Лабораторная работа №5

Библиотека Open CV

Каждое изображение представлено набором пикселей, то есть матрицей значений пикселей. Для изображения в градациях серого значения пикселей варьируются от 0 до 255, и они представляют интенсивность этого пикселя. Например, если у вас есть изображение размером 20 x 20, оно будет представлено матрицей 20 x 20 (всего 400-пиксельных значений).

Если вы имеете дело с цветным изображением, вы должны знать, что оно будет иметь три канала - Красный, Зеленый и Синий (RGB). Следовательно, было бы три таких матрицы для одного изображения.

Прежде чем перейти к использованию обработки изображений в приложении, важно получить представление о том, какие операции попадают в эту категорию и как выполнять эти операции. Эти операции, наряду с другими, будут использоваться позже в наших приложениях. Итак, давайте вернемся к этому.

Для этой статьи мы будем использовать следующее изображение:

Вы, вероятно, заметили, что изображение в настоящее время цветное, что означает, что оно представлено тремя цветными каналами: красным, зеленым и синим. Мы будем преобразовывать изображение в градации серого, а также разбивать изображение на отдельные каналы, используя код ниже.

Поиск деталей изображения

После загрузки изображения с помощью функции imread() мы можем получить некоторые простые свойства, такие как количество пикселей и размеры:

In [1]:
```
import cv2
from matplotlib import pyplot as plt
img=cv2.imread('v1.jpg')
print("Число пикселей: "+str(img.size))
print("Размеры изображения: "+str(img.shape))
```

```
Число пикселей: 1980000
Размеры изображения: (600, 1100, 3)
```

Разделение изображения на отдельные каналы

Теперь мы разделим изображение на его красные, зеленые и синие компоненты с помощью OpenCV и отобразим их:

```
b = img.copy()
# set green and red channels to 0
b[:, :, 1] = 0
b[:, :, 2] = 0

g = img.copy()
# set blue and red channels to 0
g[:, :, 0] = 0
g[:, :, 2] = 0

r = img.copy()
# set blue and green channels to 0
r[:, :, 0] = 0
r[:, :, 1] = 0

# RGB - Blue
cv2.imshow('B-RGB', b)

# RGB - Green
cv2.imshow('G-RGB', g)

# RGB - Red
cv2.imshow('R-RGB', r)

cv2.waitKey(0)
```

Изображения отображаются в отдельных окнах.

При чтении способом выше изображение находится в цветовом пространстве не RGB (как все привыкли), а BGR. Возможно, в начале это не так важно, но как только вы начнёте работать с цветом — стоит знать об этой особенности. Есть 2 пути решения:

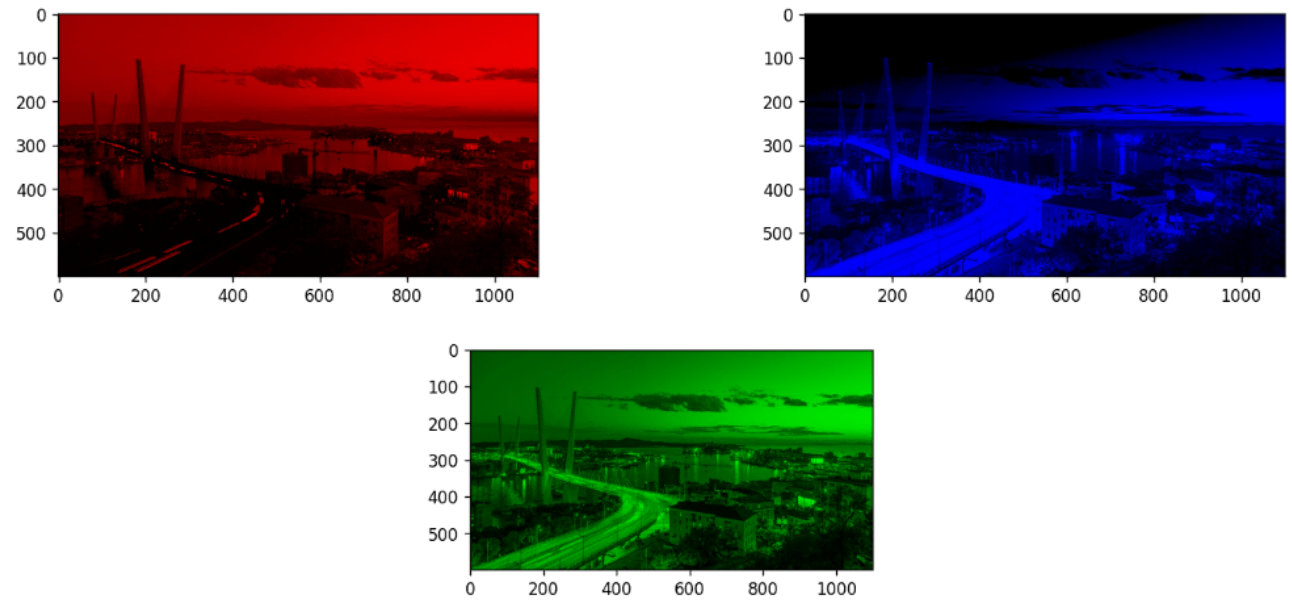
1. Поменять местами 1-й канал (R — красный) с 3-м каналом (B — синий), и тогда красный цвет будет `(0,0,255)`, а не `(255,0,0)`.
2. Поменять цветовое пространство на RGB:

```
rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

И тогда в коде работать уже не с `image`, а с `rgb_image`.

Можно выводить и непосредственно в Jupyter notebook с помощью matplotlib, но надо учесть, что каналы поменяны местами

```
fig = plt.figure(figsize=(15, 10), dpi=120)
ax_1 = fig.add_subplot(3, 2, 1)
plt.imshow(b)
ax_2 = fig.add_subplot(3, 2, 2)
plt.imshow(r)
ax_3 = fig.add_subplot(3, 3, 5)
plt.imshow(g)
plt.show()
# каналы  r и b меняются местами для  plt.imshow     и  cv2.imshow
```



Задание: загрузить картинку с участием себя, определить параметры изображения, разделить на каналы , вывести двумя способами cv2 и matplotlib.

Распознавание лиц

`detectMultiScale` — общая функция для распознавания как лиц, так и объектов. Чтобы функция искала именно лица, мы передаём ей соответствующий каскад.

Функция `detectMultiScale` принимает 4 параметра:

1. Обрабатываемое изображение в градации серого.
2. Параметр `scaleFactor`. Некоторые лица могут быть больше других, поскольку находятся ближе, чем остальные. Этот параметр компенсирует перспективу.

3. Алгоритм распознавания использует скользящее окно во время распознавания объектов. Параметр `minNeighbors` определяет количество объектов вокруг лица. То есть чем больше значение этого параметра, тем больше аналогичных объектов необходимо алгоритму, чтобы он определил текущий объект, как лицо. Слишком маленькое значение увеличит количество ложных срабатываний, а слишком большое сделает алгоритм более требовательным.
4. `minSize` — непосредственно размер этих областей.

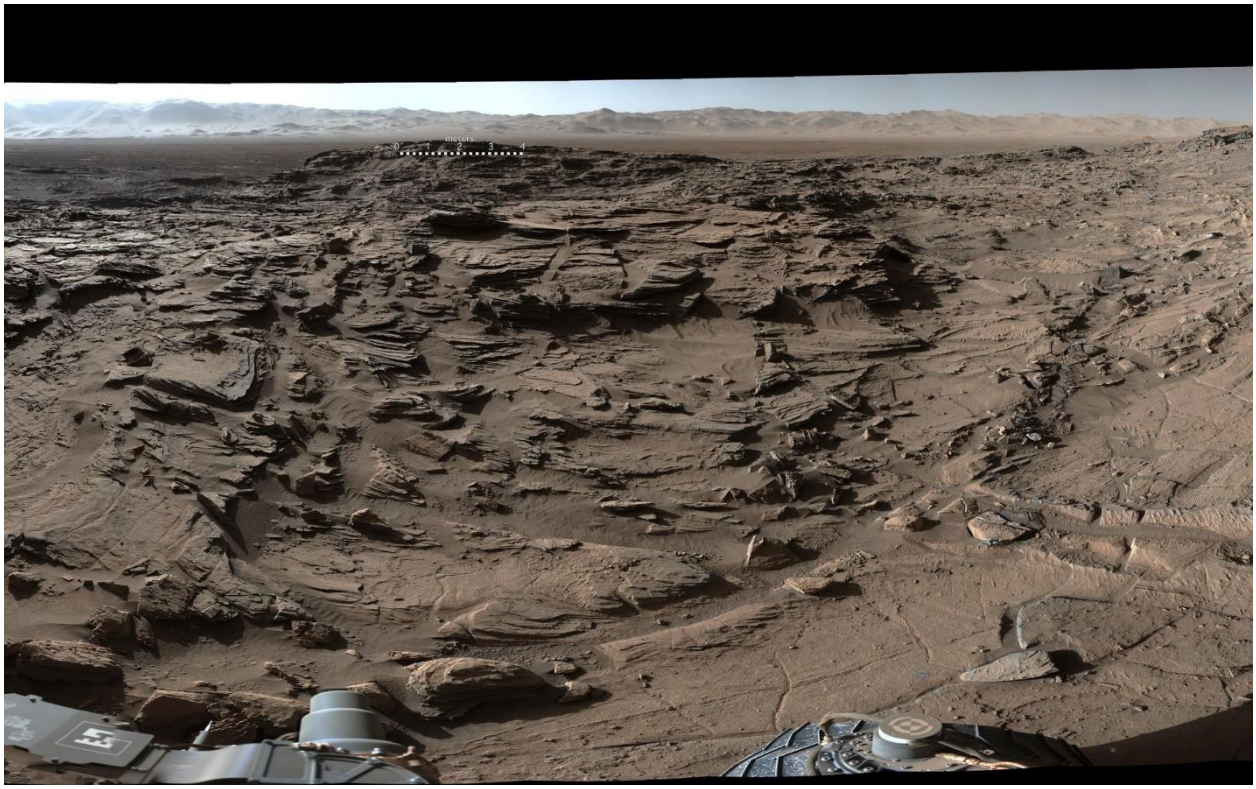
```
import cv2
from matplotlib import pyplot as plt
image_path = "faces.jpg"
face_cascade = cv2.CascadeClassifier('c:\python38-32\Lib\site-packages\cv2\data\haarcascade_frontalface_default.xml')
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, scaleFactor= 1.2, minNeighbors= 15,minSize=(50, 50))

faces_detected = "Лиц обнаружено: " + format(len(faces))
print(faces_detected)
# Рисуем квадраты вокруг лиц
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 255, 0), 2)
fig = plt.figure(figsize=(18, 14), dpi=120)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
#viewImage(image,faces_detected)
```

```
Лиц обнаружено: 17
```

Задание: загрузить коллективную картинку с участием себя, детектировать все лица по возможности, подбирая параметры распознавания.

Наложение изображений

Наложим на панораму Марса ***mars1.jpg*** файлы с прозрачной подложкой, заодно его отмасштабировав



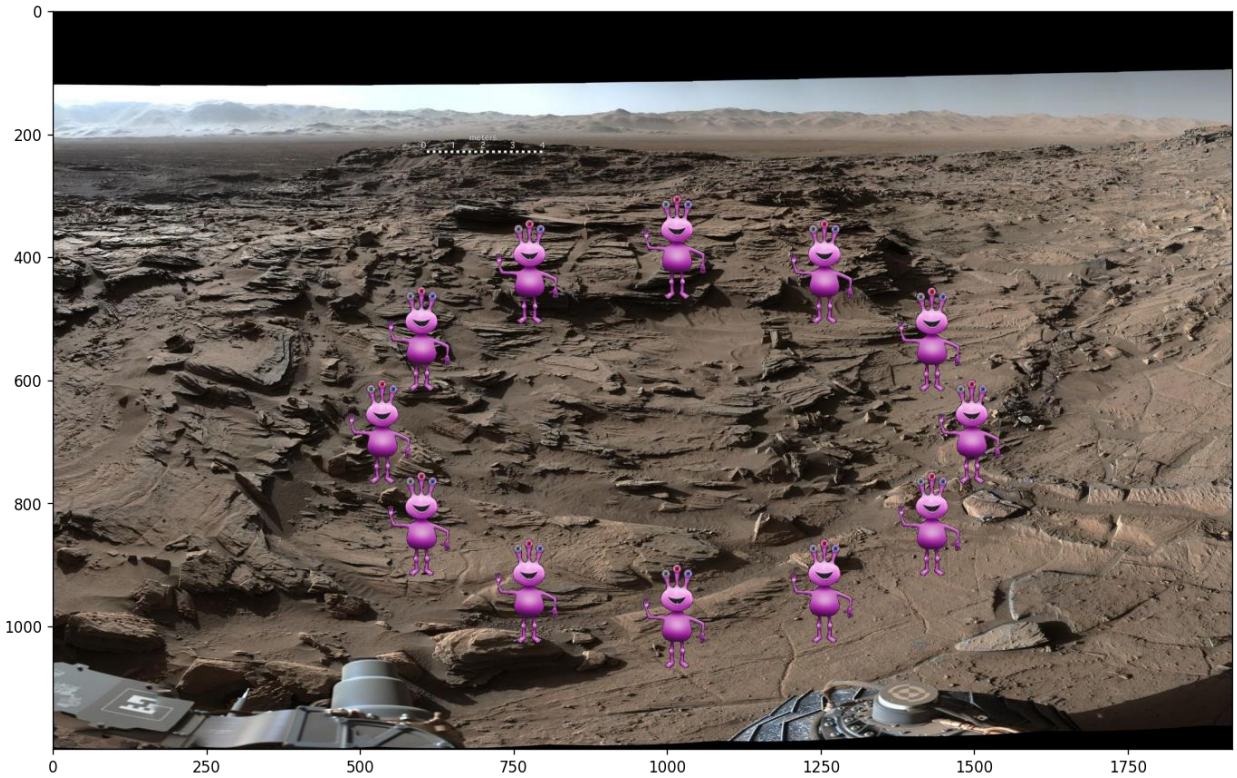
Изображение в формате ***uf2.png*** с прозрачным фоном

Ввод [1]:

```
import numpy as np, cv2
from matplotlib import pyplot as plt  # подключаем библиотеки
img1 = cv2.imread('mars1.jpg')
img2 = cv2.imread('uf2.png', cv2.IMREAD_UNCHANGED)
h1, w1 = img1.shape[:2]
h2, w2 = img2.shape[:2]
```

```
    b = np.copy(img1)
    h3, w3 = b.shape[:2]
scaling_factor=0.3
# масштабирование
img3 = cv2.resize(img2, None, fx=scaling_factor, fy=scaling_factor, interpolation=cv2.INTER_AREA)
y=100
x=200

for t in range(0,350,30): # Размещаем несколько картинок
        y=int(h3/2+h3/4*np.cos(t*np.pi/180)) #задаем координаты
        x=int(w3/2+w3/4*np.sin(t*np.pi/180))

        for i in range(y,(y+ img3.shape[0])):   #задаем пиксели наложения
            for k in range(x,(x + img3.shape[1])):
                if img3[i-y,k-x,3]!=0:  #если подложка прозрачна
                   b[i, k,:]=img3[i-y,k-x,:3]   #накладываем изображение


fig = plt.figure(figsize=(14, 10), dpi=120)
plt.imshow(cv2.cvtColor(b, cv2.COLOR_BGR2RGB))
```

Результат



Задание: загрузить свою картинку, наложить другое изображение по собственной схеме размещения с масштабированием.

Хромакéй

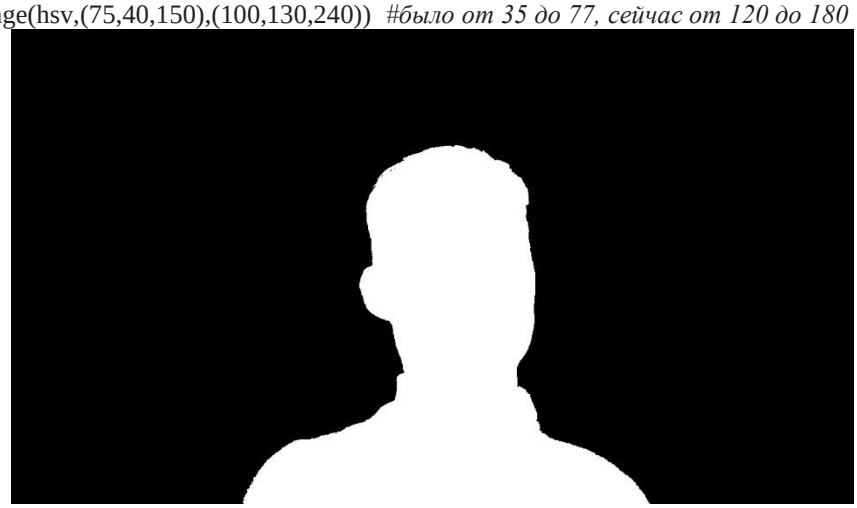
Хромакéй— технология совмещения двух и более изображений или кадров в одной композиции, цветовая электронная рирпроекция, использующаяся на телевидении и в современной цифровой технологии кинопроизводства. С помощью хромакея можно поместить людей или предметы на произвольный фон, снятый в другом месте.

Кадр из видео (1.jpg)

Фон (2.jpg)

```
import cv2
from matplotlib import pyplot as plt
import numpy as np

image = cv2.imread("1.jpg")

fon=cv2.imread("2.jpg")
h1, w1 = image.shape[:2]
print(h1, w1)
h2, w2 = fon.shape[:2]
```

```
print(h2, w2)
scaling_factor=w1/w2

fon = cv2.resize(fon, None, fx=scaling_factor, fy=h1/h2, interpolation=cv2.INTER_AREA)

print(fon.shape)
# Преобразование изображения BGR в HSV для обработки
hsv = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)

 #В диапазоне зеленого значения изображения, за исключением символов изображения, все белое 255, а символы изображения черные 0
mask = cv2.inRange(hsv,(75,40,150),(100,130,240))  #было от 35 до 77, сейчас от 120 до 180
```



```
fon1= cv2.bitwise_and(fon,fon,mask=mask)

 #At изображение изображения, затем символ изображения белый 255, а другие - черные 0
cv2.bitwise_not(mask,mask)

result = cv2.bitwise_and(image,image,mask=mask)

res2 = cv2.add(result, fon1)

plt.imshow(cv2.cvtColor(res2, cv2.COLOR_BGR2RGB))
plt.colorbar=mask
plt.show()
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()

#cv2.imwrite("mask.jpg",mask)
cv2.imwrite("res2.jpg",result)
```

Результат

Задание: повторить хромакей со своими видео и фоном.