**Лабораторная работа №1**

**Содержание**

1. Цветовые пространства. Преобразование между цветовыми пространствами. Поиск объекта по цвету. Цветовая коррекция.……2
2. Геометрические преобразования изображений…………………...…12

Краткое содержание заданий:

I

Задание 1. Отображение отдельных каналов. Преобразование цветовых пространств.

Задание 2. Написание собственной функции преобразования цветового пространства без использования функции cvtColor().

Задание 3. Поиск объекта по цвету.

Задание 4. Цветовая коррекция изображения.

II

Задание 5. Аффинные преобразования с использованием функции warpAffine() и warpPerspective().

Задание 6. Самостоятельная реализация аффинных преобразований без использования библиотек.

Задание 7. Изучение функции cv2.getAffineTransform().

Задание 8. Изучение функции cv2.getPerspectiveTransform().

Все задания выполняются на языке программирования Python!

1. **Цветовые пространства. Преобразование между цветовыми пространствами. Поиск объекта по цвету.**
2. **Перед выполнением данной части лабораторной работы ознакомьтесь с приложенными теоретическими сведениями из папки – “Цветовые пространства”.**

Рассмотрим, какие бывают световые схемы:

**BGR** — стандартное цветовое пространство OpenCV (используется по умолчанию). Является аналогом RGB пространства, но с другим порядком компонентов.

**RGB** — в данном методе отображения цвета мы раскладываем каждый пиксель на составляющие в виде красной (red), зеленой (green) и голубой (blue) компоненты.

**HSV** — это отображение создано для упрощения представление цвета человеком. Люди в случае описания цвета редко пользуются компонентами цвета, обычно они применяют такие слова как оттенок (hue), насыщенность (saturation) и интенсивность (value) цвета.

**XYZ** — модель была выведена в лаборатории CIE. Задача состояла в описании всех цветов, которые может видеть глаз человека. Компоненты X, Y, Z в данной модели описывают чувствительность среднестатистического наблюдателя к стандартным возбуждениям.

**GRAY** — черно-белое пространство, где яркость рассчитывается как средняя яркость всех трех компонент модели BGR.

Y = 0.299\*R + 0.587\*G + 0.114\*B;

**YUV** — данная схема пришла из телевидения, где компонента Y – значение яркости, а UV – двух цветоразностных сигналов.

**YCbCr, Y′CbCr, или Y Pb/Cb Pr/Cr**, также пишется как Y’CBCR или YCBCR — семейство цветовых пространств, которые используются для передачи цветных изображений в компонентном видео и цифровой фотографии. Y' — компонента яркости, CB и CR являются синей и красной цветоразностными компонентами.

**CIE Luv**, позволяющее определить различение цветов для человека с "усредненным" зрением, (т.е. различные люди неодинаково воспринимают разницу между цветами). Свое название пространство получило благодаря его компонентам L, u и v. Параметр L соответствует яркости цвета, u отвечает за переход от зеленого к красному (при увеличении), а при увеличении параметра v происходит переход от синего к фиолетовому. Если u и v равны 0, то, меняя L, получаем цвета, являющиеся градациями серого.

Под аббревиатурой **LAB** обычно понимается цветовое пространство CIE L\*a\*b\*, которое на данный момент является международным стандартом. Основной целью при разработке CIELAB было устранение нелинейности системы CIE XYZ с точки зрения человеческого восприятия. В системе CIE L\*a\*b координата L означает светлоту (в диапазоне от 0 до 100), а координаты a,b – означают позицию между зелёным-пурпурным, и синим-жёлтым цветами

Перевод изображения из одного цветового пространства в другое осуществляется по определенным формулам в зависимости от исходного и конечного цветового пространства.

Для перевода изображения в другое цветовое пространство с помощью библиотеки **OpenCV** достаточно написать следующую строчку:

*img = cv.cvtColor(img,cv.COLOR\_BGR2RGB)*

Функция img = cv.cvtColor() принимает на вход изменяемое изображение и «направление» для перехода: cv.COLOR\_BGR2RGB. Первым указывается текущий «формат представления» (BGR), далее символ «2» (созвучно с английским «to», которое интерпретируется как русское «в») после конечный «формат представления». Получается, что мы вводим команду перевести формат BGR в RGB. Получаем код:

import sys

import numpy as np

import cv2 as cv

fn = "KOT.jpg" # путь к файлу с картинкой

img = cv.imread(fn) # загрузка изображения

img = cv.cvtColor(img,cv.COLOR\_BGR2RGB)

cv.imshow('KOT', img) # выводим итоговое изображение в окно

cv.waitKey() # окно изображения открыто пока не нажмем какую-нибудь клавишу

cv.destroyAllWindows()

Все «направления» для перехода можно найти по ссылке:

<https://huningxin.github.io/opencv_docs/d7/d1b/group__imgproc__misc.html#ga4e0972be5de079fed4e3a10e24ef5ef0>

Основные из них:

1. COLOR\_BGR2RGB
2. COLOR\_BGR2XYZ
3. COLOR\_BGR2YCrCb
4. COLOR\_BGR2Lab
5. COLOR\_BGR2Luv
6. COLOR\_BGR2HLS
7. COLOR\_BGR2YUV
8. COLOR\_BGR2HSV
9. **Преобразование цветового пространства изображения**

**Задание 1.** Напишите код на языке программирования Python c применением библиотеки OpenCV, реализующий следующие действия:

1.1. Загрузить произвольное изображение с диска или по url ссылке;

1.2. Отобразить на экране исходное изображение;

1.3. Отобразить каждый из каналов (R, G, B) по отдельности, используя функцию cv2.split;

1.4. Отобразить исходное изображение обнулив по отдельности канал R, канал G, канал B.

#Понадобятся следующие функции библиотек OpenCV и numPy

#Создание пустого изображения, используя массивы numPy

*new\_image = np.zeros((высота,ширина, количество каналов), dtype=np.uint8)*

#Разделение изображения на каналы

*Split()*

#Создание изображения из отдельных каналов

*Merge()*

1.5. Преобразовать загруженное изображение в цветовое пространство из списка (*по номеру в списке подгруппы/группы*):

1. COLOR\_BGR2RGB
2. COLOR\_BGR2XYZ
3. COLOR\_BGR2YCrCb
4. COLOR\_BGR2Lab
5. COLOR\_BGR2Luv
6. COLOR\_BGR2HLS
7. COLOR\_BGR2YUV
8. COLOR\_BGR2HSV

1.5.1. Повторить пункты 1.2, 1.3, 1.4 для полученного изображения;

1.6. Сохранить все полученные изображения на диск.

**Задание 2.**

2.1.По номеру в списке подгруппы/группы выбрат вариант преобразования из списка:

1. RGB ↔ HSV
2. RGB ↔ HLS
3. RGB ↔ CIE L\*a\*b\*
4. RGB ↔ CIE L\*u\*v\*

Для выбранного варианта:

2.2. Изучить формулы преобразования по ссылке:

*https://docs.opencv.org/3.4/de/d25/imgproc\_color\_conversions.*html

2.3. Реализовать изученные формулы на языке Python и преобразовать цветовое пространство изображения используя свой программный код.

2.4. Сравнить полученные результаты с аналогичным преобразованием функцией cv.cvtColor библиотеки OpenCV. Сохраните все полученные изображения на диск.

**Помните, что opencv считывает изображения как BGR, а не как RGB! Вы можете использовать метод item() для доступа и itemset для внесения изменений в изображение, либо прямой доступ к массивам numpy - image[h,k].**

1. **Поиск объекта по цвету**

В системах с компьютерным зрением одной из ключевых задач является идентификация объектов, попавших в поле зрения фото- или видеокамеры. Наиболее эффективным методом поиска объекта считается определение объекта по его цвету. Это делается с помощью цветовой маски. Цветовая маска при наложении на исходное изображение позволяет отфильтровать цвета согласно выбранному цветовому диапазону. Области изображения, которые не соответствуют указанному диапазону цветов, отображаться не будут. После применения цветовой маски эти области будут окрашены в черный цвет, а все, что соответствует цветовому диапазону, — в белый цвет. Это сильно упрощает задачу поиска объектов. Таким образом, подбирая нужный диапазон цветов, можно вывести на экран только искомый объект или группу объектов.

Рассмотрим задачу поиска объекта по цвету на графическом изображении. Для этой задачи возьмем изображение из Интернета:

<https://up.kukudesk.com/pic/58/b7/e1/58b7e1f00ba76d6edfd74e26b19d3909.jpg>



Давайте попробуем разобрать решение задачи на приведенном ниже программном коде.

import cv2

# загрузка изображения

image = cv2.imread('D://ax92e3q1SemIDpzFtucg\_tennis\_ball\_186332.jpg')

#image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

cv2.imshow("Original", image)

# задаем границы диапазона:

# нижнюю

low\_color = (0,150,0)

# и верхнюю

high\_color = (255,255,255)

# наложение цветовой маски на исходное изображение,

# результат присваиваем переменной only object

only\_object = cv2.inRange(image, low\_color, high\_color)

# вывод отфильтрованного изображения на экран

cv2.imshow('only object', only\_object)

cv2.waitKey(0)

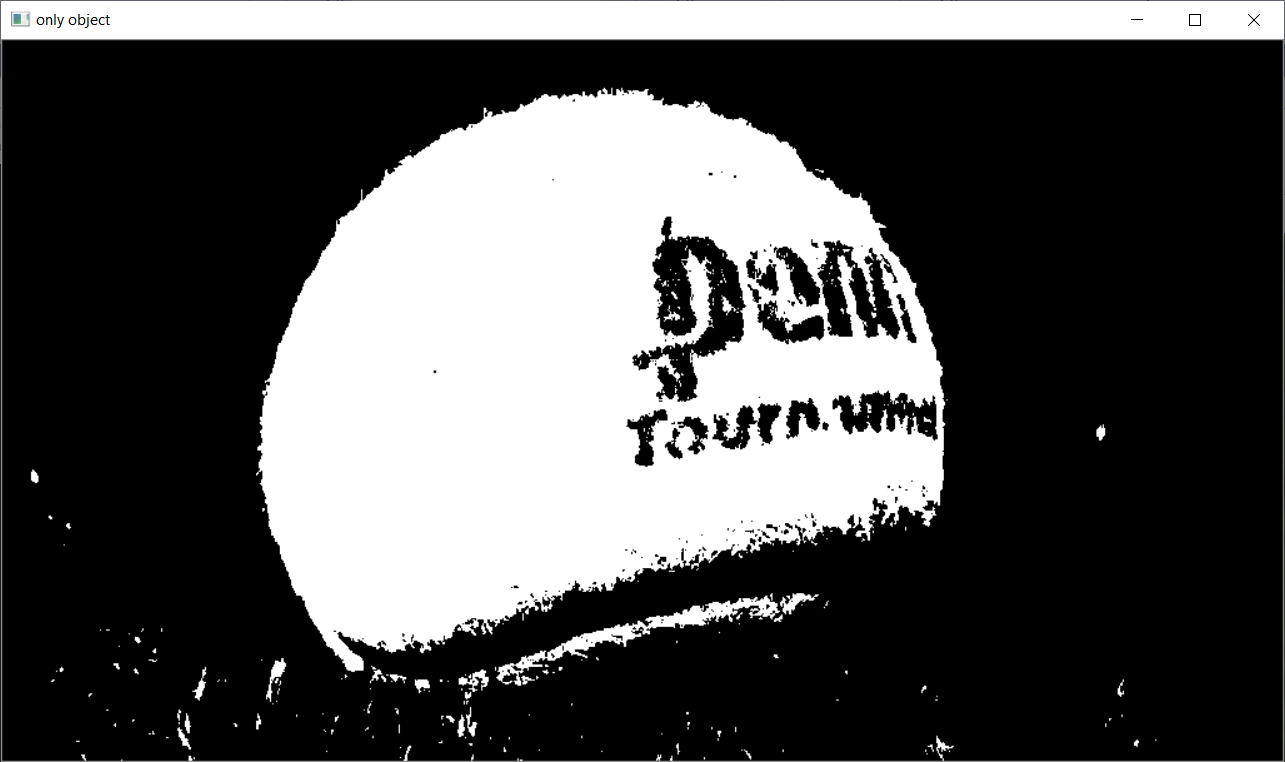
После загрузки изображения (функция cv2.imread()) требуется подобрать нижнюю и верхнюю границы цветового диапазона для искомого объекта. Поскольку цвет пикселей изображения задается смешением компонент красного, зеленого и синего цветов (Red- Green-Blue — цветовое RGB-пространство), границы диапазона будут определяться совокупностью трех значений каждой компоненты по шкале от 0 до 255.

Искомый объект на изображении представляет собой группу пикселей различных желтых оттенков, поэтому в диапазоне между нижней и верхней границей будут преобладать компоненты зеленого и синего цветов. Однако красная компонента тоже будет присутствовать, так как мячик все-таки не идеально желтого цвета. В программе границы диапазона задаются в переменных low\_color, high\_color. В данном случае используется цветовое пространство BGR: (Синий (Blue), Зеленый (Green), Красный (Red)) по шкале от 0 до 255.

Затем с помощью функции cv2. inRange () на исходное изображение (переменная image) накладывается цветовая BGR-маска и отфильтровываются все цвета вне диапазона low\_color, high\_color. Функция cv2.inRange() принимает на входе цветное изображение и возвращает на выходе черно-белое изображение, где белыми пикселями отрисованы области, цвета которых попадали в заданный диапазон.

Ниже показан результат работы программы. Как видно из рисунка, кроме теннисного мячика, присутствует большое количество белых вкраплений. Они тоже попали в наш цветовой диапазон.





**Поиск объекта в цветовом пространстве HSV**

Чтобы найти теннисный мячик, переведем изображение с ним в цветовое пространство HSV и попытаемся найти мячик на картинке. Ниже представлен пример программного кода:

import cv2

# загрузка изображения

image = cv2.imread('D://ax92e3q1SemIDpzFtucg\_tennis\_ball\_186332.jpg')

cv2.imshow("Original", image)

# конвертируем исходное изображение в HSV,

# результат присваиваем переменной hsv img

hsv\_img = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

# нижняя граница — это темный ненасыщенный цвет

color\_low = (35,120,170)

# верхняя граница — это яркий насыщенный цвет

color\_high = (50,255,255)

# наложение цветовой маски на HSV-изображение,

# результат присваиваем переменной only object

only\_object = cv2.inRange(hsv\_img, color\_low,color\_high)

# результат присваиваем переменной only\_object

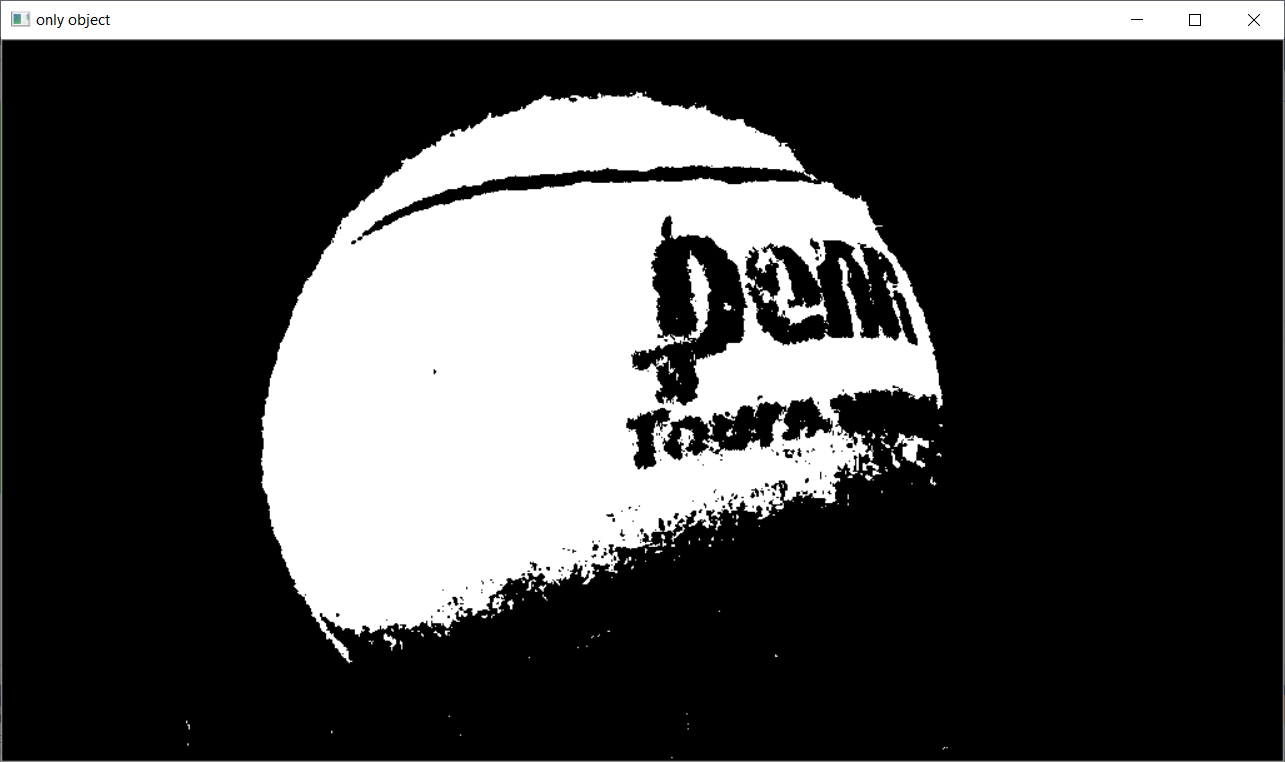
# вывод отфильтрованного изображения на экран

cv2.imshow('only object', only\_object)

cv2.waitKey(0)

Для поиска цветного объекта необходимо подобрать диапазон color\_low = (H1,S1,V1), color\_high = (H2,S2,V2). С помощью значений H1, H2 (H1 < H2) определяются цветовые оттенки, в которые окрашен искомый объект (в OpenCV3 шкала цветового тона (H) лежит в значениях 0...180 , шкалы (S) и (V) — в значениях 0...255).

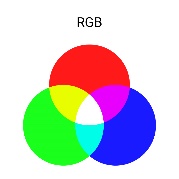
Результат работы программы представлен на рисунке ниже.



**Задание 3.**

1. Загрузите произвольное изображение с выделяющимися по цвету яркими объектами *(*

*НЕ использовать изображение из примера!!!*

*НЕ выбирать изображения с предметами чистого красного, зеленого, синего цветов, по типу вот такого: )*

и подберите параметры low\_color, high\_color для выделения одного из объектов. Выполните вывод на экран исходного и обработанного изображений. Сохраните их на диск. Произведите указанные действия для цветового пространства RGB; второе цветовое пространство выбрать из списка согласно своему номеру в списке группы/подгруппы:

1. HLS
2. HSV

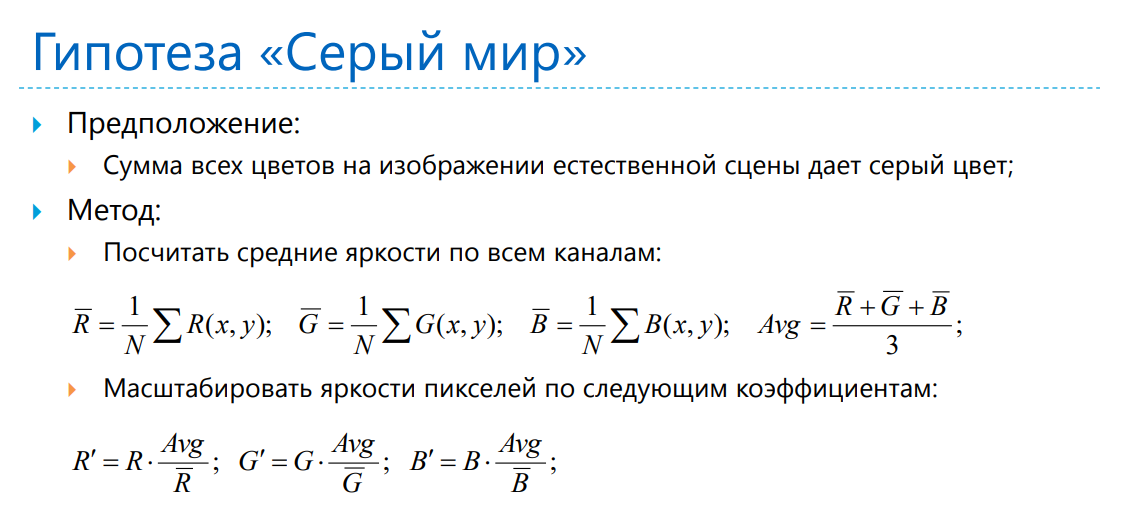
и провести поиск объекта по цвету в цветовом пространстве согласно варианту.

**Цветовая коррекция**

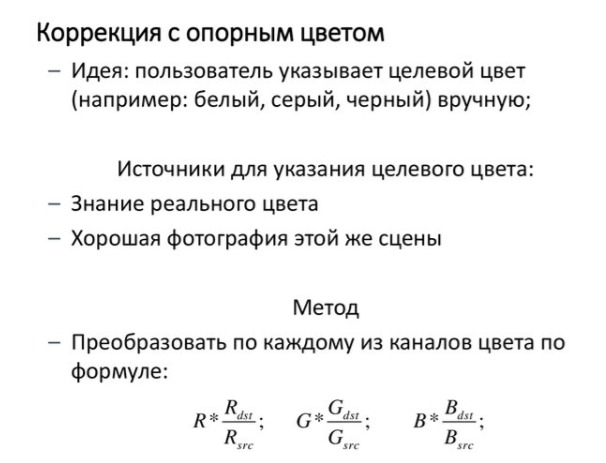
Коррекция с опорным цветом

Цветовой баланс фотографии указывает на соотношение различных цветов между собой. Цветокоррекция производится с целью улучшения цветового баланса, путем изменения цветовых параметров пикселей оригинала изображения. Существуют различные методы, благодаря которым возможно улучить качество изображения. Рассмотрим широко известные методы коррекции цветового баланса.

1. Гипотеза “Серый мир”

****

2. Коррекция с опорным цветом

****

**Задание 4.**

На ресурсах Яндекс или Google Картинки найти изображение с нарушенным балансом белого, например, по запросу “ нарушение баланса белого”.

Примеры:



Если изображение состоит из двух частей (до цветокоррекции и после), как на картинках из примера, то вырезать только ту часть где баланс белого нарушен.

Реализовать приведенные выше два алгоритма цветовой коррекции на языке программирования Python и применить их для выбранного изображения. Отобразить на экран и сохранить на диск исходные и скорректированные изображения.

1. **Геометрические преобразования изображений.**
2. **Перед выполнением данной части лабораторной работы ознакомьтесь с приложенными теоретическими сведениями из папки – “Геометрические преобразования”.**

Изучить необходимые для выполнения лаб. раб. функции библиотеки OpenCV:

**warpAffine()**

**warpPerspective()**

Ссылки:

<https://huningxin.github.io/opencv_docs/da/d6e/tutorial_py_geometric_transformations.html>

<https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#ga0203d9ee5fcd28d40dbc4a1ea4451983>

<https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87>

Документ из папки **“Геометрические преобразования”**:

*Обработка изображений с использованием OpenCV в Python.pdf*

Аффинные преобразования:

1. Масштабирование
2. Сдвиг
3. Поворот
4. Скос

**Задание 5.**

- загрузить произвольное изображение с диска или по url ссылке;

- отобразить на экране исходное изображение;

- применить к изображению аффинные преобразования, приведенные выше (параметры выбрать произвольно), с использованием функций warpAffine() и warpPerspective() библиотеки OpenCV;

- отобразить преобразованные изображения на экран

- сохранить все полученные изображения на диск.

**Задание 6.**

Выполнить задание 4, но не используя для аффинных преобразований библиотеку OpenCV (либо любую другую). Реализовать аффинные преобразования самостоятельно!Для упрощения допускается в начале преобразовать изображение в одноканальное в оттенках серого. Сравните результаты с применением аналогичных преобразований с помощью библиотеки OpenCV. При самостоятельных реализациях аффинных преобразований изображения будут иметь искажения (кроме операции сдвига). Объясните почему. Каких дополнительных действий не хватает для исправления этих искажений?

Вы можете использовать метод item() для доступа и itemset для внесения изменений в изображение, либо прямой доступ к массивам numpy - image[h,k].

При выполнении задания может понадобится:

*Создать пустое изображение в оттенках серого*

*import numpy as np*

*gray\_value=127*

*gray\_image = np.full(height, width), gray\_value, dtype=np.uint8)*

*Трехканальное*

*import cv2*

*import numpy as np*

*blank\_image = np.zeros((height,width,3), dtype = np.uint8)*

**Задание 7.**

Изучите функцию

cv2.getAffineTransform

Привести пример её использования с последующим применением функции warpAffine() для произвольного изображения. Исходное и полученное изображения показать на экран и сохранить на диск. Опишите какового назначение функции getAffineTransform.

**Перспективное Преобразование**

**Задание 8.**

Изучить функцию cv2.getPerspectiveTransform().

Применить функцию cv2. warpPerspective() к произвольному изображению передав ей матрицу, полученную функцией getPerspectiveTransform.

Исходное и полученное изображения показать на экран и сохранить на диск. Объясните для чего служат указанные функции и для чего могут применятся.

! Оформить отчет по проделанной работе!