Программирование

на языке

Python

Блок 3:

Прикладное программирование.

В. Б. Пикулев,
доцент КФТТ ПетрГУ.

Scilink.ru, 2024

OpenCV. Что это такое?

OpenCV (open source computer vision library) кроссплатформенная библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым исходным кодом. Реализована на языке C/C++. Может свободно использоваться в академических и коммерческих целях. Фактически эта библиотека одна из первых содержала реализации алгоритмов искусственного интеллекта, применяемые для распознавания образов.

```
# Windows
> pip install opencv-python

# Linux
> apt-get install
libopencv-dev python-opencv
```

Примеры модулей

- **Core Functionality** (основная функциональность) — определяет основные структуры данных и функции библиотеки, которые используются в других модулях.
- **Image Processing** (обработка изображений) — позволяет работать со статичными изображениями: простыми картинками в форматах PNG, JPG и других.
- **Video I/O** (ввод и вывод видео) — позволяет считывать и обрабатывать видеофайлы.
- **Video Analysis** (анализ видео) — используется для отслеживания движений объектов и работы с фоном.
- **Camera Calibration and 3D Reconstruction** (калибровка камеры и 3D-реконструкция) — работает с геометрией объектов, позволяя создавать их 3D-модели на основе нескольких изображений или видео.
- **2D Features Framework** (фреймворк двумерных особенностей) — определяет фрагменты изображения, которые отличаются от других, запоминая их контуры, и может находить похожие среди них.
- **Object Detection** (обнаружение объектов) — находит объекты, например лица, автомобили, птиц и другое.

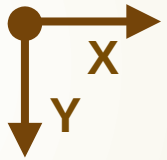
Изображения.

Black	rgb(0, 0, 0)
White	rgb(255, 255, 255)
Red	rgb(255, 0, 0)
Blue	rgb(0, 0, 255)
Green	rgb(0, 255, 0)
Yellow	rgb(255, 255, 0)
Magenta	rgb(255, 0, 255)
Cyan	rgb(0, 255, 255)
Violet	rgb(136, 0, 255)
Orange	rgb(255, 136, 0)

Пиксель - цветовой элемент матрицы изображения: цвет пикселя определяется значением в палитре изображения, для полноцветных (TrueColor) изображений это три байта — **R**ed, **G**reen и **B**lue. Одинаковые значения R, G и B дают оттенки серого. В графических файлах матрица изображения обычно представлена в закодированном виде ради уменьшения размера файла; если в файле реализовано сжатие с потерями — изображение будет существенно отличаться от оригинальной матрицы.

Формат	*Макс. число бит/пиксель*	*Макс. число цветов*	*Макс. размер изображения, пикселей*	*Методы сжатия*	*Несколько изображений*
BMP	24	16777216	65535 x 65535	RLE	-
GIF	8	256	65535 x 65535	LZW	+
JPEG	24	16777216	65535 x 65535	JPEG (по Хаффману)	-
PNG	48	281474976710656	2147483647 x 2147483647	Deflate (вариант LZ77)	-
TIFF	24	16777216	всего 4294967295	LZW, RLE и другие	+

4 Представление изображений.




```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TER

      Ширина изображения: 1024
      Количество каналов: 3

      Первые девять пикселей:
      [[[ 9 11 19]
  [ 6  7 17]
  [ 3  5 16]]

 [[ 5  8 16]
  [ 3  5 15]
  [ 1  5 16]]

 [[ 0  8 15]
  [ 1  8 17]
  [ 2  8 19]]]
```

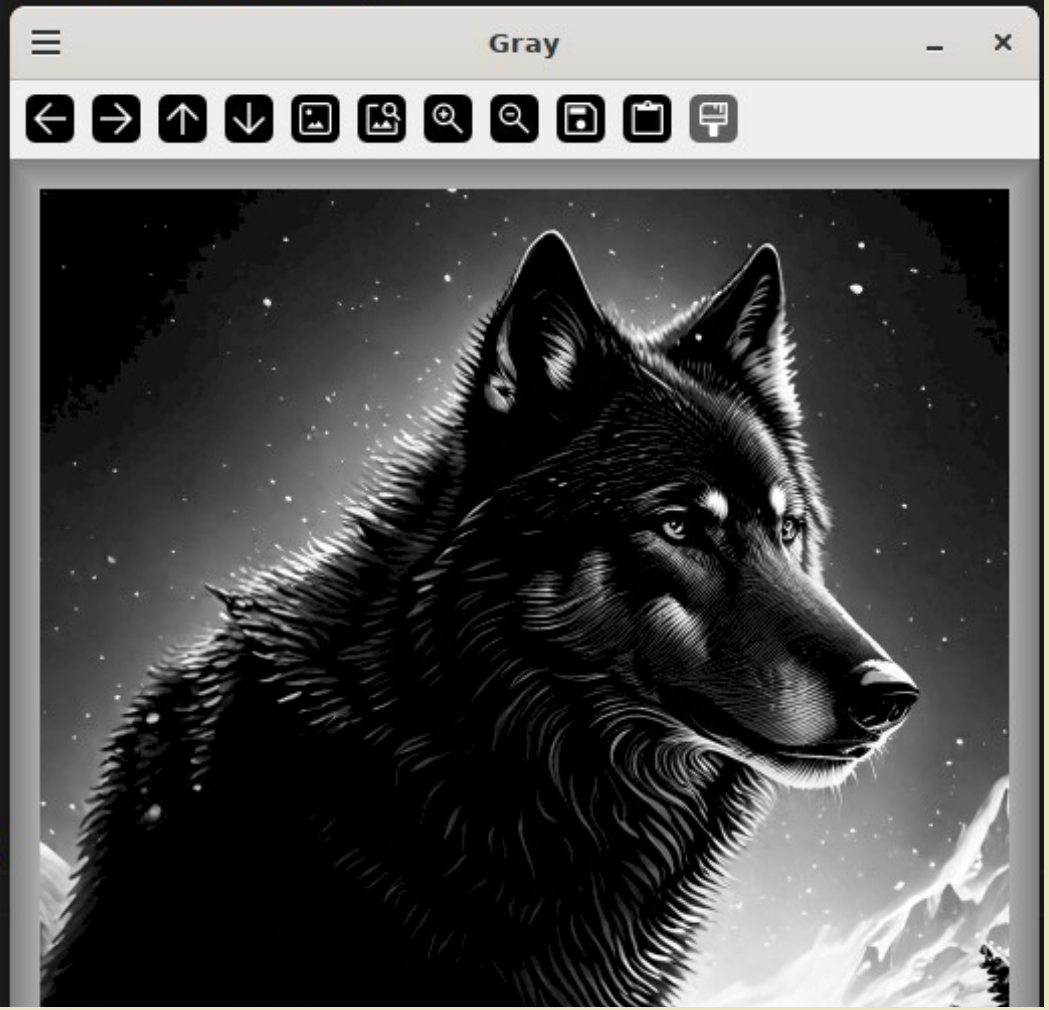

```python
OpenCV > ocv_01.py > ...
# читаем изображение из файла
import cv2
import os
wpath = os.path.dirname(os.path.realpath(__file__))

img = cv2.imread(wpath +'/images/6.jpg', cv2.IMREAD_COLOR)
# выводим размеры изображения
print(f'''
        Высота изображения: {img.shape[0]}
        Ширина изображения: {img.shape[1]}
        Количество каналов: {img.shape[2]}

        Первые девять пикселей:
        {img[0:3, 0:3]}
''')

# показываем изображение
cv2.imshow("Sorcerer's Doll", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

Изображение представляется в виде вложенных списков. Если режим "в оттенках серого", то уровень вложенности 2, если цветное, то — 3. Верхний уровень (строки) — координата Y, далее — столбцы (координата X), самый глубокий уровень — цвета (для IMREAD_COLOR последовательно G, B, R).

Ещё пример

OpenCV > ocv_01a.py > ...

```python
# читаем изображение из файла
import cv2
import os
wpath = os.path.dirname(os.path.realpath(__file__))

# читаем изображение из файла
img = cv2.imread(wpath +'/images/1.png', cv2.IMREAD_GRAYSCALE)

# делаем рамку на изображении
for i in range(15):
    for j in range(i, img.shape[0]-i):
        img[i][j] = 128 + i*3
        img[-1-i][j] = 128 + i*3
for i in range(15):
    for j in range(i, img.shape[1]-i):
        img[j][i] = 128 + i*3
        img[j][-1-i] = 128 + i*3

# показываем изображение
cv2.imshow("Gray", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# сохраняем изменённое изображение
# в новый файл
cv2.imwrite(wpath +'/images/gray.png', img)
```



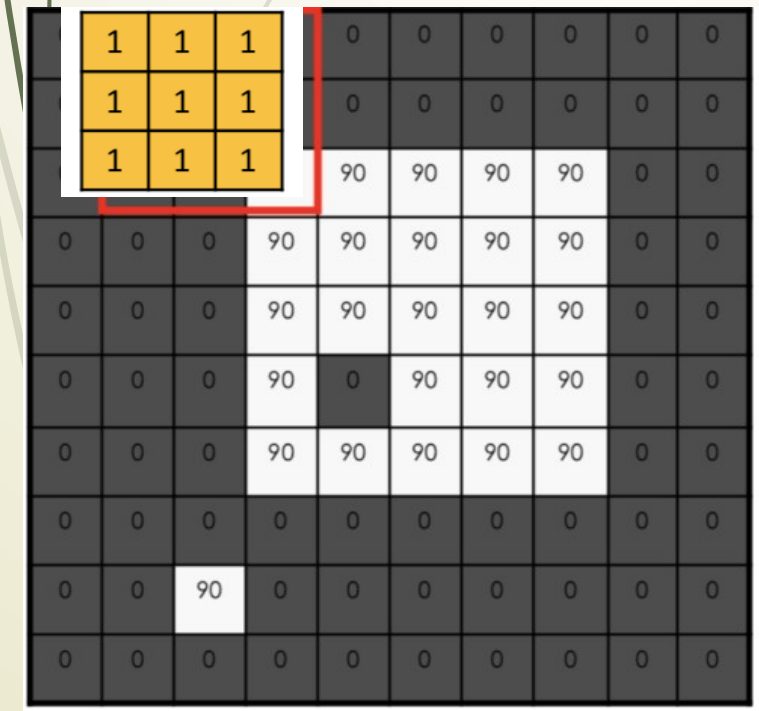
Работа с изображениями.

6

Иллюстрация применения фильтров:

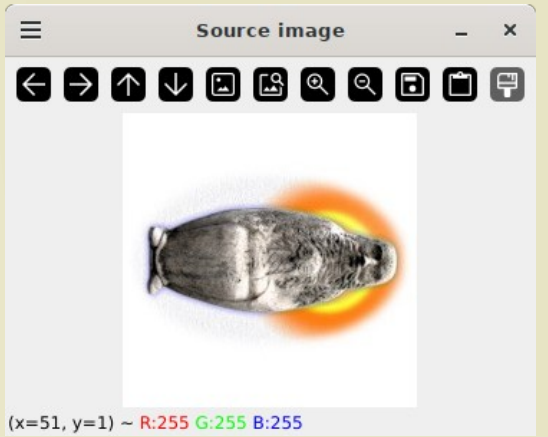
$$A_{m+1,n+1} = \frac{1}{S} \sum_{i=0}^{2} \sum_{j=0}^{2} M_{i,j} A_{m+i,n+j}$$

медианный фильтр




```python
# редактируем изображение
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
wpath = os.path.dirname(os.path.realpath(__file__))

img = cv2.imread(wpath +'/images/2.jpg')
cv2.imshow('Source image', img)
cv2.waitKey(0)

# поворачиваем изображение
img = cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE)
# масштабируем изображение
img = cv2.resize(img, (0,0), fx=2.5, fy=2.5,
    interpolation=cv2.INTER_CUBIC)
# кадрируем изображение
img = img[10:img.shape[0], 70:img.shape[1]-70]
# подписываем изображение
img = cv2.putText(img, 'KUL MASCOT',
    (90, img.shape[0]-15), cv2.FONT_HERSHEY_SIMPLEX,
    1, (30, 130, 230), 2, cv2.LINE_8)
# добавляем резкость
sharp_filter = np.array(
    [[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]]
    )
img = cv2.filter2D(img, ddepth=-1, kernel=sharp_filter)

# показываем результат
cv2.imshow('Result', img)
while True:
    if cv2.waitKey(0)==27:
        break
cv2.destroyAllWindows()
```

-1	-1	-1
-1	9	-1
-1	-1	-1



Операции с изображениями.

Примеры:

```
img = cv2.warpAffine(img, cv2.getRotationMatrix2D((img.shape[1]//2,
img.shape[0]//2), 45, 1.0), (img.shape[1], img.shape[0]))
cv2.putText(image, "Hello!", (x,y), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)
```

Поворот изображения	**cv2.getRotationMatrix2D()** - возвращает двумерный объект, в котором будет размещено преобразованное изображение (в случае трансформации и поворота). Аргументы: 1. координаты точки, относительно которой будет осуществлен поворот и изменение размера изображения; 2. угол поворота в градусах; 3. коэффициент трансформации.
	cv2.warpAffine() - выполняет поворот изображения и возвращает новое изображение. Аргументы: 1. имя переменной исходного изображения; 2. имя переменной двумерного объекта для преобразованного изображения; 3. список с размерами выходного изображения.
Зеркальное отражение изображения	**cv2.flip()** - создание зеркальной копии изображения. Аргументы: 1. имя переменной исходного изображения; 2. ось, относительно которой производится отражение.
Вывод текста на изображение	**cv2.putText()** - изображает текст. Аргументы: 1. имя переменной исходного изображения; 2. координаты начала текста; 3. вид шрифта; 4. размер шрифта; 5. цвет шрифта; 6. толщина линии букв.

Первый шаг к распознаванию образов.

8

Сегментация изображений

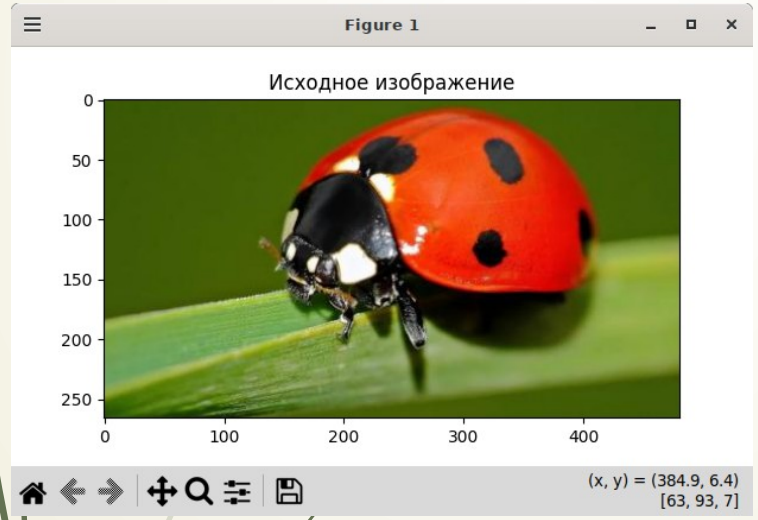
— процесс присвоения меток каждому пикселю изображения, так что пиксели с одинаковыми метками имеют общие визуальные характеристики. Все пиксели в сегменте похожи по некоторой характеристике или вычисленному свойству, например, по цвету, яркости или текстуре.

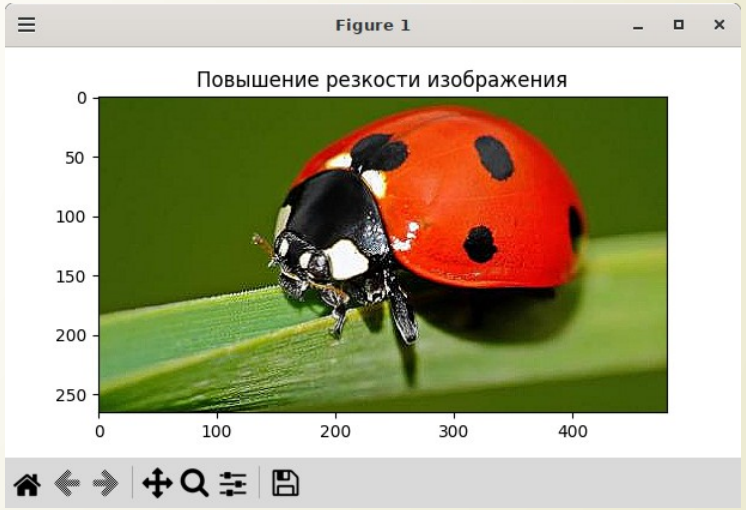
Цель сегментации заключается в упрощении и/или изменении представления изображения, чтобы его было проще и легче анализировать.

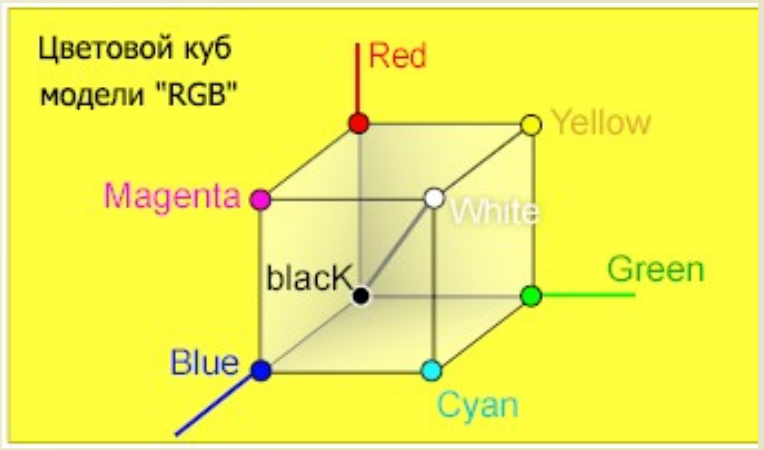
Один из самых простых методов сегментации — на основе сходного цвета. Близкий к нему метод — выделение краёв объектов (анализируется перепад яркости на границах областей).

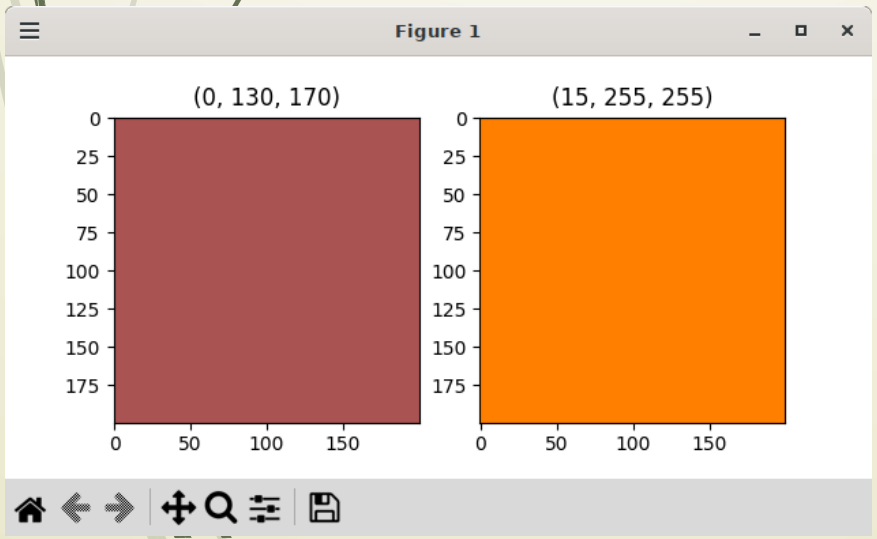
```python
# поиск областей сходного цвета

import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
wpath = os.path.dirname(os.path.realpath(__file__))

ladybug = cv2.imread(wpath+'/images/ladybag.png')
ladybug = cv2.cvtColor(ladybug, cv2.COLOR_BGR2RGB)
plt.imshow(ladybug)
plt.title('Исходное изображение')
plt.show()

# Повышение резкости
kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
ladybug = cv2.filter2D(ladybug, -1, kernel)
plt.imshow(ladybug)
plt.title('Повышение резкости изображения')
plt.show()

# Переход в палитру HSV
hsv_ladybug = cv2.cvtColor(ladybug, cv2.COLOR_RGB2HSV)

```

Иллюстрации работы алгоритма.

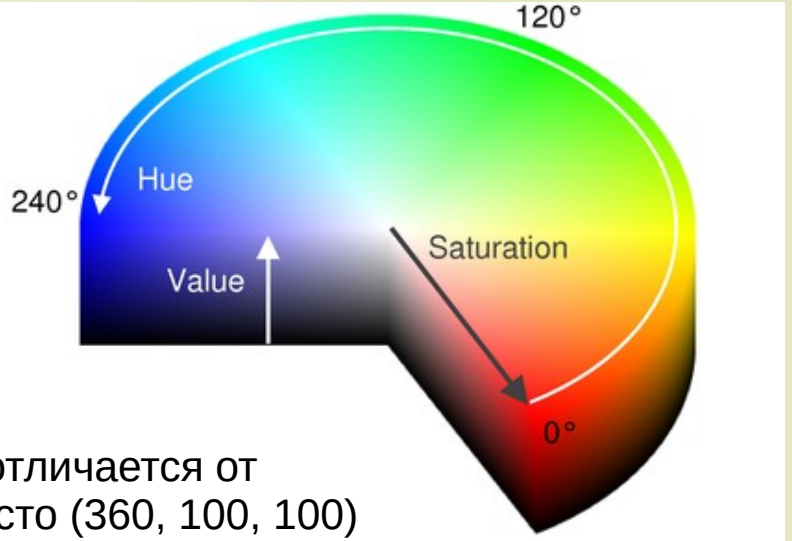








cv2.cvtColor — конверсия изображения из одного цветового пространства в другое.



В OpenCV диапазон палитры HSV отличается от общепринятого: (180, 255, 255) вместо (360, 100, 100)

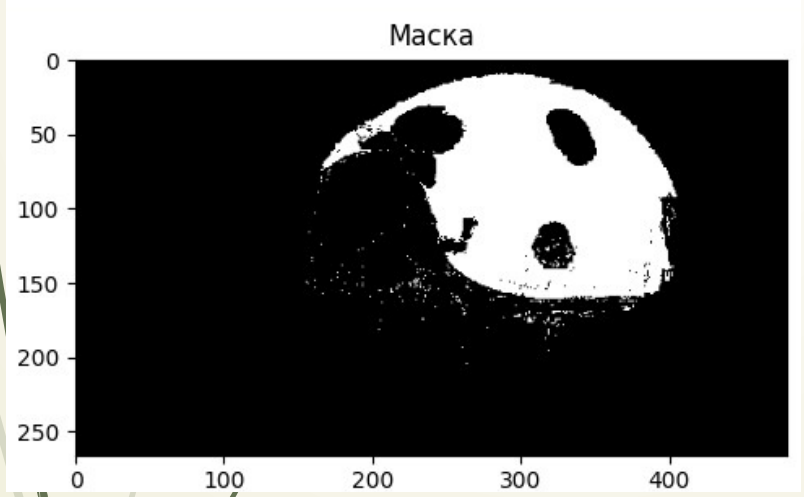
Фрагмент кода

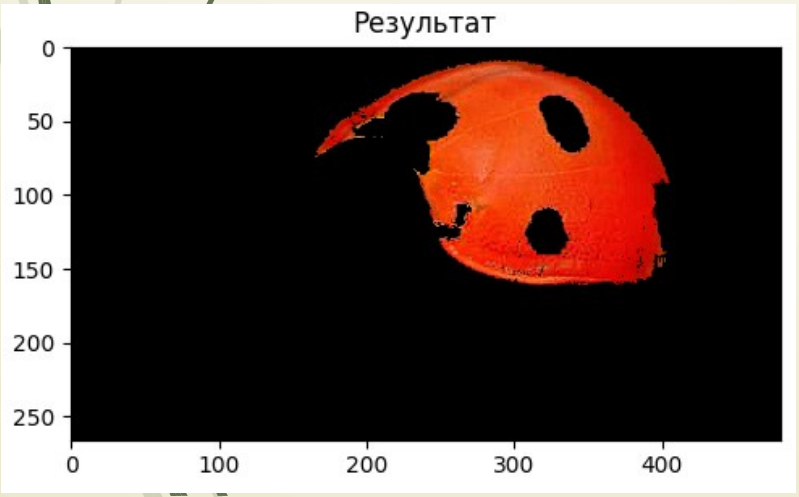
```
24
25  # Цветовые границы региона
26  low_color_hsv = (0, 130, 170)
27  high_color_hsv = (15, 255, 255)
28
29  # Демонстрация двух выбранных границ
30  low_img = np.zeros((200, 200, 3), np.uint8)
31  low_img[:] = low_color_hsv
32  low_img = cv2.cvtColor(low_img, cv2.COLOR_HSV2RGB)
33  plt.subplot(1, 2, 1)
34  plt.imshow(low_img)
35  plt.title(f'{low_color_hsv}')
36
37  high_img = np.zeros((200, 200, 3), np.uint8)
38  high_img[:] = high_color_hsv
39  high_img = cv2.cvtColor(high_img, cv2.COLOR_HSV2RGB)
40  plt.subplot(1, 2, 2)
41  plt.imshow(high_img)
42  plt.title(f'{high_color_hsv}')
43  plt.show()
44
45  # Границы для белых оттенков
46  low_white_hsv = (0, 150, 0)
47  high_white_hsv = (0, 255, 10)
48
```

Сегментация по цвету (продолжение).

cv2.inRange — маска. Входные параметры: изображение и цветовой диапазон. Возвращает двоичную маску с размером изображения, в которой значения 1 указывают на разрешение, а нулевые - на запрет вывода соответствующих пикселей.

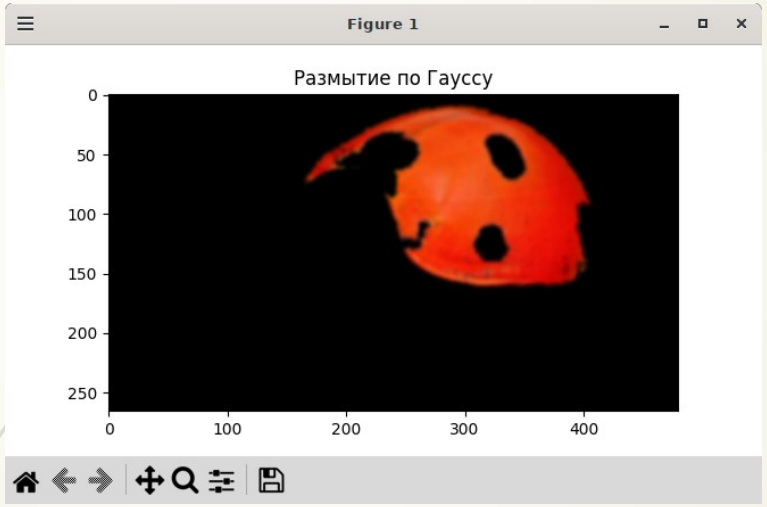
cv2.bitwise_and — здесь используется для побитового соединения двух масок.



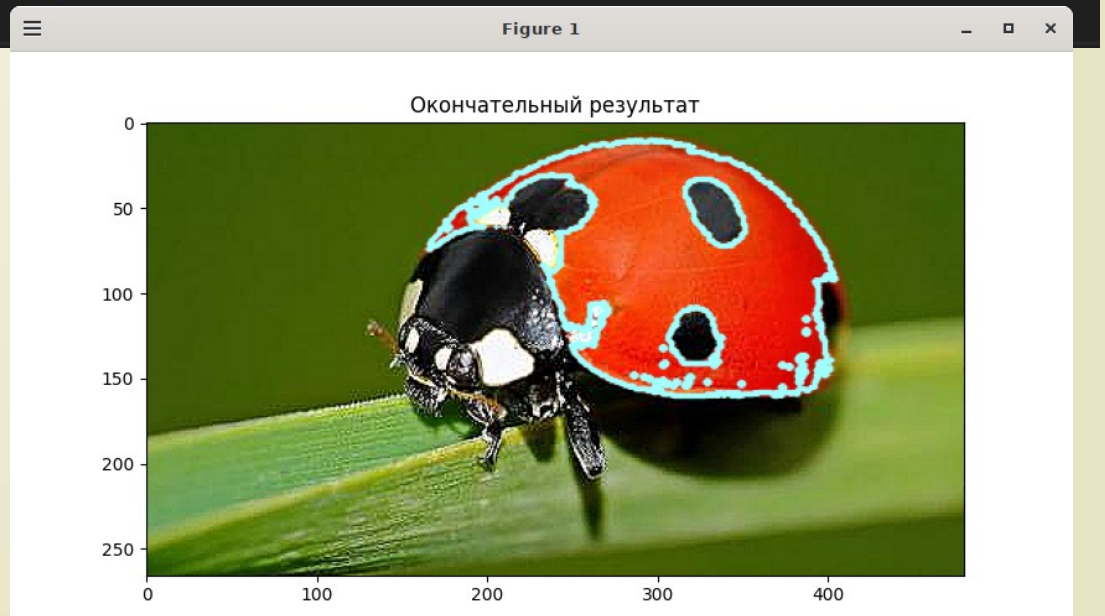



```
48
49  # Получение двоичной финальной маски
50  mask_white = cv2.inRange(hsv_ladybug, low_white_hsv, high_white_hsv)
51  mask_color = cv2.inRange(hsv_ladybug, low_color_hsv, high_color_hsv)
52  mask_full = mask_color + mask_white
53
54  # Применение маски
55  #result = cv2.imread('result.jpg')
56  result = cv2.bitwise_and(ladybug, ladybug, mask = mask_full)
57
58  # Маска и исходное изображение с маской сверху
59  plt.subplot(1, 2, 1)
60  plt.imshow(mask_full, cmap='gray')
61  plt.title('Маска')
62  plt.subplot(1, 2, 2)
63  plt.imshow(result)
64  plt.title('Результат')
65  plt.show()
66
```

Сегментация по цвету (3).




```python
# Сглаживание
blur = cv2.GaussianBlur(result, (9, 9), 0)
plt.imshow(blur)
plt.title('Размытие по Гауссу')
plt.show()

# Поиск краев, основанный на выбранном цветовом регионе
blue_color = (161, 255, 255)
contours, hierarchy = cv2.findContours(mask_color.copy(),
    cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
result = cv2.drawContours(ladybug, contours, -1,
    color=blue_color, thickness=2, lineType=cv2.LINE_AA,
    hierarchy=hierarchy, maxLevel=1)
plt.imshow(result)
plt.title('Окончательный результат')
plt.show()
```



cv2.GaussianBlur — размытие изображения по Гауссу. Вместо простого среднего в фильтре размытия здесь используется взвешенное среднее (т.е. пиксели, которые ближе к центральному, вносят больший вклад в среднее). Параметры: входное изображение, размер матрицы фильтра, дисперсия (0 — автоматически рассчитываемая).

cv2.findContours - метод для поиска контуров. Параметры: изображение, режим группировки (CV_RETR_TREE — группирует контуры в многоуровневую иерархию), метод упаковки (CV_CHAIN_APPROX_SIMPLE — склеивает все горизонтальные, вертикальные и диагональные контуры).

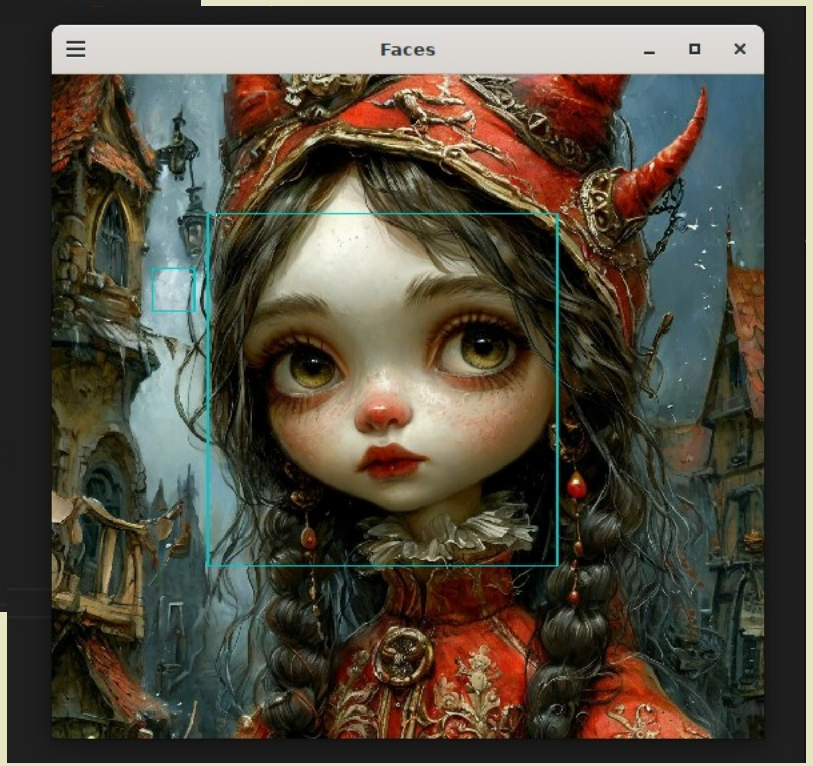
cv2.drawContours — метод для отображения контуров. Параметры: фоновое изображение, набор контуров, индекс контура (-1, чтобы отобразить все), цвет, толщина и тип линии контура, информация об иерархии, макс. слой (1 — отображается контур с дочерними).

Пример программы распознавания лиц.

OpenCV > ocv_03.py > ...

```python
# пример стандартной программы распознавания лиц на изображении
import cv2
import os
wpath = os.path.dirname(os.path.realpath(__file__))
# загружаем ИИ-модель
face_cascade = cv2.CascadeClassifier(wpath+'/haarcascade_frontalface_default.xml')
# читаем изображение
image = cv2.imread(wpath+'/images/6.jpg')
# преобразуем изображение в черно-белое
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# обнаруживаем лица
faces = face_cascade.detectMultiScale(
    gray,
    scaleFactor= 1.1,
    minNeighbors= 5,
    minSize=(30, 30)
)
print("Лиц обнаружено: " + format(len(faces)))
# оконтуриваем лица
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (200, 200, 0), 2)
# показываем изображение
cv2.namedWindow('Faces', cv2.WINDOW_NORMAL)
cv2.imshow('Faces', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

detectMultiScale — общая функция для распознавания как лиц, так и объектов. Чтобы функция искала именно лица, ей необходимо указать соответствующий каскад Хаара (обученную модель).



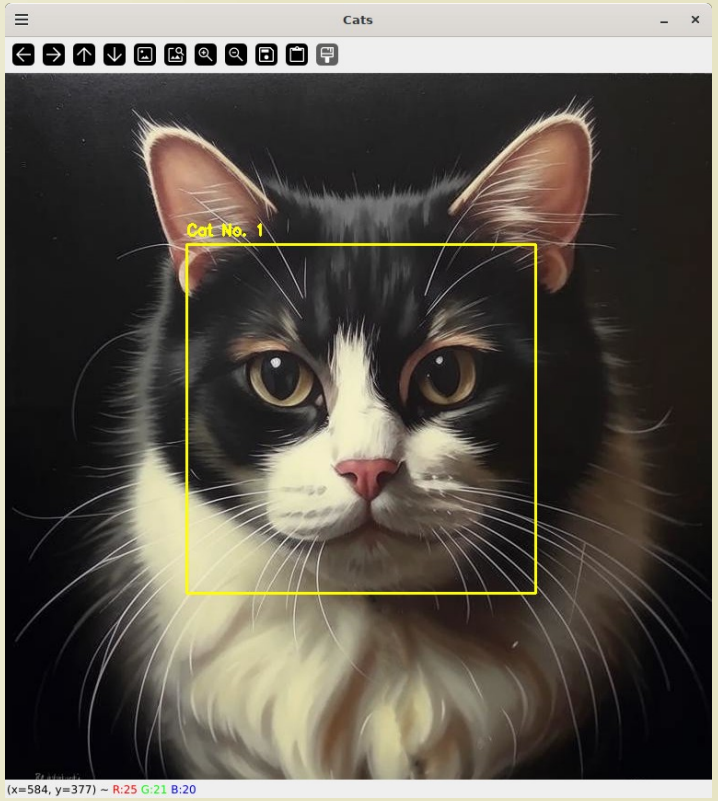
Каскадный классификатор Хаара — подход к формированию критериев распознавания образов на основе машинного обучения (обучение на "правильных" и "неправильных" изображениях).

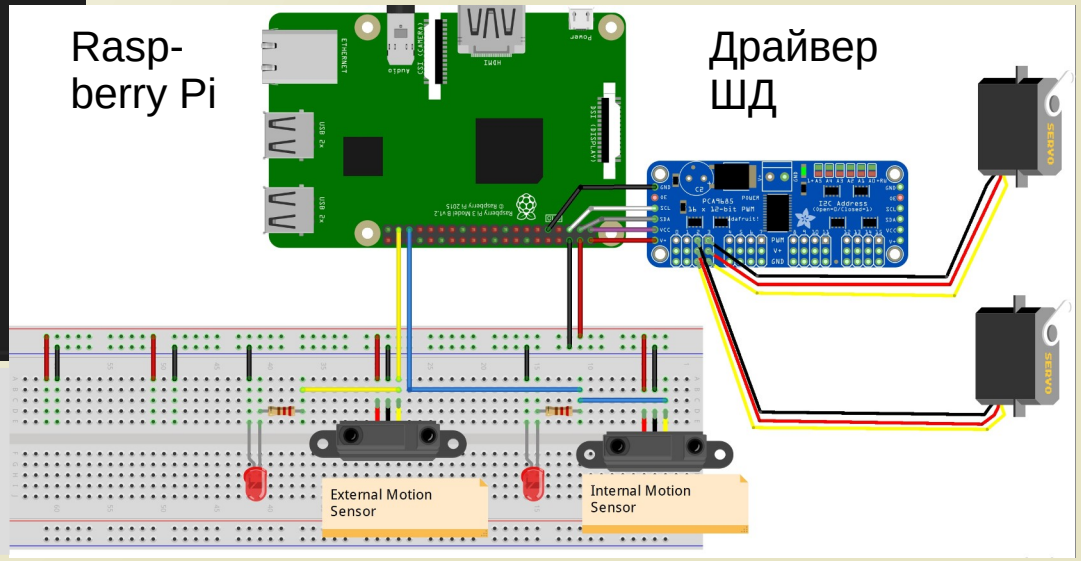
Распознавание котов.

14

Прелюдия к Pet Recognition Door Project

```python
# import the necessary packages
import cv2
import os

wpath = os.path.dirname(os.path.realpath(__file__))
# загружаем каскад Хаара для обнаружения кошек
cascade = cv2.CascadeClassifier(wpath + '/haarcascade_frontalcatface.xml')
image = cv2.imread(wpath + '/images/3.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

rects = cascade.detectMultiScale(gray, scaleFactor=1.1,
    minNeighbors=10, minSize=(70, 70))
print(rects)
# оконтуриваем мордочки кошек
for (i, (x, y, w, h)) in enumerate(rects):
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 255), 2)
    cv2.putText(image, "Cat No. {}".format(i + 1), (x, y - 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 255, 255), 2)
# выводим изображение в окончательном виде
cv2.imshow("Cats", image)
while True:
    key_press = cv2.waitKey()
    if key_press == ord('q'):
        break
```





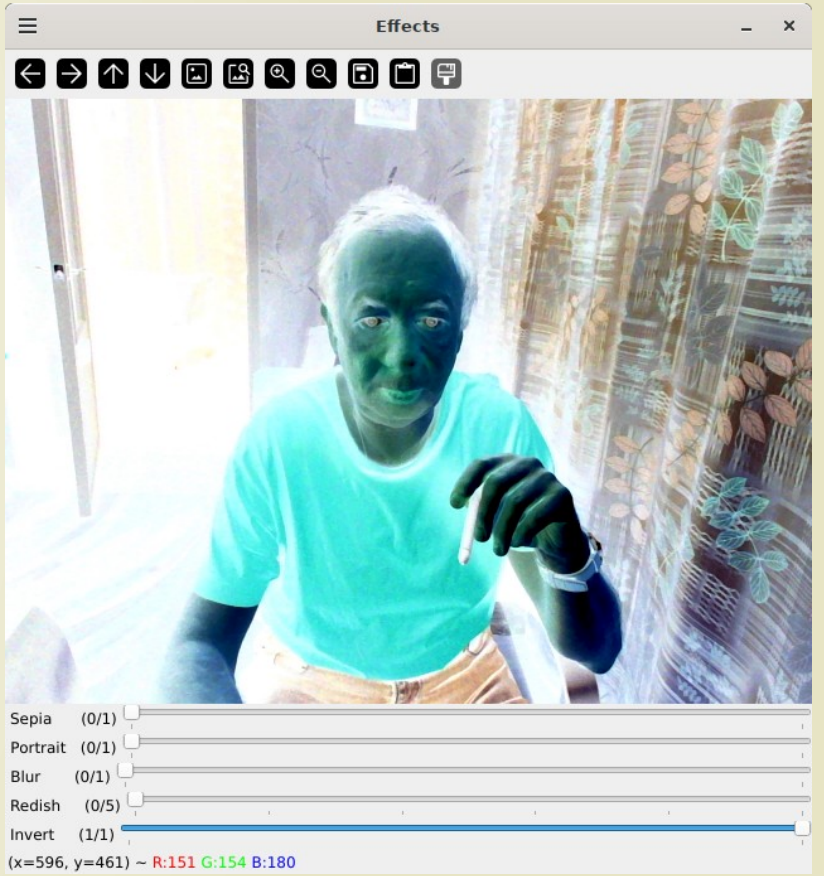
Когда питомец подходит к дверке для домашних животных, то срабатывает датчик движения и активируется веб-камера. Производится захват нескольких кадров животного. Изображения обрабатываются с использованием классификатора OpenCV и далее животному предоставляется доступ в дом.

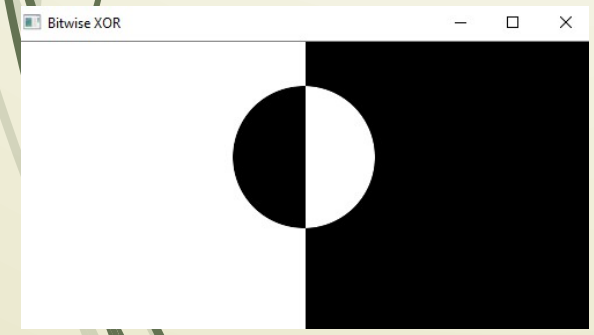
Работаем с Web-камерой.

15

Достаточно иметь одну или несколько работающих web-камер.

```python
# берём видео с web-камеры
import cv2

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FPS, 50) # Частота кадров
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 600) # Ширина кадра
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 500) # Высота кадра

while True:
    _, img = cap.read()
    cv2.imshow("Camera", img)
    if cv2.waitKey(10) == 27: # Клавиша Esc
        break

cap.release()
cv2.destroyAllWindows()
```





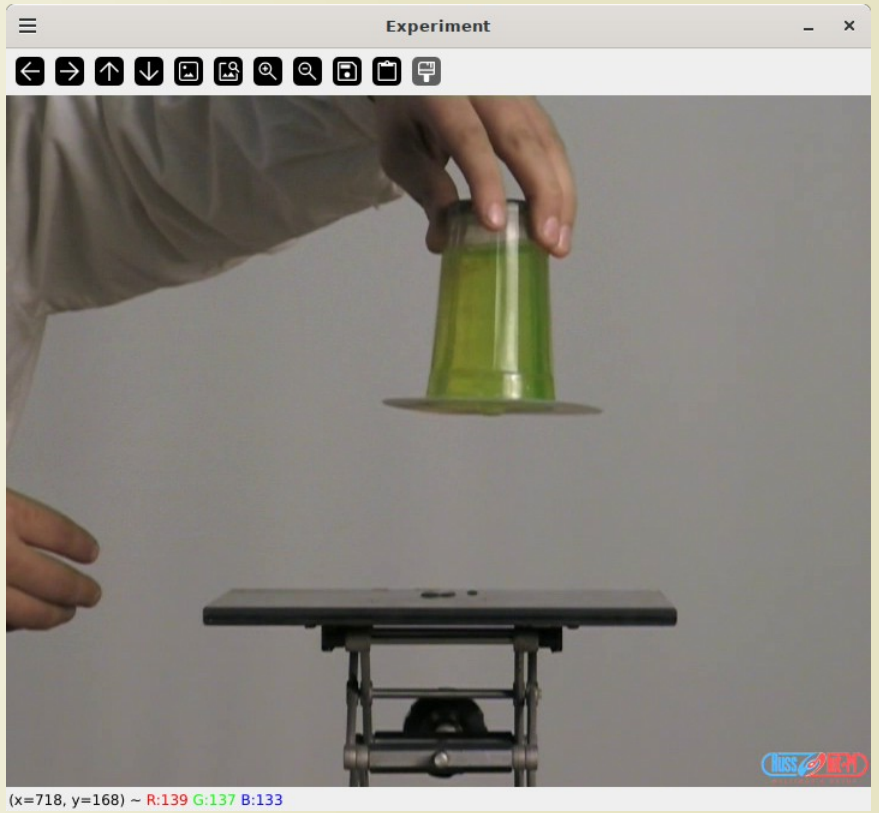
Четыре операции OpenCV позволяют проводить логические действия с пикселями любого изображения:

- cv2.bitwise_and(источник 1, источник 2)
- cv2.bitwise_or(источник 1, источник 2)
- cv2.bitwise_xor(источник 1, источник 2)
- cv2.bitwise_not(источник 1)

```python
def apply_invert(frame):
    return cv2.bitwise_not(frame)
…
while True:
    _, frame = cap.read()
    if invert == 1:
        frame = apply_invert(frame)
    cv2.imshow('Effects', frame)
    if cv2.waitKey(20) & 0xFF == ord('q'):
        break
```

Работаем с видеофайлом.


```python
import cv2
import os
wpath = os.path.dirname(os.path.realpath(__file__))
capImg = cv2.VideoCapture(wpath+'/images/007.avi')

while(capImg.isOpened()):
    # получаем кадр из видеопотока файла
    ret, frame = capImg.read()
    if frame is None:
        # если кадры закончились, прерываем работу цикла
        break
    # показываем кадр из файла в окне
    cv2.imshow("Experiment", frame)
    key_press = cv2.waitKey(30)
    if key_press == ord('q'):
        break

capImg.release()
cv2.destroyAllWindows()
```



Добавив после 14 строки:

```
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cv2.imshow('In gray scale', gray)
```

получим одновременное отображение двух окон с одним видео.
Запись видео в файл также осуществляется весьма просто:

```
    out.write(frame) # внутри цикла по кадрам
```

однако требуется выбор параметров записи видео (в т.ч. выбор кодека):

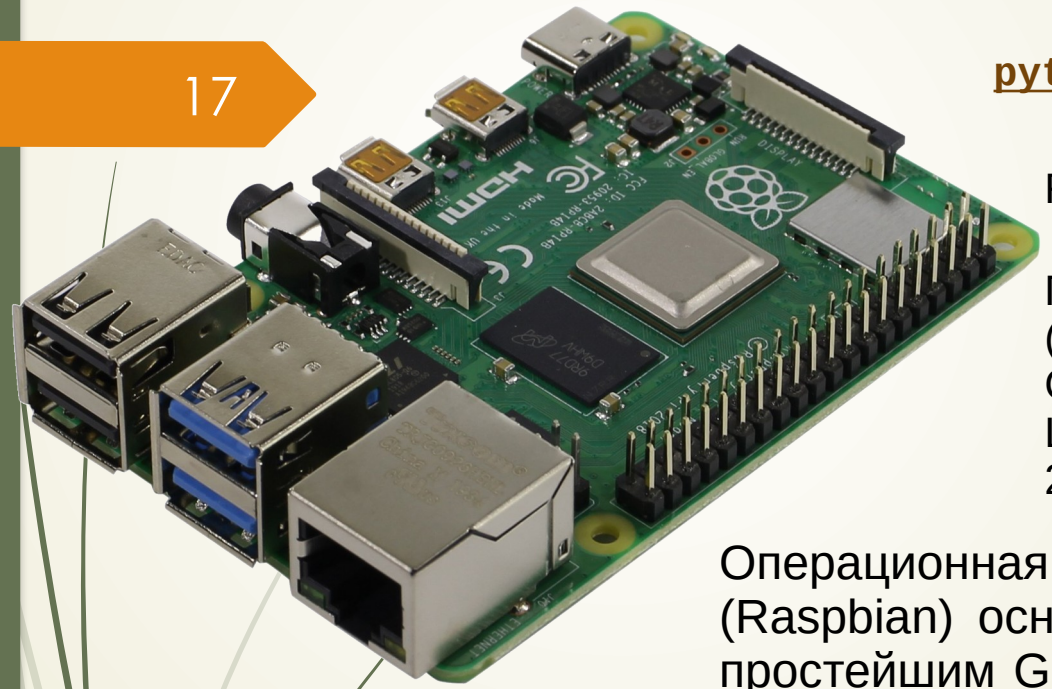
```
    codec = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(wpath + '/images/007-
out.mp4', codec, fps=30, frameSize=(width, height))
```

```python
print('Количество кадров в видеопотоке: ', frame_number)
bitrate = int(capImg.get(cv2.CAP_PROP_BITRATE))
print('Битрейт видеопотока: ', bitrate)
fps = capImg.get(cv2.CAP_PROP_FPS)
print('Частота кадров в видеопотоке: ', fps)
width = int(capImg.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(capImg.get(cv2.CAP_PROP_FRAME_HEIGHT))
print(f'Размеры видео: ширина {width}, высота {height}')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
Количество кадров в видеопотоке:  4496
Битрейт видеопотока:  1319
Частота кадров в видеопотоке:  25.0
Размеры видео: ширина 720, высота 576
```

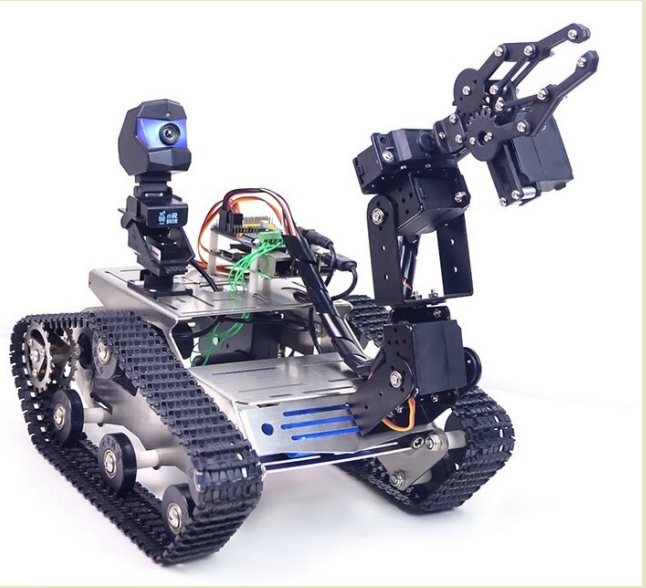
Python в микросистемах управления и контроля.



```
python3 -m http.server 5000
```

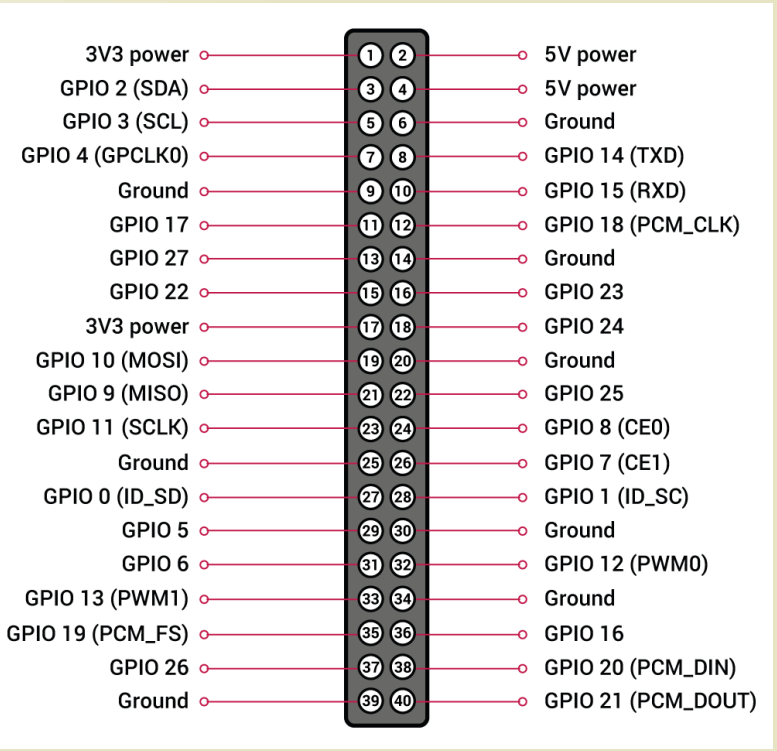
Raspberry Pi 4B:

Процессор - Cortex-A72 (ARM v8) 1,5 ГГц (64 бит, 4 ядра), память от 1 до 8 ГБ, GPIO, micro-SD, 4 USB, из них 2 - USB3, LAN 1 Гб, WiFi 802.11ac, Bluetooth 5.0
2 порта mini-HDMI, питание USB-C 5В.



Операционная система Raspberry OS (Raspbian) основана на Linux Debian с простейшим GUI, поэтому большинство возможностей этой ОС сохранено.


```
from picamera import PiCamera
from time import sleep
camera = PiCamera()
camera.start_preview()
sleep(15)
camera.stop_preview()
```



UART, I²C, SPI

Python в микросистемах управления и контроля (2).

18

Использовать Python для написания программ для контороллера Arduino (пока) нельзя, но взаимодействовать с этим контроллером можно без проблем.

Для написания, компиляции и загрузки пользовательских программ в память микроконтроллера используется собственная среда разработки Arduino IDE. Основой среды разработки является язык Processing/Wiring – очень похожий на язык C++, дополненный функциями для управления вводом/выводом на контактах.

Arduino подключается к управляющему компьютеру как USB-устройство и имеет собственный драйвер, реализующий интерфейс последовательного порта. Задачей управляющей программы на Python является только передача управляющих команд через виртуальный последовательный порт и приём результатов их выполнения.

Микроконтроллер – Atmega 328
Память flash – 32 кБ (основное хранилище для команд).
Оперативная SRAM память - 2 кБ (переменные). Энергонезависимая память (EEPROM) - 1кБ.

```
import serial, time
rate = 19200 # скорость последовательного порта
ser = serial.Serial("COM3", rate, timeout = 0)
comm = input("Введите команду (quit для выхода): ")
comm = comm + "\n"
# unicode переводим в битовый массив
ser.write(comm.encode())
time.sleep(0.1)
# битовый массив переводим обратно в строку
line = ser.readline().decode()
# убираем из выдачи последние символы \r\n
line = line[:-2]
print(line)
```

Спасибо за внимание!

Конец блока 3. Конец курса.