Дата: 10.06.2023
ФИО: Козлов Евгений Юрьевич
Группа: 224-322

ПРАКТИЧЕСКАЯ РАБОТА № 2
Применение методов градационной коррекции по переходным кривым

1. Цель работы

Познакомится с пространственными методами коррекции на примере градационной коррекции по переходным кривым.

2. Содержание работы

Этапы выполнения:

1. Подобрать 2 изображения для коррекции
2. Перевести изображения в черно-белое
3. Преобразовать изображения в негатив
4. Провести логарифмическое преобразование
5. Провести степенное преобразование с $\gamma>1$, $\gamma<1$
6. Провести кусочно-линейное преобразование
7. Провести вырезание уровней в изображении (для одного изображения)

Содержание отчета:

1. Название цель работы
2. Используемый язык программирования
3. Параметры исходных изображений (назвать изображения 01 и 02)
    a. глубина цвета - k, bpp
    b. размер - m x n, pix
4. Изображение, преобразованное в негатив (01_neg, 02_neg)
5. Вид функции преобразования
6. Параметры логарифмических преобразований
7. Изображение после логарифмического преобразования (01_log_x, 01_log_y, 02_log_x, 02_log_y)
8. Вид функций преобразования
9. Параметры степенных преобразований
10. Изображение после степенного преобразования (01_deg_x, 01_deg_y, 02_deg_x, 02_deg_y)
11. Вид функций преобразования
12. Параметры кусочно-линейного преобразования
13. Изображения после кусочно-линейного преобразования (01_sl, 02_sl)
14. Вид функций преобразования
15. Номера вырезаемых уровней в выбранном изображении
16. Изображения вырезанных уровней
17. Приложить код программы

Исходные изображения и все изображения после коррекций выложить на облачное хранилище и приложить ссылку.

3. Исходные данные и программное обеспечение

Используемая среда программирования: Visual Studio Code
Используемый язык программирования: Python 3.11.1 64-bit
Используемые библиотеки:
cv2 – https://opencvguide.readthedocs.io/en/latest/opencvpython/basics.html
NumPy - работа с массивами

4. Выполнение работы

1. Подобрать 2 изображения для коррекции

Исходные изображения:


Изображения, преобразованные на основе яркости (LAB):


2. Перевести изображения в черно-белое:


В cv2 параметр L (яркость в LAB) принимает значения от 0 до 255, поэтому если L > (255 / 2), то ставится L = 255, иначе 0.

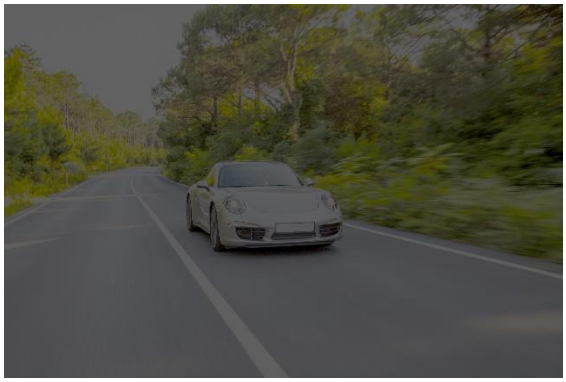
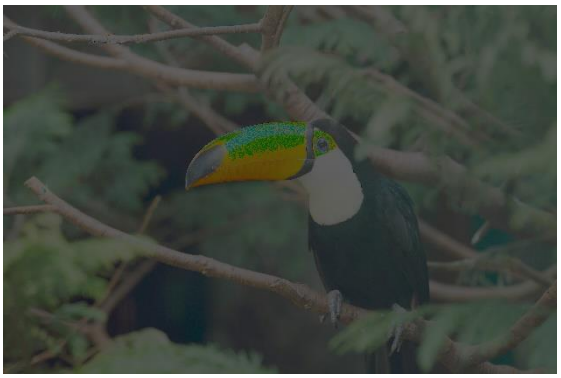
3. Преобразовать изображения в негатив
   Изображения в негативе на основе яркости (LAB):


Изображения в негативе на основе инвертированных пикселей пространства RGB:


Чтобы получить негативное изображение следует применить формулу s=255-r, где s –
результат (выходные пиксели), r – исходное изображение (входные пиксели), таким образом, если пиксель был светлый, то он станет тёмным и наоборот.

4. Провести логарифмическое преобразование
   (c = 255):


(c = 100):

Формула функции преобразования:
s = c * (np.log(1 + r)/(np.log(1 + np.max(r)))),
где r – входное изображение (входные пиксели), c – константа масштабирования.
np.log берёт натуральный логарифм,
np.max возвращает максимальное значениев массиве r.
Формула берет логарифм от каждого элемента массива r, затем нормализует значения таким образом, чтобы максимальноезначение было равно 1. Затем она умножает результат на c, чтобы получить значения в диапазоне от 0 до c.

5. Провести степенное преобразование с $\gamma>1$, $\gamma<1$
   γ (гамма) = 1.4:


γ (гамма) = 0.7:

Здесь вычисляется таблица соответствия пикселей по формуле выше, где c – константа, r – входной пиксель,y – константа гаммы, s – выходной пиксель.
После с помощью функции LUT от cv2 происходит замена пикселей на те, которые указаны в таблице соответствия.
Параметры: r – входное изображение (входные пиксели), y – константа гаммы, если она больше 1, то пиксели будут более темными, если меньше, то более светлыми.

6. Провести кусочно-линейное преобразование
   Изображения после кусочно-линейного преобразования на основе пространства RGB с параметрами (20, 0, 228, 255):

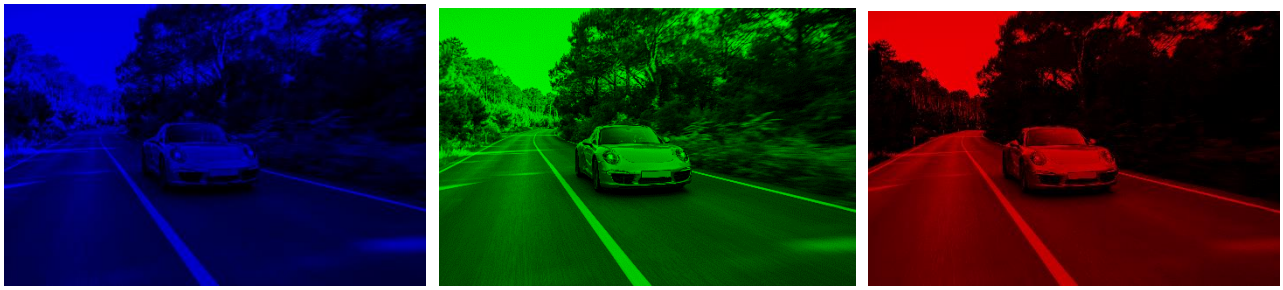
Вид функции преобразования выглядит так:

```
if x < r1:
        return (s1 / r1) * x
elif r1 <= x < r2:
        return ((s2 - s1) / (r2 - r1)) * (x - r1) + s1
else:
        return ((255 - s2) / (255 - r2)) * (x - r2) + s2
```

Параметры: r – входное изображение (пиксели), r1 – параметр первой

границы, (до 20), s1 – результирующий параметр первой границы (обычно берется 0), означает, что результирующие пиксели станут0, r2 – параметр второй границы, s2 – результирующий параметр второй границы.

7. Провести вырезание уровней в изображении (для одного изображения)
   B (blue), G (green) и R (red) каналы сответственно:



Номера вырезаемых каналов: 0, 1, 2.

Изображения первой картинки после вырезания 3-х уровней в отдельные файлы на основе RGB:

Номера вырезаемых каналов: 4, 5, 6;

Вывод:

В ходе работы были применены методы градационной коррекции по переходным кривым. При преобразовании изображений в ч/б, на первом изображении черным участком становится асфальт,а на втором изображении природа и задний фон, при инвертировании все пиксели меняются на противоположные, то есть белые становятся черными, а черные белыми. Логарифмические преобразования делают изображение более светлым и смягчает контраст. Степенное преобразование с гаммой больше 1 делают изображение более темным (увеличивает насыщенность), а с менее 1 более светлым (уменьшает насыщенность). Кусочно-линейное преобразование позволяет делать одни участки более темными, а другие более светлыми, регулируя это с помощью кривой с входными узловыми точками. Вырезание цветовых уровней представляет изображения по отдельным цветовым каналам, а вырезание битовых уровней помогает определить вклад тех или иных битов в целостность изображения (старшие биты отвечают за основную

визуальную часть, младшие биты – за детали).

Все изображения хранятся на гугл-диске: https://drive.google.com/drive/folders/1xSvyRVj76hBNEuRcTdeEyYz_6bgrtEaS?usp=sharing

Код программы:

```
# теория
# https://www.geeksforgeeks.org/python-intensity-transformation-
operations-on-images/

import cv2
import numpy as np

# 1. Исходные изображения (+и все остальные файлы размещены на гугл-диске)
INIT_IMG1 = cv2.imread('img/init/1.jpg')
INIT_IMG2 = cv2.imread('img/init/2.jpg')

LAB_INIT_IMG1 = cv2.cvtColor(INIT_IMG1.copy(), cv2.COLOR_BGR2LAB)
LAB_INIT_IMG2 = cv2.cvtColor(INIT_IMG2.copy(), cv2.COLOR_BGR2LAB)
cv2.imwrite('img/dist/1/LAB_INIT_IMG1.png', LAB_INIT_IMG1)
cv2.imwrite('img/dist/1/LAB_INIT_IMG2.png', LAB_INIT_IMG2)

# 2. Преобразуйте изображение в черно-белое
def monochrome_from_lab(input_lab_img):
    width, height, depth = input_lab_img.shape
    for i in range(width):
        for j in range(height):
            l, a, b = input_lab_img[i, j]
            gray = 255 if (l > 255 / 2) else 0
            input_lab_img[i, j] = (gray, gray, gray)
    return input_lab_img

MONO_FROM_LAB_1 = monochrome_from_lab(LAB_INIT_IMG1.copy())
MONO_FROM_LAB_2 = monochrome_from_lab(LAB_INIT_IMG2.copy())
cv2.imwrite('img/dist/2/MONO_FROM_LAB_1.png',
MONO_FROM_LAB_1)
cv2.imwrite('img/dist/2/MONO_FROM_LAB_2.png',
MONO_FROM_LAB_2)

# 3. Преобразуйте изображение в негатив
NEGATIVE_FROM_MONO_1 = 255 - MONO_FROM_LAB_1.copy()
NEGATIVE_FROM_MONO_2 = 255 - MONO_FROM_LAB_2.copy()
cv2.imwrite('img/dist/3/NEGATIVE_FROM_MONO_1.png',
NEGATIVE_FROM_MONO_1)
cv2.imwrite('img/dist/3/NEGATIVE_FROM_MONO_2.png',
NEGATIVE_FROM_MONO_2)

# NEGATIVE RGB
NEGATIVE_FROM_RGB_1 = 255 - INIT_IMG1.copy()
NEGATIVE_FROM_RGB_2 = 255 - INIT_IMG2.copy()
```

```
cv2.imwrite('img/dist/3/NEGATIVE_FROM_RGB_1.png', NEGATIVE_FROM_RGB_1)
cv2.imwrite('img/dist/3/NEGATIVE_FROM_RGB_2.png', NEGATIVE_FROM_RGB_2)

# 4. Логарифмическое преобразование изображения
# (используется для улучшения контраста)
# s = c * log(1+r), где
# R - значение пикселя входного изображения
# c - константа
# S - значение пикселя выходного изображения

# RGB
def log_from_rgb(r, c):
    # Формула лог. преобразования
    max = np.max(r) # Значение 255
    log_max = np.log(1 + max) # Значение 5.54
    s = c * (np.log(1 + r)/log_max)
    # Перевод в формат для вывода
    s = np.array(s, dtype=np.uint8)
    return s

LOG_FROM_RGB_C255_1 = log_from_rgb(INIT_IMG1.copy(), 255)
LOG_FROM_RGB_C255_2 = log_from_rgb(INIT_IMG2.copy(), 255)

LOG_FROM_RGB_C100_1 = log_from_rgb(INIT_IMG1.copy(), 100)
LOG_FROM_RGB_C100_2 = log_from_rgb(INIT_IMG2.copy(), 100)

cv2.imwrite('img/dist/4/LOG_FROM_RGB_C255_1.png', LOG_FROM_RGB_C255_1)
cv2.imwrite('img/dist/4/LOG_FROM_RGB_C255_2.png', LOG_FROM_RGB_C255_2)

cv2.imwrite('img/dist/4/LOG_FROM_RGB_C100_1.png', LOG_FROM_RGB_C100_1)
cv2.imwrite('img/dist/4/LOG_FROM_RGB_C100_2.png', LOG_FROM_RGB_C100_2)

# 5. Провести степенное преобразование с γ>1, γ<1
# S = c * r ** γ
def deg_rgb(r, y):
    table = np.array([((i / 255.0) ** y) * 255 for i in
np.arange(0,256)]).astype("uint8")
    corrected_gamma = cv2.LUT(r, table)
    return corrected_gamma

DEG_MORE_SAT_1 = deg_rgb(INIT_IMG1.copy(), 1.4)
DEG_MORE_SAT_2 = deg_rgb(INIT_IMG2.copy(), 1.4)

DEG_LESS_SAT_1 = deg_rgb(INIT_IMG1.copy(), 0.7)
DEG_LESS_SAT_2 = deg_rgb(INIT_IMG2.copy(), 0.7)

cv2.imwrite('img/dist/5/DEG_MORE_SAT_1.png', DEG_MORE_SAT_1)
cv2.imwrite('img/dist/5/DEG_MORE_SAT_2.png', DEG_MORE_SAT_2)

cv2.imwrite('img/dist/5/DEG_LESS_SAT_1.png', DEG_LESS_SAT_1)
cv2.imwrite('img/dist/5/DEG_LESS_SAT_2.png', DEG_LESS_SAT_2)

# 6. Провести кусочно-линейное преобразование
```

```python
# Значения (r1, s1) (r2, s2) обеспечивают различную степень растяжения уровней
# яркости на результирующем изображении (r - яркость на входе, s - яркость на выходе), меняя тем самым его контраст.
# Зачастую наиболее эффективным выбором параметров является следующий: r1=rmin, r2=rmax, s1=0 и s2=L−1,
# где rmin и rmax — означают минимальную и максимальную яркости исходного изображения.

def plt(img, r1, s1, r2, s2):
    def piecewise_linear(x):
        if x < r1:
            return (s1 / r1) * x
        elif r1 <= x < r2:
            return ((s2 - s1) / (r2 - r1)) * (x - r1) + s1
        else:
            return ((255 - s2) / (255 - r2)) * (x - r2) + s2
    table = np.array([piecewise_linear(i) for i in range(256)]).astype('uint8')
    return cv2.LUT(img, table)

PLT_FROM_RGB_1 = plt(INIT_IMG1.copy(), 20, 0, 228, 255)
PLT_FROM_RGB_2 = plt(INIT_IMG2.copy(), 20, 0, 228, 255)
cv2.imwrite('img/dist/6/PLT_FROM_RGB_1.png', PLT_FROM_RGB_1)
cv2.imwrite('img/dist/6/PLT_FROM_RGB_2.png', PLT_FROM_RGB_2)

# 7. Провести вырезание уровней в изображении

# Пространственная область изображения, это массив пикселей, каждый
# пиксель обладает определенным значением светлоты L в интервале [0: L-1].
# Число уровней L зависит от числа уровней квантования, число уровней рассчитывается как 2 ** n,
# где n - глубина цвета (кол-во битов на пиксель - bpp)

def cut_channel_from_rgb(input_img, number_channel):
    width, height, channels = input_img.shape
    for i in range(width):
        for j in range(height):
            color_rgb = input_img[i, j]
            for c in range(channels):
                # обнуляем каналы, которые не равны вырезаемому
                if (number_channel != c):
                    color_rgb[c] = 0
            input_img[i, j] = color_rgb
    return input_img

SRC_RGB_IMG = cv2.cvtColor(INIT_IMG1, cv2.COLOR_BGR2RGB)
RGB_BLUE = cut_channel_from_rgb(SRC_RGB_IMG.copy(), 0)
RGB_GREEN = cut_channel_from_rgb(SRC_RGB_IMG.copy(), 1)
RGB_RED = cut_channel_from_rgb(SRC_RGB_IMG.copy(), 2)

cv2.imwrite('img/dist/7/RGB_BLUE.png', RGB_BLUE)
cv2.imwrite('img/dist/7/RGB_GREEN.png', RGB_GREEN)
cv2.imwrite('img/dist/7/RGB_RED.png', RGB_RED)
```

```python
# Вместо выделения диапазонов яркостей, может оказаться полезным выделение информации о вкладе тех или иных битов
# в общее изображение. Пусть каждый пиксель изображения представлен 8 битами. В этом случае все изображение можно
# представить себе в виде 8-битовых плоскостей, ранжированных от плоскости 0 с наименее значащими
# битами до плоскости 7 с наиболее значащими битами.   Старшие биты с 7 по 4 содержат основную часть
# визуально значимых данных, октальные битовые плоскости с 0 по 3 дают вклад в более тонкие детали изображения.
# Разделение цифрового изображения на битовые плоскости полезно для анализа относительной информативности,
# которую несет каждый бит изображения, что позволяет оценить необходимое число битов,
# требуемое для квантования каждого пикселя.

def cut_bit_from_rgb(input_img, number_level):
    width, height, channels = input_img.shape
    min_bit = 2 ** (number_level - 1)
    select_bit = 2 ** number_level
    max_bit = 2 ** (number_level + 1)
    for i in range(width):
        for j in range(height):
            r, g, b = input_img[i, j]
            avg_rgb = (r + g + b) / 3
            if (avg_rgb > min_bit and avg_rgb < max_bit):
                input_img[i, j] = r, g, b
            else:
                input_img[i, j] = 255, 255, 255
    return input_img

BIT_4_RGB_1 = cut_bit_from_rgb(SRC_RGB_IMG.copy(), 4)
BIT_5_RGB_1 = cut_bit_from_rgb(SRC_RGB_IMG.copy(), 5)
BIT_6_RGB_1 = cut_bit_from_rgb(SRC_RGB_IMG.copy(), 6)
cv2.imwrite('img/dist/7/BIT_4_RGB_1.png', BIT_4_RGB_1)
cv2.imwrite('img/dist/7/BIT_5_RGB_1.png', BIT_5_RGB_1)
cv2.imwrite('img/dist/7/BIT_6_RGB_1.png', BIT_6_RGB_1)
```