0
    center = np.array(center_color)
    distances_sq = np.sum((img_float - center) ** 2, axis=2)
    mask = distances_sq > radius ** 2
    result_image = np.copy(image)
    neutral_pixel = (np.array(neutral_color) * 255).astype(np.uint8)
    result_image[mask] = neutral_pixel
    return result_image


if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)

    image_filename = "Fig0630(01)(strawberries_fullcolor).tif"
    image_path = os.path.join(input_dir, image_filename)
    original_image = imageio.imread(image_path)
    base_name = os.path.splitext(image_filename)[0]

    center_color_a = (0.6863, 0.1608, 0.1922)
    width_W = 0.2549
```

```
    radius_R0 = 0.1765

    cube_sliced_image = color_slicing_cube(original_image, center_color_a, width_W)
    sphere_sliced_image = color_slicing_sphere(original_image, center_color_a,
radius_R0)

    fig, axes = plt.subplots(1, 2, figsize=(12, 6))
    fig.suptitle(f'Color Slicing (Fig 6.32) - {image_filename}', fontsize=16)

    axes[0].imshow(cube_sliced_image)
    axes[0].set_title(f'a) Cube transform, $W={width_W}$')
    axes[0].axis('off')

    axes[1].imshow(sphere_sliced_image)
    axes[1].set_title(f'b) Sphere transform, $R_0={radius_R0}$')
    axes[1].axis('off')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])

    combined_output_path = os.path.join(output_dir,
f"{base_name}_color_slicing_results.png")
    plt.savefig(combined_output_path, bbox_inches='tight')
    plt.close()
```

Processed Images:



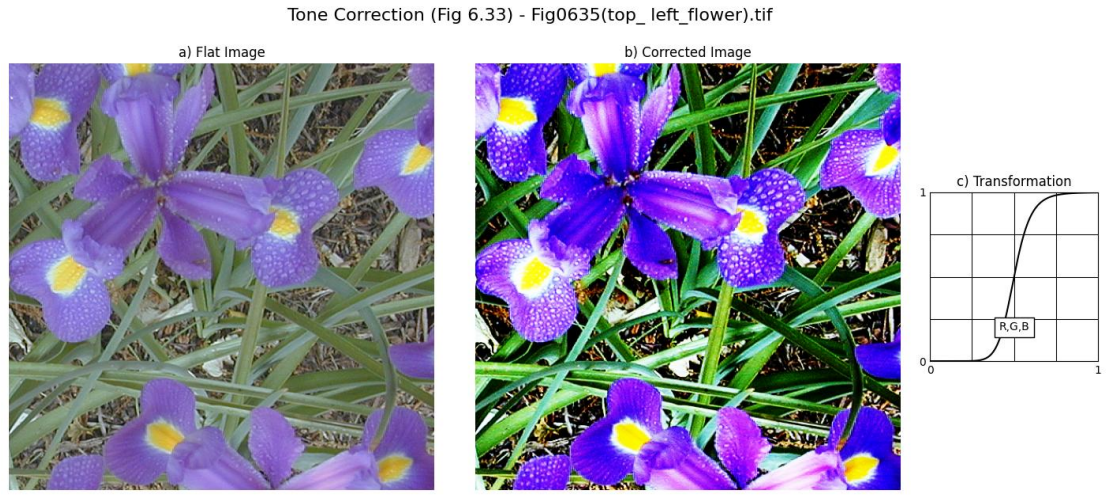
Homework4: Implement Figure 6.33 by programming with Python

Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os


def tone_correction(image, contrast, midpoint):
    img_float = image.astype(np.float64) / 255.0
    epsilon = 1e-6
    corrected_float = 1 / (1 + (midpoint / (img_float + epsilon)) ** contrast)
    corrected_image = np.clip(corrected_float * 255, 0, 255).astype(np.uint8)
    return corrected_image


def plot_tone_curve(ax, contrast, midpoint):
    r = np.linspace(0, 1, 256)
    epsilon = 1e-6
    s = 1 / (1 + (midpoint / (r + epsilon)) ** contrast)
    ax.plot(r, s, 'k')
    ax.set_aspect('equal', adjustable='box')
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ticks = np.linspace(0, 1, 5)  # For 4 squares
    ax.set_xticks(ticks)
    ax.set_yticks(ticks)
    ax.set_xticklabels(['0', '', '', '', '1'])
    ax.set_yticklabels(['0', '', '', '', '1'])
    ax.grid(True, color='black', linewidth=0.75)
    ax.tick_params(length=0)  # Remove tick marks
    ax.text(0.5, 0.2, 'R,G,B', ha='center', va='center',
            bbox={'facecolor': 'white', 'edgecolor': 'black', 'pad': 4})


if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)
    image_filename = "Fig0635(top_ left_flower).tif"
    image_path = os.path.join(input_dir, image_filename)
    original_image = imageio.imread(image_path)
    base_name = os.path.splitext(image_filename)[0]
```

```
    contrast_E = 10
    midpoint_m = 0.5
    corrected_image = tone_correction(original_image, contrast_E, midpoint_m)
    fig = plt.figure(figsize=(18, 7))
    gs = fig.add_gridspec(1, 3, width_ratios=[4, 4, 1.5], wspace=0.05)
    fig.suptitle(f'Tone Correction (Fig 6.33) - {image_filename}', fontsize=16)

    ax0 = fig.add_subplot(gs[0])
    ax0.imshow(original_image)
    ax0.set_title('a) Flat Image')
    ax0.axis('off')

    ax1 = fig.add_subplot(gs[1])
    ax1.imshow(corrected_image)
    ax1.set_title('b) Corrected Image')
    ax1.axis('off')

    ax2 = fig.add_subplot(gs[2])
    plot_tone_curve(ax2, contrast_E, midpoint_m)
    ax2.set_title('c) Transformation')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])

    combined_output_path = os.path.join(output_dir,
f"{base_name}_tone_correction_results.png")
    plt.savefig(combined_output_path, bbox_inches='tight')
    plt.close()
```

Processed Images:



Homework5: Implement Figure 6.35 by programming with Python

Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os
import warnings
import matplotlib.gridspec as gridspec


def rgb_to_hsi(rgb_image):
    img_float = rgb_image.astype(float) / 255.0
    R, G, B = img_float[:, :, 0], img_float[:, :, 1], img_float[:, :, 2]
    H = np.zeros_like(R)
    S = np.zeros_like(R)
    I = (R + G + B) / 3.0
    min_rgb = np.minimum(np.minimum(R, G), B)
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", RuntimeWarning)
        S = 1 - (3.0 / (R + G + B + 1e-6)) * min_rgb
    num = 0.5 * ((R - G) + (R - B))
    den = np.sqrt((R - G) ** 2 + (R - B) * (G - B))
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", RuntimeWarning)
        theta = np.arccos(np.clip(num / (den + 1e-6), -1, 1))
    theta = np.degrees(theta)
    H[B <= G] = theta[B <= G]
    H[B > G] = 360 - theta[B > G]
    H[S == 0] = 0
    hsi_image = np.stack([H, S, I], axis=2)
    return hsi_image


def hsi_to_rgb(hsi_image):
    H, S, I = hsi_image[:, :, 0], hsi_image[:, :, 1], hsi_image[:, :, 2]
    R, G, B = np.zeros_like(H), np.zeros_like(H), np.zeros_like(H)
    idx = (H >= 0) & (H < 120)
    B[idx] = I[idx] * (1 - S[idx])
    R[idx] = I[idx] * (1 + (S[idx] * np.cos(np.radians(H[idx]))) /
np.cos(np.radians(60 - H[idx])))
    G[idx] = 3 * I[idx] - (R[idx] + B[idx])
    idx = (H >= 120) & (H < 240)
    H_prime = H[idx] - 120
```

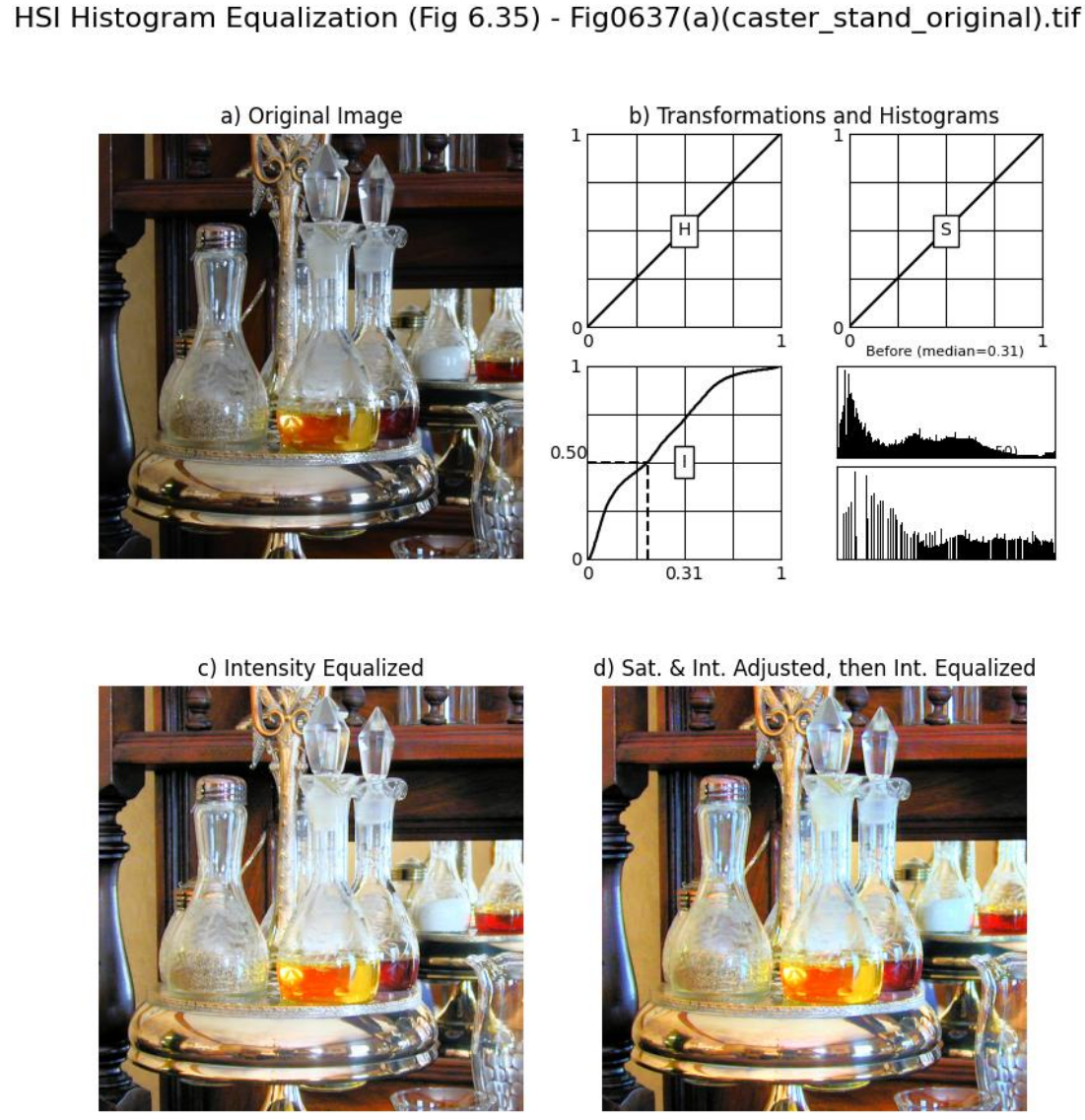
```
    R[idx] = I[idx] * (1 - S[idx])
    G[idx] = I[idx] * (1 + (S[idx] * np.cos(np.radians(H_prime))) /
np.cos(np.radians(60 - H_prime)))
    B[idx] = 3 * I[idx] - (R[idx] + G[idx])
    idx = (H >= 240) & (H < 360)
    H_prime = H[idx] - 240
    G[idx] = I[idx] * (1 - S[idx])
    B[idx] = I[idx] * (1 + (S[idx] * np.cos(np.radians(H_prime))) /
np.cos(np.radians(60 - H_prime)))
    R[idx] = 3 * I[idx] - (G[idx] + B[idx])
    rgb_image = np.stack([R, G, B], axis=2)
    rgb_image = np.clip(rgb_image * 255, 0, 255).astype(np.uint8)
    return rgb_image


def histogram_equalization(intensity_channel):
    I_uint8 = np.clip(intensity_channel * 255, 0, 255).astype(np.uint8)
    hist, _ = np.histogram(I_uint8.flatten(), 256, [0, 256])
    cdf = hist.cumsum()
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
    cdf = np.ma.filled(cdf_m, 0).astype('uint8')
    equalized_I_uint8 = cdf[I_uint8]
    equalized_I_float = equalized_I_uint8.astype(float) / 255.0
    equalized_hist, _ = np.histogram(equalized_I_uint8.flatten(), 256, [0, 256])
    transform_func = cdf / 255.0
    return equalized_I_float, hist, equalized_hist, transform_func


def plot_panel_b(fig, spec, hist_orig, hist_eq, transform_func, median_orig,
median_eq):
    gs_nested = gridspec.GridSpecFromSubplotSpec(2, 2, subplot_spec=spec,
wspace=0.2, hspace=0.2)

    def style_ax(ax):
        ax.set_aspect('equal', adjustable='box')
        ax.set_xlim(0, 1)
        ax.set_ylim(0, 1)
        ticks = np.linspace(0, 1, 5)
        ax.set_xticks(ticks)
        ax.set_yticks(ticks)
        ax.set_xticklabels(['0', '', '', '', '1'])
        ax.set_yticklabels(['0', '', '', '', '1'])
        ax.grid(True, color='black', linewidth=0.75)
```

```
        ax.tick_params(length=0)

    ax_h = fig.add_subplot(gs_nested[0, 0])
    ax_h.plot([0, 1], [0, 1], 'k')
    ax_h.text(0.5, 0.5, 'H', ha='center', va='center', bbox={'facecolor': 'white',
'edgecolor': 'black'})
    style_ax(ax_h)

    ax_s = fig.add_subplot(gs_nested[0, 1])
    ax_s.plot([0, 1], [0, 1], 'k')
    ax_s.text(0.5, 0.5, 'S', ha='center', va='center', bbox={'facecolor': 'white',
'edgecolor': 'black'})
    style_ax(ax_s)

    ax_i = fig.add_subplot(gs_nested[1, 0])
    r = np.linspace(0, 1, 256)
    ax_i.plot(r, transform_func, 'k')
    ax_i.plot([median_orig, median_orig], [0, median_eq], 'k--')
    ax_i.plot([0, median_orig], [median_eq, median_eq], 'k--')
    ax_i.text(0.5, 0.5, 'I', ha='center', va='center', bbox={'facecolor': 'white',
'edgecolor': 'black'})
    style_ax(ax_i)
    # Specific labels for I plot
    ax_i.set_xticklabels(['0', '', '', '', '1'])
    ax_i.set_yticklabels(['0', '', '', '', '1'])
    ax_i.set_xlabel(f'{median_orig:.2f}', labelpad=-10)
    ax_i.set_ylabel(f'{median_eq:.2f}', labelpad=-10, rotation=0, ha='right')

    gs_hist = gridspec.GridSpecFromSubplotSpec(2, 1, subplot_spec=gs_nested[1, 1],
hspace=0.1)
    ax_hist1 = fig.add_subplot(gs_hist[0])
    ax_hist2 = fig.add_subplot(gs_hist[1])
    ax_hist1.bar(range(256), hist_orig, color='black', width=1.0)
    ax_hist1.set_title(f'Before (median={median_orig:.2f})', fontsize=8)
    ax_hist2.bar(range(256), hist_eq, color='black', width=1.0)
    ax_hist2.set_title(f'After (median={median_eq:.2f})', fontsize=8)
    for ax in [ax_hist1, ax_hist2]:
        ax.set_yticks([])
        ax.set_xticks([])
        ax.set_xlim(0, 255)


if __name__ == "__main__":
    input_dir = "original_image"
```

```
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)

    image_filename = "Fig0637(a)(caster_stand_original).tif"
    image_path = os.path.join(input_dir, image_filename)
    original_image = imageio.imread(image_path)
    base_name = os.path.splitext(image_filename)[0]

    hsi_orig = rgb_to_hsi(original_image)
    H_orig, S_orig, I_orig = hsi_orig[:, :, 0], hsi_orig[:, :, 1], hsi_orig[:, :,
2]

    I_eq, hist_orig, hist_eq, transform_func = histogram_equalization(I_orig)
    hsi_c = np.stack([H_orig, S_orig, I_eq], axis=2)
    image_c = hsi_to_rgb(hsi_c)

    S_d_pre = S_orig ** 0.75
    I_d_pre = I_orig ** 0.75
    I_d_eq, _, _, _ = histogram_equalization(I_d_pre)
    hsi_d = np.stack([H_orig, S_d_pre, I_d_eq], axis=2)
    image_d = hsi_to_rgb(hsi_d)

    fig = plt.figure(figsize=(10, 10))
    gs_main = gridspec.GridSpec(2, 2, figure=fig, wspace=0.05, hspace=0.3)
    fig.suptitle(f'HSI Histogram Equalization (Fig 6.35) - {image_filename}',
fontsize=16)

    ax_a = fig.add_subplot(gs_main[0, 0])
    ax_a.imshow(original_image)
    ax_a.set_title('a) Original Image')
    ax_a.axis('off')

    median_orig = np.median(I_orig)
    median_eq = np.median(I_eq)
    plot_panel_b(fig, gs_main[0, 1], hist_orig, hist_eq, transform_func,
median_orig, median_eq)
    ax_b_title = fig.add_subplot(gs_main[0, 1])
    ax_b_title.set_title('b) Transformations and Histograms')
    ax_b_title.axis('off')

    ax_c = fig.add_subplot(gs_main[1, 0])
    ax_c.imshow(image_c)
    ax_c.set_title('c) Intensity Equalized')
    ax_c.axis('off')
```

```
    ax_d = fig.add_subplot(gs_main[1, 1])
    ax_d.imshow(image_d)
    ax_d.set_title('d) Sat. & Int. Adjusted, then Int. Equalized')
    ax_d.axis('off')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])

    combined_output_path = os.path.join(output_dir,
f"{base_name}_hsi_equalization_results.png")
    plt.savefig(combined_output_path, bbox_inches='tight')
    plt.close()
```

Processed Images:



Homework6: Implement Figure 6.44 by programming with Python

Python Code:

```
import numpy as np
import imageio.v2 as imageio
import matplotlib.pyplot as plt
import os
from scipy import ndimage


def vector_gradient(image):
    img_float = image.astype(np.float64)
    g_xx = np.zeros_like(img_float[:, :, 0])
    g_yy = np.zeros_like(img_float[:, :, 0])
    for i in range(3):
        channel = img_float[:, :, i]
        g_x = ndimage.sobel(channel, axis=1)
        g_y = ndimage.sobel(channel, axis=0)
        g_xx += g_x ** 2
        g_yy += g_y ** 2
    M = np.sqrt(g_xx + g_yy)
    return M


def sum_of_gradients(image):
    img_float = image.astype(np.float64)
    M_sum = np.zeros_like(img_float[:, :, 0])
    for i in range(3):
        channel = img_float[:, :, i]
        g_x = ndimage.sobel(channel, axis=1)
        g_y = ndimage.sobel(channel, axis=0)
        M_sum += np.sqrt(g_x ** 2 + g_y ** 2)
    return M_sum


def scale_to_uint8(image_float):
    max_val = np.max(image_float)
    if max_val > 0:
        scaled_image = (image_float / max_val) * 255
    else:
        scaled_image = image_float
    return scaled_image.astype(np.uint8)
```

```python
if __name__ == "__main__":
    input_dir = "original_image"
    output_dir = "output_image"
    os.makedirs(output_dir, exist_ok=True)

    image_filename = "Fig0646(a)(lenna_original_RGB).tif"
    image_path = os.path.join(input_dir, image_filename)
    original_image = imageio.imread(image_path)
    base_name = os.path.splitext(image_filename)[0]

    grad_vector = vector_gradient(original_image)
    grad_sum = sum_of_gradients(original_image)
    grad_diff = np.abs(grad_vector - grad_sum)

    display_vector = scale_to_uint8(grad_vector)
    display_sum = scale_to_uint8(grad_sum)
    display_diff = scale_to_uint8(grad_diff)

    fig, axes = plt.subplots(2, 2, figsize=(10, 10))
    fig.suptitle(f'RGB Edge Detection (Fig 6.44) - {image_filename}', fontsize=16)

    axes[0, 0].imshow(original_image)
    axes[0, 0].set_title('a) Original RGB Image')

    axes[0, 1].imshow(display_vector, cmap='gray')
    axes[0, 1].set_title('b) Vector Gradient')

    axes[1, 0].imshow(display_sum, cmap='gray')
    axes[1, 0].set_title('c) Sum of Individual Gradients')

    axes[1, 1].imshow(display_diff, cmap='gray')
    axes[1, 1].set_title('d) Difference (b) - (c)')

    for ax in axes.flat:
        ax.axis('off')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])

    combined_output_path = os.path.join(output_dir,
f"{base_name}_rgb_edge_detection_results.png")
    plt.savefig(combined_output_path, bbox_inches='tight')
    plt.close()
```

Processed Images:

RGB Edge Detection (Fig 6.44) - Fig0646(a)(lenna_original_RGB).tif

a) Original RGB Image

b) Vector Gradient

c) Sum of Individual Gradients

d) Difference (b) - (c)