# Дата: 10.06.2023

**ФИО: Козлов Евгений Юрьевич Группа: 224-322**

# ПРАКТИЧЕСКАЯ РАБОТА № 2

**Применение методов градационной коррекции по переходным кривым**

# Цель работы

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

# 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 γ>1, γ<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)
   1. глубина цвета - k, bpp
   2. 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.

1. **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

# Выполнение работы

1. Подобрать 2 изображения для коррекции

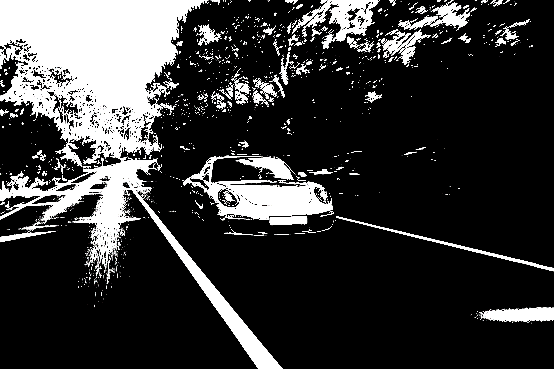
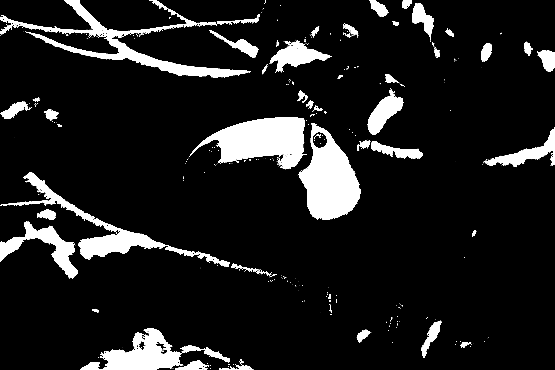
Исходные изображения:



Изображения, преобразованные на основе яркости (LAB):

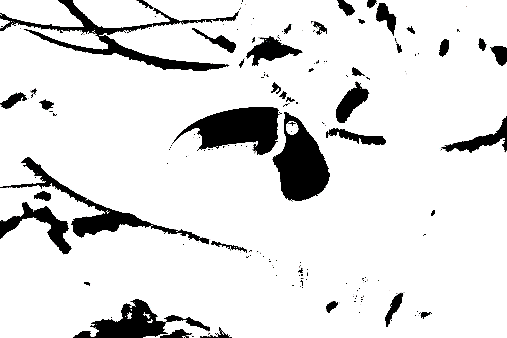
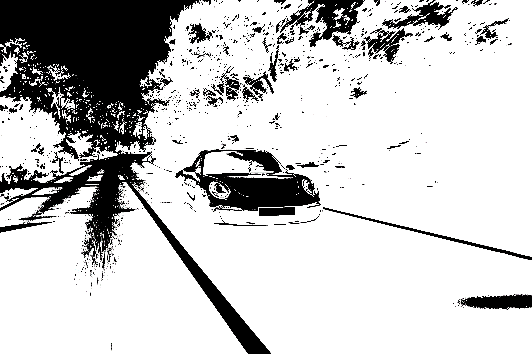


1. Перевести изображения в черно-белое:



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

1. 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.

1. Провести логарифмическое преобразование

(с = 255):



(с = 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 значение в массиве r.

Формула берет логарифм от каждого элемента массива r, затем нормализует значения таким образом, чтобы максимальное значение было равно 1. Затем она умножает результат на c, чтобы получить значения в диапазоне от 0 до c.

1. Провести степенное преобразование с γ>1, γ<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.

1. 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 станут 0, r2 – параметр второй границы, s2 – результирующий параметр второй границы.

1. 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

# 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)

# 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)