Sistemas de Cores
RGB, HIS, CMYK e RAL

Sistemas de Cores

-Métodos Padronizados para Reproduzir Cores

RGB

-Aditivo

-Utilizado principalmente em Eletrônicos

-Cubo de lado 256

RGB

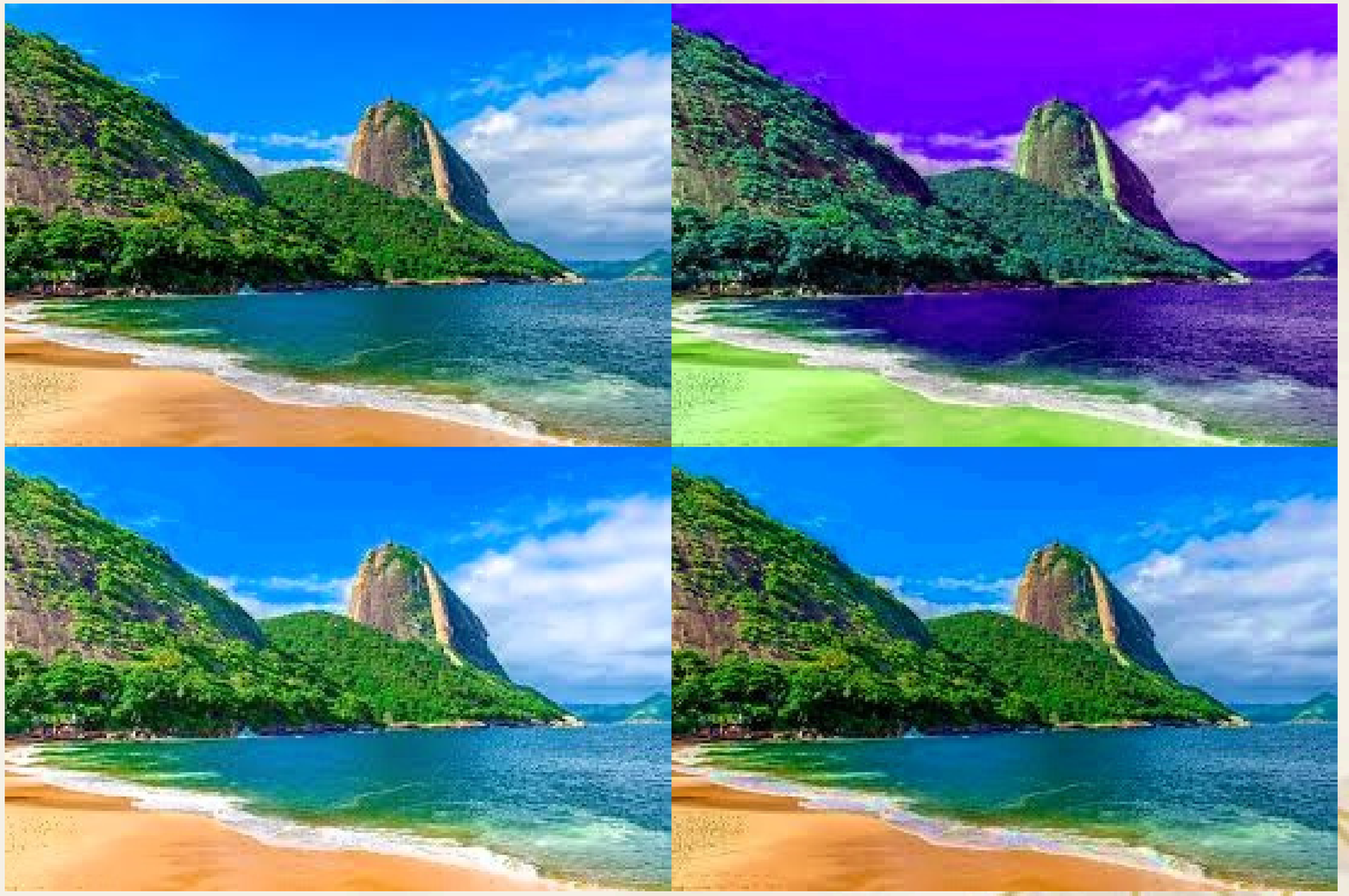
```python
# Lê a imagem
img = cv2.imread(imagepath)

# Separa os canais
blue, green, red = cv2.split(img)

# Cria imagens para cada canal, preenchendo os outros canais com zero
zeros = np.zeros(img.shape[:2], dtype="uint8")
blue_channel = cv2.merge([blue, zeros, zeros])
green_channel = cv2.merge([zeros, green, zeros])
red_channel = cv2.merge([zeros, zeros, red])

cv2.imwrite('imagem_original.jpg', img)
cv2.imwrite('canal_vermelho.jpg', red_channel)
cv2.imwrite('canal_verde.jpg', green_channel)
cv2.imwrite('canal_azul.jpg', blue_channel)
```

HIS

-Matiz(D), Intensidade(B) e Saturação(BD)

-Utilizado no Design Gráfico

-Necessidade de Conversão



HIS

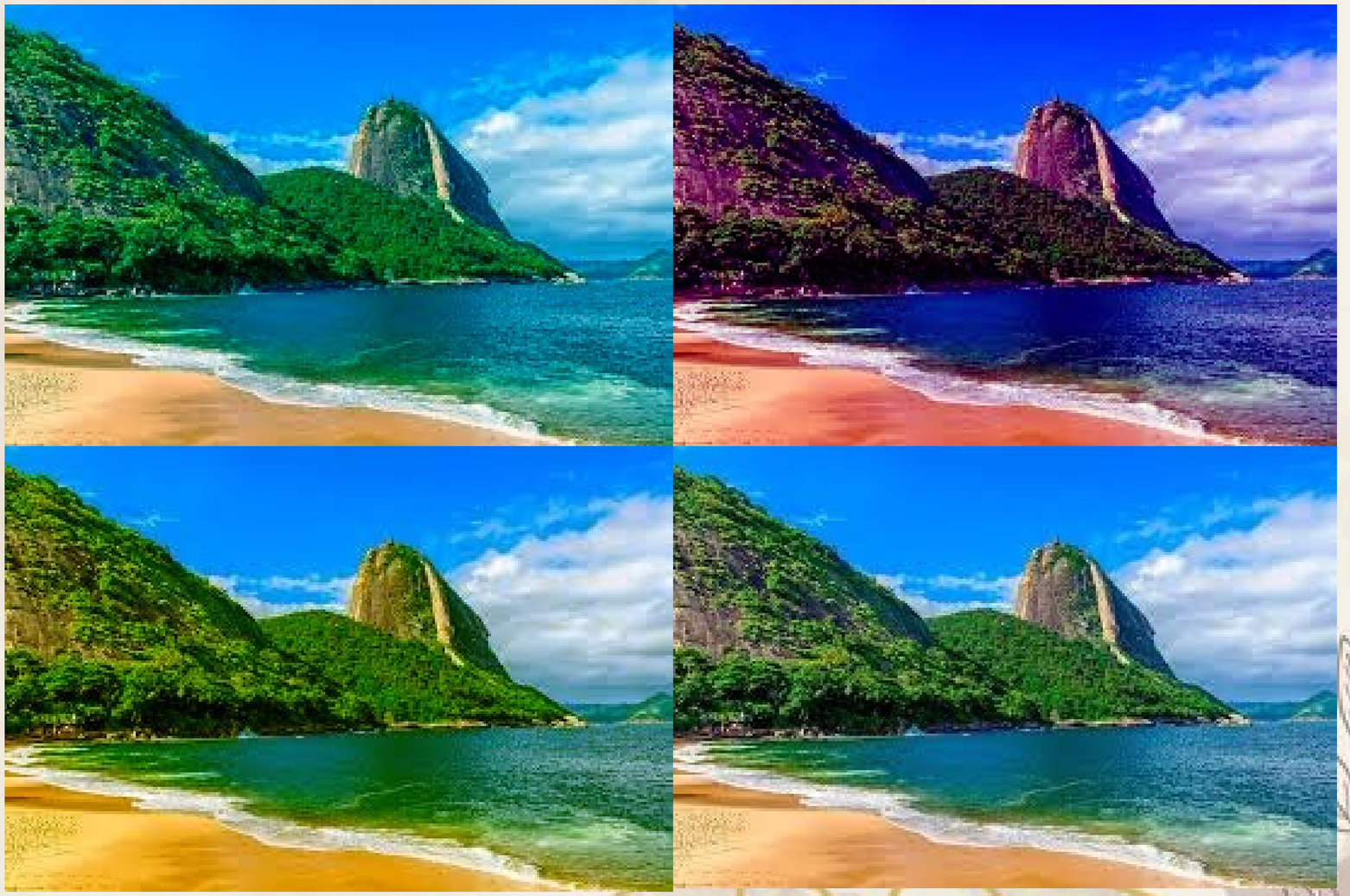
```
    # Modifica o canal H (Matiz)
    # Adiciona 30 graus ao matiz (lembrando que H vai de 0-179 no OpenCV)
h_modified[:,:,0] = (h_modified[:,:,0] + 30) % 180

    # Modifica o canal S (Saturação)
    # Aumenta a saturação em 30 (lembrando que vai de 0 a 255 no OpenCV)
s_modified[:,:,1] = cv2.add(s_modified[:,:,1], 30)

    # Modifica o canal I (Intensidade/Valor)
    # Aumenta o brilho em 30 (lembrando que vai de 0 a 255 no OpenCV)
i_modified[:,:,2] = cv2.add(i_modified[:,:,2], 30)

# Converte de volta para BGR para exibição
h_result = cv2.cvtColor(h_modified, cv2.COLOR_HSV2BGR)
s_result = cv2.cvtColor(s_modified, cv2.COLOR_HSV2BGR)
i_result = cv2.cvtColor(i_modified, cv2.COLOR_HSV2BGR)
```

CMYK

-Ciano(CE), Magenta(CD), Amarelo(BE) e Preto(BD)

-Utilizado na Impressão

-Menor Gama de Cores que o RGB



CMYK

```python
# Função auxiliar para aumentar a intensidade de um canal
def aumentar_intensidade(canal, fator=1.5):
    canal_array = np.array(canal)
    canal_modificado = np.clip(canal_array * fator, 0, 255).astype(np.uint8)
    return Image.fromarray(canal_modificado)

# Cria 4 versões diferentes

# 1. Aumenta Ciano
c_modificado = aumentar_intensidade(c)
imagem_c = Image.merge('CMYK', (c_modificado, m, y, k))

# 2. Aumenta Magenta
m_modificado = aumentar_intensidade(m)
imagem_m = Image.merge('CMYK', (c, m_modificado, y, k))

# 3. Aumenta Amarelo
y_modificado = aumentar_intensidade(y)
imagem_y = Image.merge('CMYK', (c, m, y_modificado, k))

# 4. Aumenta Preto
k_modificado = aumentar_intensidade(k)
imagem_k = Image.merge('CMYK', (c, m, y, k_modificado))
```

RAL

-Sistema Industrial Padronizado

-Sistema clássico possui 216 Cores

-Garante Padrão ->Comunicação cliente fornecedor

-Telas digitais não conseguem reproduzir fielmente

RAL

```python
# Dicionário com algumas cores RAL comuns (código RAL: (R,G,B))
RAL_COLORS: Dict[str, Tuple[int, int, int]] = {
    'RAL 9010': (255, 255, 255),  # Branco Puro
    'RAL 9005': (0, 0, 0),        # Preto Brilhante
    'RAL 3020': (204, 6, 5),      # Vermelho Tráfego
    'RAL 5015': (0, 127, 167),    # Azul Céu
    'RAL 6029': (0, 111, 41),     # Verde Menta
    'RAL 1021': (243, 218, 11),   # Amarelo Colza
    'RAL 2004': (244, 70, 17),    # Laranja Puro
    'RAL 4006': (160, 52, 114),   # Púrpura Tráfego
    'RAL 8017': (69, 50, 46),     # Marrom Chocolate
    'RAL 7035': (215, 215, 215),  # Cinza Claro
}

def find_closest_ral_color(pixel: np.ndarray) -> str:
    """
    Encontra a cor RAL mais próxima para um dado pixel RGB
    """
    min_distance = float('inf')
    closest_ral = None

    # Converte o pixel para o formato correto se necessário
    if len(pixel.shape) > 1:
        pixel = pixel.flatten()

    for ral_code, ral_rgb in RAL_COLORS.items():
        # Calcula a distância euclidiana entre as cores
        distance = np.sqrt(sum((pixel - np.array(ral_rgb)) ** 2))

        if distance < min_distance:
            min_distance = distance
            closest_ral = ral_code

    return closest_ral
```

Obrigado!