# БИБЛИОТЕКИ ЯЗЫКА PYTHON ДЛЯ КОМПЬЮТЕРНЫХ ВЫЧИСЛЕНИЙ И МОДЕЛИРОВАНИЯ

Библиотека OpenCV в Python


**Лобанов Алексей Владимирович**
Главный специалист отдела разработки и внедрения АИС,
Ассистент кафедры информационных компьютерных технологий РХТУ им. Д.И. Менделеева

# ПОЛЕЗНЫЕ РЕСУРСЫ

- Библиотека OpenCV (оф. сайт)
- Библиотека OpenCV (GitHub)
- Библиотека OpenCV (OpenCV Tutorials)

# ТЕМЫ

- Основные операции в **OpenCV**
- Отображение изображения
- Отображение видео
- Поиск объекта по цвету
- Настройка цветового фильтра средствами OpenCV
- Выделение контуров объектов с помощью OpenCV
- Поиск объектов

# БИБЛИОТЕКА OPENCV

pip install opencv-python

OpenCV (Open Source Computer Vision Library) — это библиотека компьютерного зрения с открытым исходным кодом, выпускается под лицензией BSD (Berkeley Software Distribution license) и, следовательно, является бесплатной как для академического, так и для коммерческого использования. OpenCV разработана под языки программирования C++, Python, Java и поддерживает операционные системы Windows, Linux, Mac OS, iOS и Android. Библиотека OpenCV предназначена для создания приложений, выполняющих большие объемы вычислений в реальном времени. Написанная на C / C++ библиотека может использовать аппаратные возможности многоядерной вычислительной платформы.

Практическое применение OpenCV — это создание различных систем, использующих в своей работе функции компьютерного зрения. Системы с компьютерным зрением особое значение имеют в робототехнике.

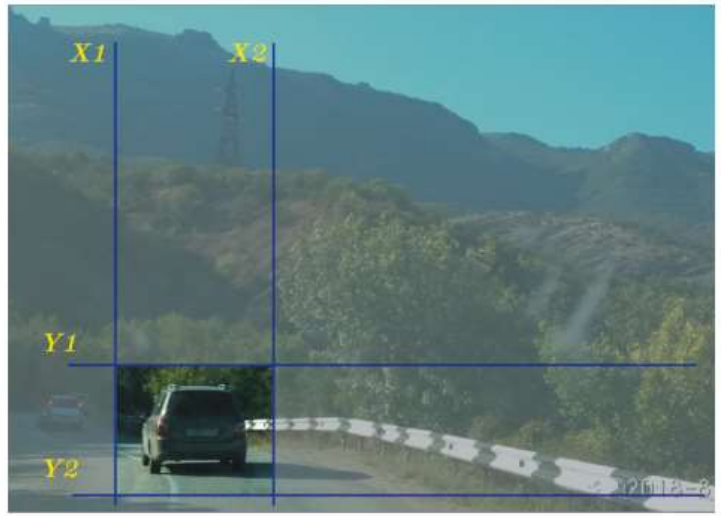
# ОТКРЫТИЕ ИЗОБРАЖЕНИЕ

```
# добавим необходимый пакет с opencv
import cv2
# присваиваем переменной image изображение из файла
# testfile.jpg
image = cv2.imread("testfile.jpg")
# выводим изображение image на экран в окне Open
# image
cv2.imshow("Open image", image)
# ожидаем нажатия любой клавиши
cv2.waitKey(0)
```

# ИЗМЕНЕНИЕ РАЗМЕРА ИЗОБРАЖЕНИЯ

```
# добавляем необходимый пакет с opencv
import cv2
# загружаем изображение
image = cv2.imread("testfile.jpg")
# задаем ширину изображения в пикселях
wide = 250
# вычисляем коэффициент, чтобы сохранить соотношение
# сторон
f = float(wide) / image.shape[1]
# формируем кортеж (x,y) с размерами изображения
new_size = (wide, int(image.shape[0] * f))
# изменяем изображение и возвращаем его в переменную
# res
res = cv2.resize(image, new_size,
                 interpolation = cv2.INTER_AREA)
# выводим изображение res на экран в окне Resize
# image
cv2.imshow("Resize image", res)
# ???
cv2.imshow("Original", image)
# ???
cv2.waitKey(0)
```

# ВЫРЕЗКА ФРАГМЕНТА ИЗОБРАЖЕНИЯ




```
import cv2
image = cv2.imread("testfile.jpg")
# вырежем фрагмент изображения, используя срезы
# y1:y2, x1:x2
crop = image[420:550, 150:280]

cv2.imshow("Original", image)
# выводим фрагмент изображения crop в окне CropImage
cv2.imshow("CropImage", crop)

cv2.waitKey(0)
```

# ПОВОРОТ ИЗОБРАЖЕНИЯ

```
import cv2
# загружаем изображение
image = cv2.imread("testfile.jpg")
# получаем размеры исходного изображения для поворота
(h, w) = image.shape[:2]
# вычислим центр изображения, относительно которого
# выполним поворот
center = (w / 2, h / 2)
# подготовим объект для поворота изображения на 180°
# относительно центра и запишем его в переменную
# prepObj
prepObj = cv2.getRotationMatrix2D(center, 180, 1.0)
# повернем исходное изображение на 180°,
# результат запишем в переменную rotated
rotated = cv2.warpAffine(image, prepObj, (w, h))
# выводим исходное изображение на экран
cv2.imshow("Original", image)
# выводим повернутое изображение на экран
cv2.imshow("RotateImage", rotated)
cv2.waitKey(0)
```

# ЗЕРКАЛЬНОЕ ОТРАЖЕНИЕ ИЗОБРАЖЕНИЯ

```
import cv2
# загружаем изображение и отображаем его
image = cv2.imread("testfile.jpg")
# отражаем изображение по вертикальной оси
flipped = cv2.flip(image,1)

cv2.imshow("Original", image)

cv2.imshow("Flip image", flipped)

cv2.waitKey(0)
```

# СОХРАНЕНИЕ ИЗОБРАЖЕНИЯ В ФАЙЛ НА ЛОКАЛЬНЫЙ ДИСК

```
import cv2

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

flip_image = cv2.flip(image,1)

cv2.imshow("Original", image)

cv2.imshow("Flip image", flip_image)
# запишем изображение на диск в формате .png
cv2.imwrite("flip.png", flip_image)
cv2.waitKey(0)
```

В результате выполнения кода в папке с файлами программы и исходного изображения появится новый файл flip.png

testfile.jpg

flip.png

# ПОИСК ОБЪЕКТА ПО ЦВЕТУ

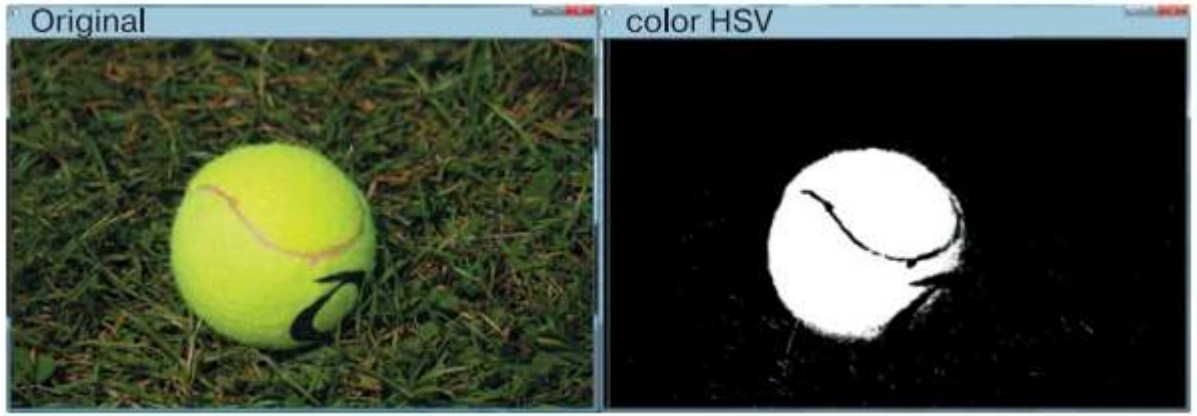
Наш объект на изображении представляет собой группу пикселей различных желтых оттенков, поэтому в диапазоне между нижней и верхней границей будут преобладать компоненты зеленого и синего цветов. Однако красная компонента тоже будет присутствовать, так как мячик все-таки не идеально желтого цвета. В программе 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.

```
import cv2
# загрузка изображения
image = cv2.imread('YellowBall.jpg')

cv2.imshow("Original", image)
# задаем границы диапазона:
# нижнюю
low_color = (0,0,150)
# 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)
```

# ПОИСК ОБЪЕКТА В ЦВЕТОВОМ ПРОСТРАНСТВЕ HSV

```
import cv2
# загрузка изображения
image = cv2.imread('YellowBall.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 = (25,100,175)
# верхняя граница — это яркий насыщенный цвет
color_high = (35,255,255)
# наложение цветовой маски на HSV-изображение,
# результат присваиваем переменной only_object
only_object = cv2.inRange(hsv_img, color_low,
                          color_high)
# вывод отфильтрованного изображения на экран
cv2.imshow('color_hsv', only_object)
cv2.waitKey(0)
```



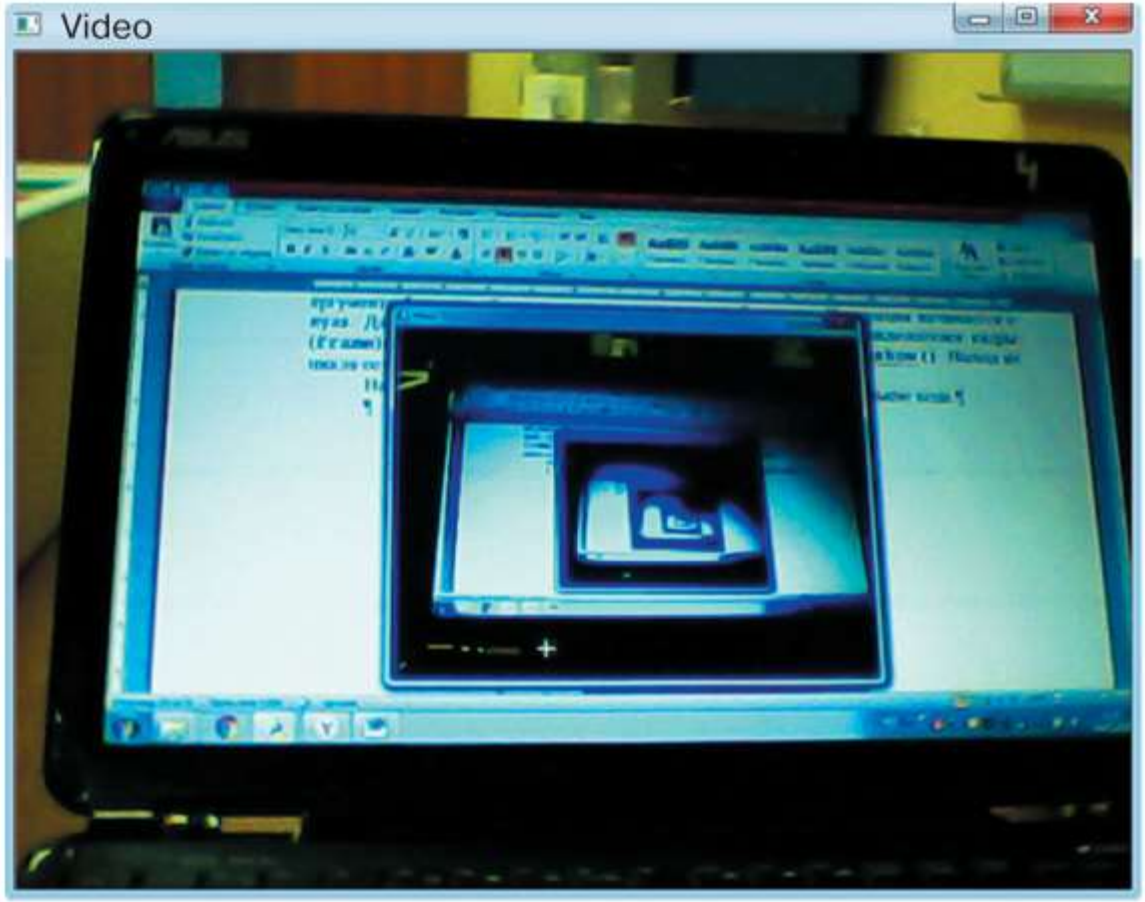

Результат поиска с использованием цветовой HSV-маски

# ОПРЕДЕЛЕНИЕ КООРДИНАТ НАЙДЕННОГО ОБЪЕКТА

```
import cv2
# загрузка изображения
image = cv2.imread('YellowBall.jpg')
# 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 = (25,60,160)
# верхняя граница - это яркий насыщенный цвет
color_high = (60,255,255)
# наложение цветовой маски на HSV-изображение,
# результат присваиваем переменной only_object
only_object = cv2.inRange(hsv_img, color_low,
                          color_high)
# вычисляем моменты отфильтрованного HSV-изображения
moments = cv2.moments(only_object, 1)
# вычисляем сумму x-координат всех точек пятна
x_moment = moments['m01']
# вычисляем сумму y-координат всех точек пятна
y_moment = moments['m10']
# вычисляем общее число всех точек пятна
area = moments['m00']
# вычисляем среднее значение координаты x объекта
x = int(x_moment / area)
# вычисляем среднее значение координаты y объекта
y = int(y_moment / area)
# выводим надпись "Yellow ball" на изображение
cv2.putText(image, "Yellow ball", (x,y),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)
# выводим координаты объекта
cv2.putText(image, "%d, %d" % (x,y), (x,y+30),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)
# выводим картинку в окне found
cv2.imshow('found', image)
cv2.waitKey(0)
```

# ОТОБРАЖЕНИЕ ВИДЕО В OPENCV-PYTHON

```
import cv2
# связываем видеопоток камеры с переменной capImg
capImg = cv2.VideoCapture(0)
# запускаем бесконечный цикл, чтобы следить
# в реальном времени
while(True):
# получаем кадр из видеопотока,
# кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
# показываем кадр в окне 'Video'
    cv2.imshow('Video', frame)
#   организуем выход из цикла по нажатию клавиши,
#   ждем 30 миллисекунд нажатие, записываем код
#   нажатой клавиши
    key_press = cv2.waitKey(30)
#   если код нажатой клавиши совпадает с кодом
#   «q»(quit - выход),
    if key_press == ord('q'):
#       то прервать цикл while
        break
# освобождаем память от переменной capImg
capImg.release()
# закрываем все окна opencv
cv2.destroyAllWindows()
```




Захват видео с web-камеры с помощью OpenCV

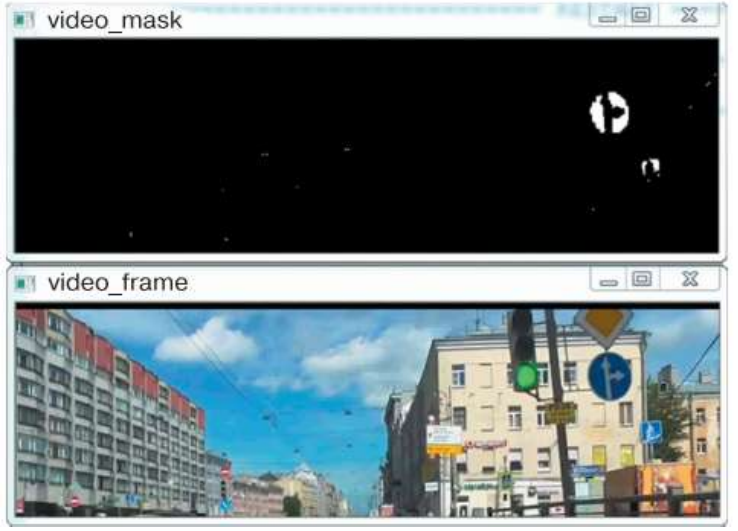
# ОТОБРАЖЕНИЕ ВИДЕОДАННЫХ ИЗ ФАЙЛА

```
import cv2
# связываем файл video.avi с переменной capImg
capImg = cv2.VideoCapture('video.avi')
# открываем в цикле файл
while(capImg.isOpened()):
# получаем кадр из видеопотока файла
# кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
#   если кадры закончились
    if frame is None:
        break # прерываем работу цикла
#  показываем кадр из файла в окне video_file
    cv2.imshow("video_file", frame)
#   организуем выход из цикла по нажатию клавиши,
#   ждем 30 миллисекунд нажатия, записываем код
#   нажатой клавиши
    key_press = cv2.waitKey(30)
#   если код нажатой клавиши совпадает с кодом «q»,
    if key_press == ord('q'):
        break # то прервать цикл while
# освобождаем память от переменной capImg
capImg.release()
# закрываем все окна opencv
cv2.destroyAllWindows()
```


Воспроизведение файла video.avi с помощью OpenCV (Источник: кадр из ролика

# ПОИСК ЦВЕТНЫХ ОБЪЕКТОВ НА ВИДЕО С ПОМОЩЬЮ OPENCV

```
# подключение библиотеки numpy
# под именем np
import numpy as np
# подключение библиотеки opencv
import cv2
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')
# открываем файл с видео
while(capImg.isOpened()):
#   получаем кадр из видеопотока файла,
#   кадры по очереди считываются в переменную frame
    ret, frame = cap.read()
#   если кадры закончились, то прерываем цикл
    if frame is None:
        break
#   переводим кадр в цветовое HSV-пространство
    frame_hsv = cv2.cvtColor(frame,
                             cv2.COLOR_BGR2HSV)
#   делаем вырезку области кадра, где ожидается
#   объект
    crop_frame = frame[60:270, 30:750]
    crop_frame_hsv = frame_hsv[60:270, 30:750]
#   задаем нижнюю и верхнюю границы цветового фильтра
#   с помощью массивов numpy
#   цвет 0...180,
#   насыщенность 0 - блеклый, 255 - насыщенный
#   яркость 0 - темный, 255 - светлый
    low_Blue = np.array([105, 150, 0],
                        dtype = "uint8")
    high_Blue = np.array([135, 255, 210],
                         dtype = "uint8")
#   применяем маску по цвету к фрагменту кадра
    blue_mask = cv2.inRange(crop_frame_hsv, low_Blue,
                            high_Blue)
#   показываем области кадров с объектом
#   в окне video_mask результат наложения маски,
    cv2.imshow("video_mask", blue_mask)
#   в окне video_frame вырезку из кадра
    cv2.imshow("video_frame", crop_frame)
#   организуем выход из цикла по нажатию клавиши
#   ждем 30 миллисекунд нажатия, записываем код
#   нажатой клавиши
    key_press = cv2.waitKey(30)
#   если код нажатой клавиши совпадает с кодом «q»,
    if key_press == ord('q'):
        break
# освобождаем память от переменной capImg
capImg.release()
# закрываем все окна opencv
cv2.destroyAllWindows()
```



# СУММИРОВАНИЕ ЦВЕТОВЫХ МАСОК

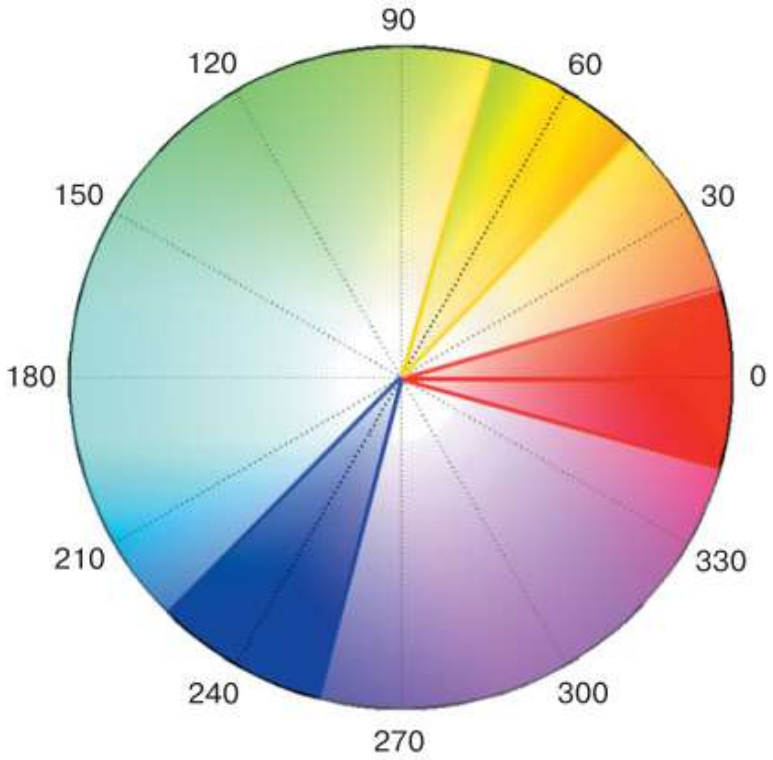
```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')
# открываем файл с видео
while(capImg.isOpened()):
#   получаем кадр из видеопотока файла,
#   кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
#   если кадры закончились, то прерываем цикл
    if frame is None:
        break
#   переводим кадр в цветовое пространство HSV
    frame_hsv = cv2.cvtColor(frame,
                             cv2.COLOR_BGR2HSV)
#   делаем вырезку области кадра, где ожидается
#   группа объектов
    crop_frame = frame[60:270, 30:750]
    crop_frame_hsv = frame_hsv[60:270, 30:750]
```

```
#   задаем нижние и верхние границы цветовых фильтров
#   с помощью массивов numpy
#   цвет 0...180
#   насыщенность 0 - блеклый, 255 - насыщенный
#   яркость 0 - темный, 255 - светлый
#   фильтр для синего цвета - подобрали ранее
    low_Blue = np.array ([105, 150, 0],
                         dtype = "uint8")
    high_Blue = np.array([135, 255, 210],
                         dtype = "uint8")
#   фильтр для желтого цвета
    low_Yel = np.array([10, 150, 100],
                       dtype = "uint8")
    high_Yel = np.array([50, 255, 255],
                        dtype = "uint8")
# фильтр для красного цвета
# красный цвет представляет собой две области,
# красный в сторону оранжевой области
    low_Red_O = np.array([0, 85, 110],
                         dtype = "uint8")
    high_Red_O = np.array([5, 165, 155],
                          dtype = "uint8")
# красный в сторону фиолетовой области
    low_Red_V = np.array([165, 55, 40],
                         dtype = "uint8")
```

# СУММИРОВАНИЕ ЦВЕТОВЫХ МАСОК

```
    high_Red_V = np.array([180, 105, 120],
                          dtype = "uint8")
#   применяем маску по каждому цвету к фрагменту
#   кадра для выделения синего цвета
    blue_mask = cv2.inRange(crop_frame_hsv, low_Blue,
                            high_Blue)
#   для выделения желтого цвета
    yel_mask = cv2.inRange(crop_frame_hsv, low_Yel,
                           high_Yel)
#   для выделения красного накладываются две маски
    red1_mask = cv2.inRange(crop_frame_hsv,
                            low_Red_O, high_Red_O)
    red2_mask = cv2.inRange(crop_frame_hsv,
                            low_Red_V, high_Red_V)
#   вычисляем полную маску
#   полная маска представляет собой сумму всех масок
    full_mask = red1_mask + red2_mask + blue_mask +
                yel_mask
#   показываем области кадров с искомой группой
#   объектов: в окне video_mask – результат наложения
#   полной маски,
    cv2.imshow("video_mask", full_mask)
#   в окне video_frame – вырезку из кадра
    cv2.imshow("video_frame", crop_frame)
#   организуем выход из цикла по нажатию клавиши,
#   ждем 30 миллисекунд нажатия, записываем код
#   нажатой клавиши
    key_press = cv2.waitKey(30)
#   если код нажатой клавиши совпадает с кодом «q»,
    if key_press == ord('q'):
        break
# освобождаем память от переменной capImg
capImg.release()
# закрываем все окна opencv
cv2.destroyAllWindows()
```

# СУММИРОВАНИЕ ЦВЕТОВЫХ МАСОК




Цветовая диаграмма HSV: границы областей составного цветового фильтра

# НАСТРОЙКА ЦВЕТОВОГО ФИЛЬТРА СРЕДСТВАМИ OPENCV

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2

# создаем «пустую» функцию
def nothing(*arg):
    pass

# создаем окно настроек с именем Set
cv2.namedWindow('Set')
# создаем 6 ползунков для настройки цветового фильтра
# для компонент H,S,V соответственно
cv2.createTrackbar('h1', 'Set', 0, 180, nothing)
cv2.createTrackbar('s1', 'Set', 0, 255, nothing)
cv2.createTrackbar('v1', 'Set', 0, 255, nothing)
cv2.createTrackbar('h2', 'Set', 180, 180, nothing)
cv2.createTrackbar('s2', 'Set', 255, 255, nothing)
cv2.createTrackbar('v2', 'Set', 255, 255, nothing)
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')
# открываем файл с видео
```

```
while(capImg.isOpened()):
# получаем кадр из видеопотока файла,
# кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
#   если кадры закончились, то прерываем цикл
    if frame is None:
        break
#   переводим кадр в цветовое пространство HSV
    frame_hsv = cv2.cvtColor(frame,
                             cv2.COLOR_BGR2HSV )
#   вводим логическую переменную для выхода из
#   программы
    cl_pr = False
#   запускаем цикл со стоп-кадром по настройке
#   фильтра
    while True:
#       считываем значения бегунков для H,S,V
#       соответственно
        h1 = cv2.getTrackbarPos('h1', 'Set')
        s1 = cv2.getTrackbarPos('s1', 'Set')
        v1 = cv2.getTrackbarPos('v1', 'Set')
        h2 = cv2.getTrackbarPos('h2', 'Set')
        s2 = cv2.getTrackbarPos('s2', 'Set')
        v2 = cv2.getTrackbarPos('v2', 'Set')
```

# НАСТРОЙКА ЦВЕТОВОГО ФИЛЬТРА СРЕДСТВАМИ OPENCV

```
#       формируем нижнюю и верхнюю границы цветового
#       фильтра
        h_min = np.array((h1, s1, v1), np.uint8)
        h_max = np.array((h2, s2, v2), np.uint8)
#       накладываем фильтр на кадр в цветовом
#       пространстве HSV,
#       результат записываем в RealTimeMask
        RealTimeMask = cv2.inRange(frame_hsv, h_min,
                                   h_max)
#       показываем исходный кадр в окне Original
        cv2.imshow('Original', frame)
#       показываем фильтрованный кадр в окне result
        cv2.imshow('result', RealTimeMask)
#       организуем управление программой по нажатию
#       клавиши, ждем 30 миллисекунд нажатия,
#       записываем код нажатой клавиши
        key_press = cv2.waitKey(30)
#       если код клавиши равен коду символа «n»
        if key_press == ord('n'):
#           прерываем цикл показа текущего кадра
#           для перехода к следующему
            break
#       если код клавиши равен коду символа «q»
        elif key_press == ord('q'):
#           переменной выхода из программы присваиваем
#           True
            cl_pr = True
#           прерываем цикл показа текущего кадра
            break
#   если переменная выхода из программы True
    if cl_pr:
#       прерываем цикл считывания кадров
        break
capImg.release()
cv2.destroyAllWindows()
```

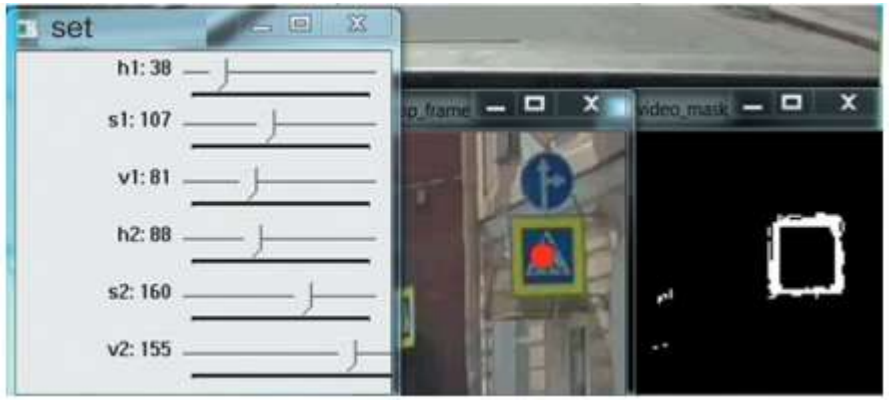
# ПОИСК ЦВЕТНЫХ ОБЪЕКТОВ НА ВИДЕО В РЕАЛЬНОМ ВРЕМЕНИ

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2

# создаем «пустую» функцию маски
def nothing(*arg):
    pass

# создаем окно настроек с именем Set
cv2.namedWindow('Set')
# создаем 6 ползунков для настройки цветового фильтра
# для компонент H,S,V соответственно
cv2.createTrackbar('h1', 'Set', 0, 180, nothing)
cv2.createTrackbar('s1', 'Set', 0, 255, nothing)
cv2.createTrackbar('v1', 'Set', 0, 255, nothing)
cv2.createTrackbar('h2', 'Set', 180, 180, nothing)
cv2.createTrackbar('s2', 'Set', 255, 255, nothing)
cv2.createTrackbar('v2', 'Set', 255, 255, nothing)
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')
# открываем файл с видео
while(capImg.isOpened()):
# получаем кадр из видеопотока файла
# кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
#   если кадры закончились, то прерываем цикл
    if frame is None:
        break
#   вырезаем область кадра, где ожидаются знаки,
#   результат записываем в cr_frame
    cr_frame = frame[150:260, 485:600]
#   задаем коэффициент увеличения k
    k = 2
#   рассчитываем новые размеры для вырезанного
#   фрагмента
    x = cr_frame.shape[1] * k
    y = cr_frame.shape[0] * k
    dim = (x, y)
#   увеличиваем размер фрагмента кадра,

#   результат записываем в RSframe
    RSframe = cv2.resize(cr_frame, dim,
                         interpolation = cv2.INTER_ARI
#   переводим трансформированный фрагмент в HSV,
#   результат записываем в RSframeHSV
    RSframeHSV = cv2.cvtColor(RSframe,
                              cv2.COLOR_BGR2HSV)
#   вводим логическую переменную для выхода из
#   программы
    cl_pr = False
#   запускаем цикл со стоп-кадром по настройке
#   фильтра
```

# ПОИСК ЦВЕТНЫХ ОБЪЕКТОВ НА ВИДЕО В РЕАЛЬНОМ ВРЕМЕНИ

```
    while True:
#       считываем значения ползунков для H,S,V
#       соответственно
        h1 = cv2.getTrackbarPos('h1', 'Set')
        s1 = cv2.getTrackbarPos('s1', 'Set')
        v1 = cv2.getTrackbarPos('v1', 'Set')
        h2 = cv2.getTrackbarPos('h2', 'Set')
        s2 = cv2.getTrackbarPos('s2', 'Set')
        v2 = cv2.getTrackbarPos('v2', 'Set')
#       формируем нижнюю и верхнюю границы цветового
#       фильтра
        h_min = np.array((h1, s1, v1), np.uint8)
        h_max = np.array((h2, s2, v2), np.uint8)
#       накладываем фильтр на кадр в цветовом
#       пространстве HSV, результат записываем
#       в RealTimeMask
        RealTimeMask = cv2.inRange(RSframeHSV, h_min,
                                   h_max)
#       вычисляем моменты отфильтрованного
#       HSV-изображения
        moments = cv2.moments(RealTimeMask, 1)
#       вычисляем сумму X, Y координат всех точек
#       пятна-объекта
        dM01 = moments['m01']
        dM10 = moments['m10']
        Area = moments['m00']
#       вычисляем средние координаты центра
#       пятна-объекта
        x = int(dM10 / Area)
        y = int(dM01 / Area)
#       ставим красную метку по полученным
#       координатам
        cv2.circle(RSframe, (x, y), 10, (0,0,255),
                   -1)
#       показываем кадры в opencv-окне
        cv2.imshow("video_frame", frame)
        cv2.imshow("Crop_frame", RSframe)
        cv2.imshow("video_mask", RealTimeMask)
#       организуем управление программой  по нажатию
#       клавиши, ждем 30 миллисекунд нажатия,
#       записываем код нажатой клавиши
        key_press = cv2.waitKey(30)
#       если код клавиши равен коду символа «n»
        if key_press == ord('n'):
#       прерываем цикл показа текущего кадра
#       для перехода к следующему
            break
#       если код клавиши равен коду символа «q»
        elif key_press == ord('q'):
#       переменной выхода из программы присваиваем
#       True
            cl_pr = True
#       прерываем цикл показа текущего кадра
            break
#   если переменная выхода из программы True
    if cl_pr:
#       прерываем цикл считывания кадров
        break
capImg.release()
cv2.destroyAllWindows()
```

# ПОИСК ЦВЕТНЫХ ОБЪЕКТОВ НА ВИДЕО В РЕАЛЬНОМ ВРЕМЕНИ


Результат поиска дорожных знаков на вырезанной области кадра с автомобильного видеорегистратора

# ВЫДЕЛЕНИЕ КОНТУРОВ ОБЪЕКТОВ С ПОМОЩЬЮ OPENCV-PYTHON

Для поиска контуров в библиотеке OpenCV имеется функция **cv2.findContours()**, которая возвращает два параметра: **contours** и **hierarchy**[1] (для версии OpenCV 4.x.x). В общем виде функция имеет следующее представление:

```
cv2.findContours (image, mode, method[,
                  contours[, hierarchy[, offset]]])
```

где аргументы означают следующее:

- **image** — переменная, в которую записано графическое изображение, специально подготовленное для анализа. В данном случае для работы используется 8-битное монохромное изображение.
- **mode** — режим группировки найденных контуров для управления их отображением. Таких режимов четыре:

1) **cv2.RETR_LIST** — режим выдает все найденные контуры без группировки;
2) **cv2.RETR_EXTERNAL** — режим выдает только крайние внешние контуры, а контуры, найденные внутри объекта, не отображаются;
3) **cv2.RETR_CCOMP** — режим группировки контуров в двухуровневую иерархию:
   — на верхнем уровне — внешние контуры объекта,
   — на втором уровне — контуры имеющихся отверстий,
   — все остальные контуры группируются на верхнем уровне;
4) **cv2.RETR_TREE** — режим группировки контуров в многоуровневую иерархию.

- **method** — способ хранения найденных контуров; выбирается один из трех методов упаковки контуров:

1) **cv2.CHAIN_APPROX_NONE** — упаковка не применяется, хранит абсолютно все точки контура, при этом любые две последующие точки контура являются соседями по горизонтали, вертикали или диагонали со смещением по координатам не больше 1;
2) **cv2.CHAIN_APPROX_SIMPLE** — метод склейки всех горизонтальных, вертикальных и диагональных сегментов контуров с сохранением только их конечных точек, например для хранения вертикального прямоугольного контура будет использовано всего четыре точки;
3) **cv2.CHAIN_APPROX_TC89_L1**, **cv2.CHAIN_APPROX_TC89_KCOS** — для хранения контуров применяется метод упаковки Teh-Chin[1].

# ВЫДЕЛЕНИЕ КОНТУРОВ ОБЪЕКТОВ С ПОМОЩЬЮ OPENCV-PYTHON

- **contours** — (необязательный параметр) список всех обнаруженных контуров, представленных в виде векторов.
- **hierarchy** — (необязательный параметр) информация о топологии контуров: каждый элемент иерархии представляет собой сборку из четырех индексов, соответствующую **contours [i]**:
  1) **hierarchy [i] [0]** — индекс следующего контура на текущем слое;
  2) **hierarchy [i] [1]** — индекс предыдущего контура на текущем слое;
  3) **hierarchy [i] [2]** — индекс первого контура на вложенном слое;
  4) **hierarchy [i] [3]** — индекс родительского контура.

- **offset** — (необязательный параметр) величина дополнительного смещения, на которое смещается каждая точка контура. Этот параметр полезно применять, если контуры извлекаются из ROI-изображения, а затем должны быть проанализированы во всем контексте изображения.

После операции поиска контуров все найденные контуры необходимо отобразить. Визуализация найденных в кадре контуров выполняется с помощью функции **cv2.drawContours()**. Эта функция в большей степени необходима пользователю, поскольку позволяет лучше понять, как выглядят найденные контуры. В общем виде функция **cv2.drawContours()** записывается следующим образом:

```
cv2.drawContours(image, contours,
                 contoursIdx, color[,thickness
                 [, lineType[, hierarchy[,
                 maxLevel[, offset]]]]])
```

где аргументы означают следующее:

- **image** — переменная с изображением, поверх которого будут нарисованы контуры.
- **contours** — переменная, в которую записаны контуры, найденные функцией **cv2.findContours()**.
- **contoursIdx** — индекс контура, который следует отобразить; если указанный индекс равен «–1», то отображаются все контуры.
- **color** — цвет линии контура, задается в виде тройки целочисленных значений (B, G, R), где каждая составляющая принимает значение от 0 до 255.

- **thickness** — (необязательный параметр) толщина линии контура, задается целочисленным параметром; если задается отрицательное значение, например **thickness=CV_FILLED**, то будут рисоваться только внутренние (вложенные) контуры.

# ВЫДЕЛЕНИЕ КОНТУРОВ ОБЪЕКТОВ С ПОМОЩЬЮ OPENCV-PYTHON

- **lineType** — (необязательный параметр) тип контурной линии; в документации OpenCV указываются три значения[1]:
  1) **8** (или не указан) — для рисования линии применяется 8-связный алгоритм Бресенхэма[2];
  2) **4** — для рисования линии применяется 4-связный алгоритм Бресенхэма[3];
  3) **cv2.LINE_AA** — рисуется сглаженная линия с применением гауссовой фильтрации.

- **hierarchy** — (необязательный параметр) информация об иерархии контуров, используется в случае, когда требуется нарисовать только некоторые контуры;

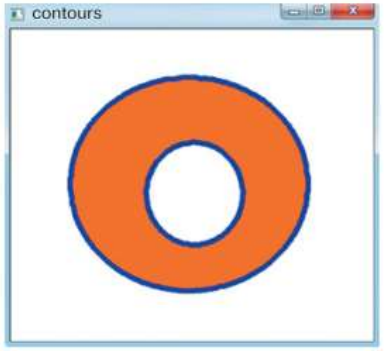
- **maxLevel** — (необязательный параметр) индекс слоя, который следует отображать:
  1) **0** — будет отображен только выбранный контур;
  2) **1** — будут отображены выбранный контур и все его дочерние контуры;
  3) **2** — будут отображены выбранный контур, все его дочерние контуры, дочерние контуры дочерних контуров и т. д.

- **offset** — (необязательный параметр) величина смещения точек контура, задается парой целочисленных значений (*dx*, *dy*).

Пример графического изображения для поиска и отображения контуров

# ВЫДЕЛЕНИЕ КОНТУРОВ ОБЪЕКТОВ С ПОМОЩЬЮ OPENCV-PYTHON

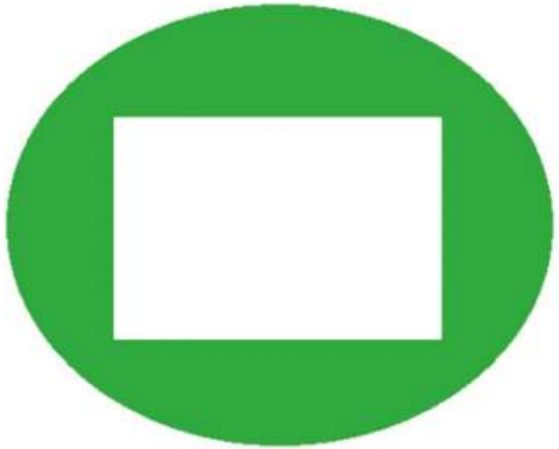
```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# задаем путь к файлу с картинкой
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'image.jpg'
# считываем данные графического файла в переменную
# image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV
# результат записываем в переменную hsv_img
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
```




Результат работы программы поиска и отображения контуров в тестовом изображении

```
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((2, 28, 65), np.uint8)
hsv_max = np.array((26, 238, 255), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                        cv2.RETR_LIST,
                                    cv2.CHAIN_APPROX_SIMPLE)
# отображаем все контуры поверх исходного изображения,
# цвет синий, толщина линии 3, сглаженная
cv2.drawContours( image, contours, -1, (255,0,0),
                  3, cv2.LINE_AA, hierarchy, 2)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# ждем нажатия любой клавиши и закрываем все окна
cv2.waitKey()
cv2.destroyAllWindows()
```

# ВЫДЕЛЕНИЕ ПРЯМОУГОЛЬНЫХ КОНТУРОВ



Пример тестового изображения для поиска прямоугольных контуров

Поскольку за основу мы берем предыдущую программу, алгоритм будет иметь некоторое сходство, но содержать несколько иную последовательность действий:

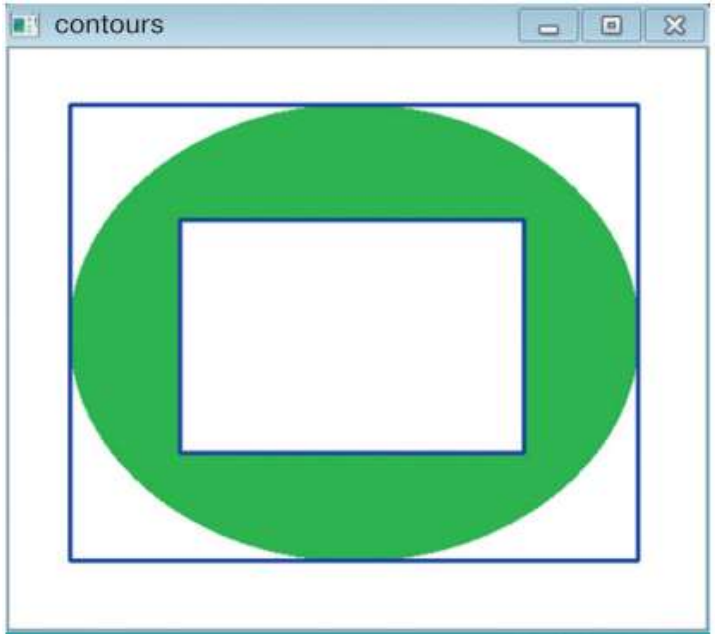
1. Получить доступ к данным графического файла.
2. Перевести изображение в цветовое HSV-пространство.
3. Выделить объект с помощью цветового HSV-фильтра.
4. Выполнить поиск всех контуров.
5. Начать цикл по всем найденным контурам.
6. Найти прямоугольник, описывающий текущий контур.
7. Отобразить полученный нами прямоугольник на исходном изображении.
8. Закончить цикл.
9. Вывести на экран окно с исходным изображением

# ВЫДЕЛЕНИЕ ПРЯМОУГОЛЬНЫХ КОНТУРОВ

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# задаем путь к файлу с картинкой,
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'rec_img.jpg'
# считываем данные графического файла в переменную
# image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV,
# результат записываем в переменную hsv_img


hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((50, 150, 155), np.uint8)
hsv_max = np.array((70, 255, 200), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                        cv2.RETR_LIST,
                                    cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
#   ищем прямоугольник, результат записываем в rect
    rect = cv2.minAreaRect(icontour)
#   поиск вершин прямоугольника, результат записываем
#   в box
    box = cv2.boxPoints(rect)
#   округление координат вершин, результат записываем
#   в box
    box = np.int0(box)
#   рисуем прямоугольник поверх исходного изображения
#   цвет синий, толщина линии 3, поскольку рисуем
#   единственный объект [box], остальные параметры
#   опускаем
    cv2.drawContours( image, [box], -1, (255,0,0),3)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# ждем нажатия любой клавиши и закрываем все окна
cv2.waitKey()
cv2.destroyAllWindows()
```

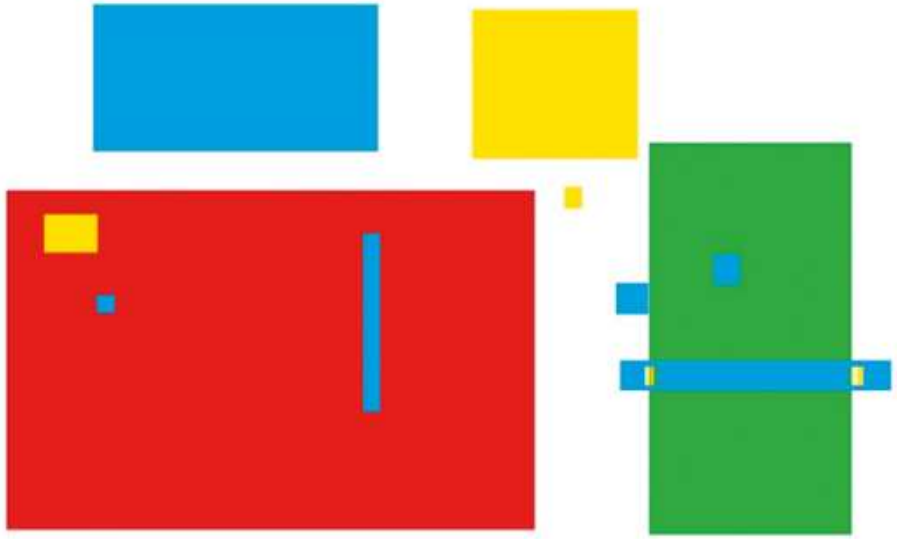
# ВЫДЕЛЕНИЕ ПРЯМОУГОЛЬНЫХ КОНТУРОВ

В результате работы программы было выделено два прямоугольных контура.
Внутренний контур полностью совпадает с прямоугольной границей объекта, а внешний прямоугольный контур описывает овальную границу объекта, причем, хотя и не совпадает с ней, описывает максимально близко. Как видим, функция cv2. minAreaRect() построила прямоугольный контур, который максимально близко описывает непрямоугольную границу.




Выделение прямоугольных контуров в тестовом изображении

# ФИЛЬТРАЦИЯ ПРЯМОУГОЛЬНЫХ КОНТУРОВ

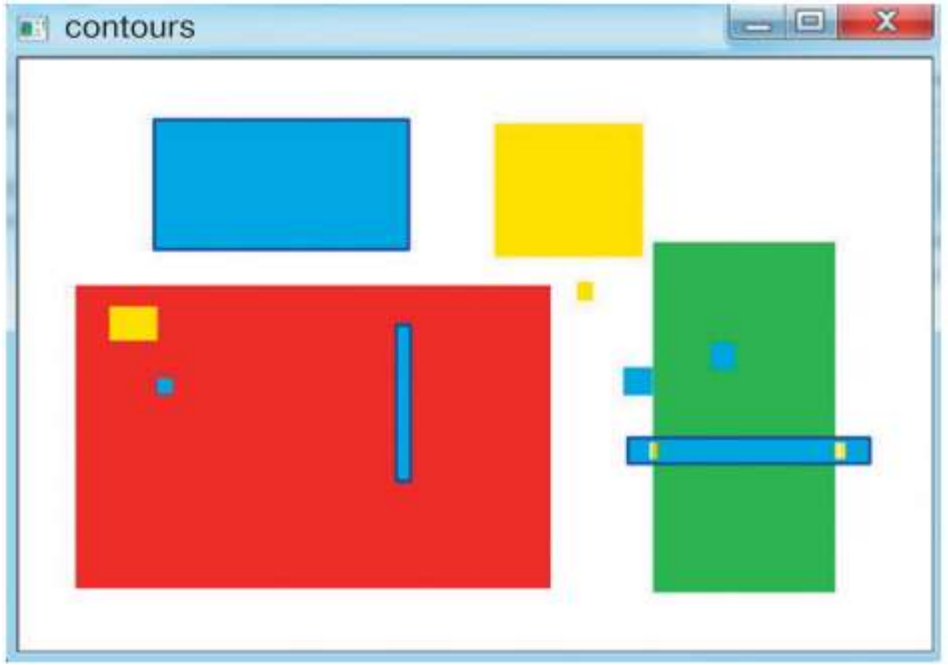


```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# задаем путь к файлу с картинкой,
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'rec_img1.jpg'
# считываем данные графического файла в переменную
# image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV,
# результат записываем в переменную hsv_img
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
```

```
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((90, 100, 100), np.uint8)
hsv_max = np.array((120, 255, 200), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                        cv2.RETR_LIST,
                                        cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
#   ищем прямоугольник, результат записываем в rect
    rect = cv2.minAreaRect(icontour)
# !!Вот здесь вставляются указанные выше две строки!!
#   вычисление площади
    area = int(rect[1][0]*rect[1][1])
```

# ФИЛЬТРАЦИЯ ПРЯМОУГОЛЬНЫХ КОНТУРОВ

```
# если площадь больше указанного значения, эти
# контуры выводим,
# значение подбираем экспериментально
    if area > 1500:
#   поиск вершин прямоугольника, результат записываем
#   в box
        box = cv2.boxPoints(rect)
#       округление координат вершин, результат
#       записываем в box
        box = np.int0(box)
#       рисуем прямоугольник поверх исходного
#       изображения, цвет синий, толщина линии 3,
#       поскольку рисуем единственный объект [box],
#       остальные параметры опускаем
        cv2.drawContours( image, [box], -1,
                          (255,0,0),3)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# ждем нажатия любой клавиши и закрываем все окна

cv2.waitKey()
cv2.destroyAllWindows()
```



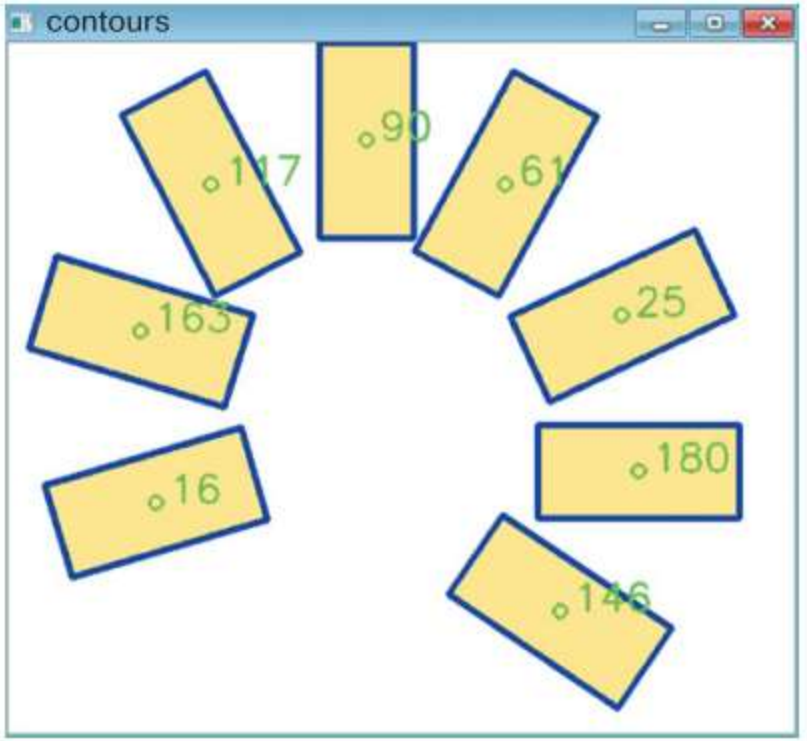

Результат работы программы фильтрации прямоугольных контуров в тестовом изображении

# ВЫЧИСЛЕНИЕ УГЛА ПОВОРОТА ПРЯМОУГОЛЬНОГО КОНТУРА

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# подключение библиотеки для математических расчетов
import math
# задаем путь к файлу с картинкой,
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'rec_img2.jpg'
# считываем данные графического файла в переменную
# image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV
# результат записываем в переменную hsv_img
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((0, 50, 50), np.uint8)
hsv_max = np.array((15, 255, 255), np.uint8)
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                        cv2.RETR_LIST,
                                    cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
#   ищем прямоугольник, результат записываем в rect
    rect = cv2.minAreaRect(icontour)
#   вычисление площади прямоугольного контура
    area = int(rect[1][0]*rect[1][1])
# отсекаем ложные контуры, если они вдруг появятся
    if area > 2500:



#   поиск вершин прямоугольника, результат записываем
#   в box
        box = cv2.boxPoints(rect)
#       округление координат вершин, результат
#       записываем в box
        box = np.int0(box)
#       рассчитываем координаты первого вектора
        vec1 = np.int0((box[1][0] - box[0][0],
                        box[1][1] - box[0][1]))
#       рассчитываем координаты второго вектора
        vec2 = np.int0((box[2][0] - box[1][0],
                        box[2][1] - box[1][1]))
#       мы не знаем, какой вектор больше, поэтому
#       думаем, что это будет первый вектор,
#       сохраняем его в used_vec
        used_vec = vec1
```

# ВЫЧИСЛЕНИЕ УГЛА ПОВОРОТА ПРЯМОУГОЛЬНОГО КОНТУРА

```
#       если длина второго вектора больше первого,
        if cv2.norm(vec2) > cv2.norm(vec1):
#       значит, в used_vec сохраним длину второго
#       вектора
            used_vec = vec2
#       записываем координаты центра прямоугольника
        center = (int(rect[0][0]),int(rect[0][1]))
#       рассчитываем угол наклона прямоугольника

        angle = 180.0/math.pi*math.acos(used_vec[0]/
                                        cv2.norm(used_vec))
#       рисуем прямоугольник
        cv2.drawContours( image, [box], -1,
                          (255,0,0),3)
#       рисуем метку-окружность в центре
#       прямоугольника
        cv2.circle(image, center, 5, (0,255,0), 2)
#       выводим рядом с прямоугольником значение угла
#       наклона
        cv2.putText(image, "%d" % int(angle),
                    (center[0]+10, center[1]),
                cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# ждем нажатия любой клавиши и закрываем все окна
cv2.waitKey()
cv2.destroyAllWindows()
```



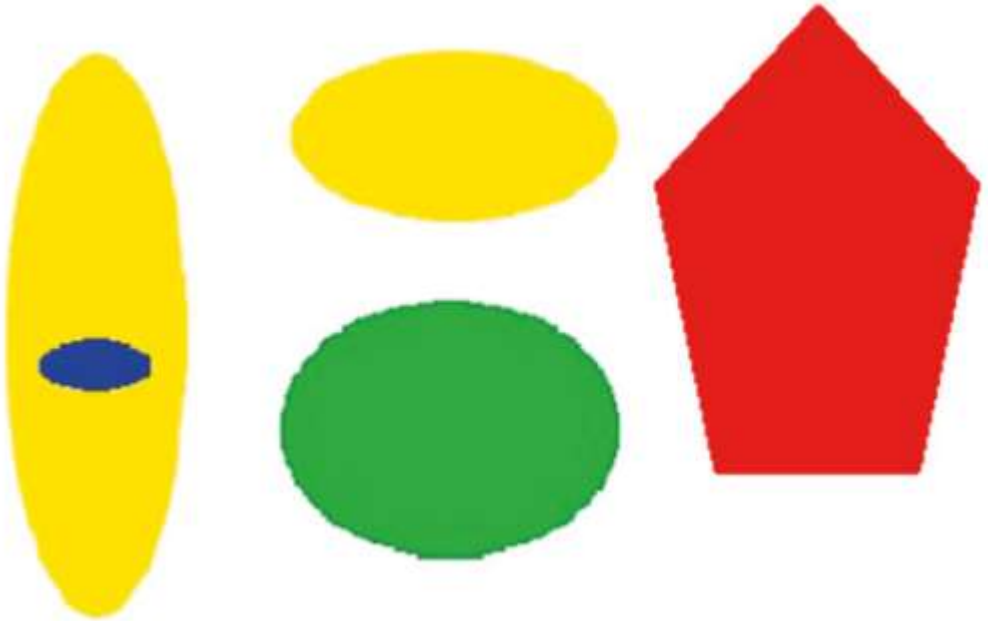

Вычисленные значения угла поворота прямоугольных контуров

# ВЫДЕЛЕНИЕ И ФИЛЬТРАЦИЯ ЭЛЛИПТИЧЕСКИХ КОНТУРОВ

Для выделения эллиптических контуров в OpenCV применяется функция **cv2.fitEllipse()**. Как и функция **cv2.inAreaRect()**, функция **cv2.fitEllipse()** не сможет отличить на картинке объект с действительно эллиптическим контуром от объекта с прямоугольным контуром. Эта функция лишь пытается *вписать* эллипс в любой контур с количеством точек, большим или равным 5. Чуть позже мы увидим, как функция **cv2.fitEllipse()** попытается *описать* объект неэллиптической формы. Результатом данной функции, согласно документации[1], является повернутый прямоугольник, в который вписан эллипс.

```
cv2.ellipse(img, box, color[, thickness[, lineType]])
```

где **img** — переменная с изображением;
**box** — альтернативное представление эллипса (эллипс вписан в повернутый прямоугольник);
**color** — цвет эллипса, задается комбинацией значений (B, G, R);
**thickness** — толщина линии в пикселях (необязательный параметр);
**lineType** — тип линии (необязательный параметр).



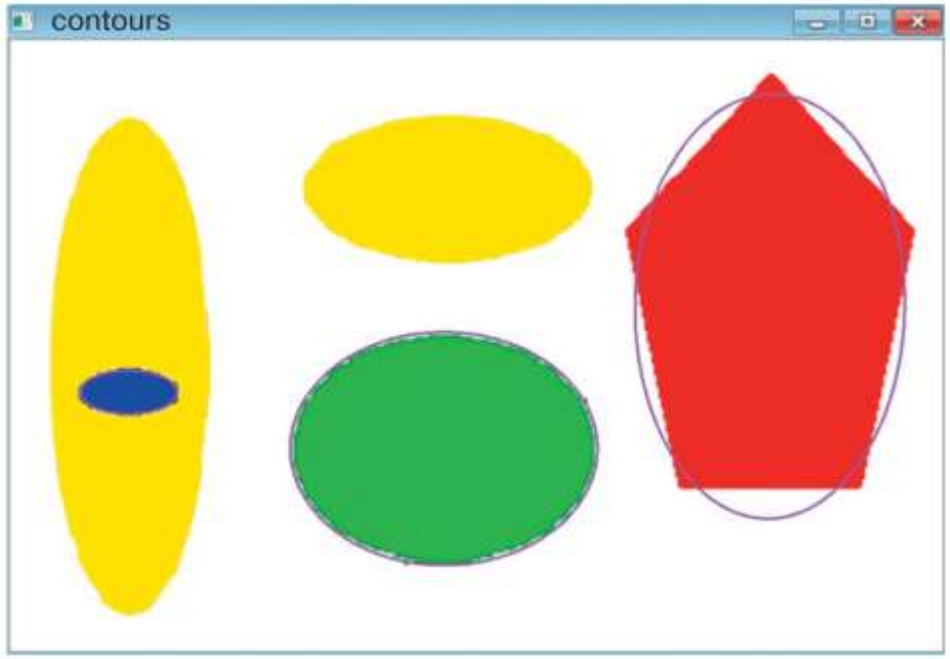
Пример графического изображения для выделения эллиптических контуров

# ВЫДЕЛЕНИЕ И ФИЛЬТРАЦИЯ ЭЛЛИПТИЧЕСКИХ КОНТУРОВ

```
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# задаем путь к файлу с картинкой,
# если файл с картинкой и файл с программой находятся
# в одной папке, то это будет просто имя графического
# файла
fileName = 'ellips.jpg'
# считываем данные графического файла в переменную
image
image = cv2.imread(fileName)
# конвертируем исходное изображение в цветовую модель
# HSV
# результат записываем в переменную hsv_img
hsv_img = cv2.cvtColor( image, cv2.COLOR_BGR2HSV )
# подбираем параметры цветового фильтра для выделения
# нашего объекта (указанные числовые значения могут
# отличаться)
hsv_min = np.array((60, 10, 10), np.uint8)
hsv_max = np.array((180, 255, 255), np.uint8)
```

```
# применяем цветовой фильтр к исходному изображению,
# результат записываем в переменную hsv_msk
hsv_msk = cv2.inRange( hsv_img, hsv_min, hsv_max )
# ищем контуры и записываем их в переменную contours
# в режиме поиска всех контуров без группировки
# cv2.RETR_LIST, для хранения контуров используем
# метод cv2.CHAIN_APPROX_SIMPLE
contours, hierarchy = cv2.findContours( hsv_msk,
                                        cv2.RETR_LIST,
                                    cv2.CHAIN_APPROX_SIMPLE)
# перебираем все найденные контуры в цикле
for icontour in contours:
#   выбираем контуры с длиной больше 40 точек
    if len(icontour)>40:
#       записываем в переменную ellipse
#       отвечающий условию контур в форме эллипса
        ellipse = cv2.fitEllipse(icontour)
#       отображаем найденный эллипс
        cv2.ellipse(image, ellipse, (255,0,255), 2)
# выводим итоговое изображение в окно contours
cv2.imshow('contours', image)
# выводим результат фильтрации изображения в окно HSV
# cv2.imshow('hsv', hsv_msk)
# ждем нажатия любой клавиши и закрываем все окна
cv2.waitKey()
cv2.destroyAllWindows()
```

# ВЫДЕЛЕНИЕ И ФИЛЬТРАЦИЯ ЭЛЛИПТИЧЕСКИХ КОНТУРОВ





Если приглядимся повнимательнее, то увидим, что желтые овалы остались без контуров. Заглянув в параметры цветового фильтра, мы обнаружим, что желтый цвет находится за пределами границ фильтра. В этом можно убедиться, если раскомментировать в программе четвертую строчку снизу и запустить весь код на исполнение. Теперь мы увидим окно, в котором нет «следов» желтых овалов

# ВЫДЕЛЕНИЕ КОНТУРОВ ДОРОЖНЫХ ЗНАКОВ В ВИДЕОПОТОКЕ

```
# ПОДКЛЮЧАЕМ БИБЛИОТЕКИ И ПОЛУЧАЕМ ДОСТУП
# К ВИДЕОФАЙЛУ
# подключение библиотеки numpy
import numpy as np
# подключение библиотеки opencv
import cv2
# подключение библиотеки функций работы со временем,
# она будет нужна для изменения скорости смены кадров
import time
# связываем видеопоток файла video.avi с переменной
# capImg
capImg = cv2.VideoCapture('video.avi')

# ИЩЕМ ЗНАКИ С ПОМОЩЬЮ HSV-ФИЛЬТРОВ
# открываем файл с видео
while(capImg.isOpened()):
#   получаем кадр из видеопотока файла,
#   кадры по очереди считываются в переменную frame
    ret, frame = capImg.read()
#   если кадры закончились, то прерываем цикл
    if frame is None:
        break
#   переводим кадр в цветовое пространство HSV
    frame_hsv = cv2.cvtColor(frame,
                             cv2.COLOR_BGR2HSV)
#   делаем вырезки области кадра, где ожидаются знаки
    crop_frame = frame[160:270, 150:650]
    crop_frame_hsv = frame_hsv[160:270, 150:650]
#   задаем нижние и верхние границы цветовых фильтров
#   с помощью массивов numpy
#   цвет 0...180
#   насыщенность 0 - блеклый, 255 - насыщенный
#   яркость 0 - темный, 255 - светлый
#   фильтр для синего цвета
    low_Blue = np.array([105, 150, 0],
                        dtype = "uint8")
    high_Blue = np.array([135, 255, 255],
                         dtype = "uint8")
#   фильтр для желтого цвета
    low_Yel = np.array([25, 95, 100],
                       dtype = "uint8")
    high_Yel = np.array([35, 255, 255],
                        dtype = "uint8")
#   фильтр для красного цвета
#   красный цвет содержит две области
#   красный в сторону оранжевой области
    low_Red_O = np.array([0, 85, 110],
                         dtype = "uint8")
    high_Red_O = np.array([5, 165, 155],
                          dtype = "uint8")
#   красный в сторону фиолетовой области
    low_Red_V = np.array([165, 55, 40],
                         dtype = "uint8")
    high_Red_V = np.array([180, 105, 120],
                          dtype = "uint8")
#   применяем маску по каждому цвету к фрагменту
#   кадра для выделения синего цвета
    blue_mask = cv2.inRange(crop_frame_hsv, low_Blue,
                            high_Blue)
#   для выделения желтого цвета
    yel_mask = cv2.inRange(crop_frame_hsv, low_Yel,
                           high_Yel)
#   для выделения красного накладываются две маски
    red1_mask = cv2.inRange(crop_frame_hsv,
                            low_Red_O, high_Red_O)
    red2_mask = cv2.inRange(crop_frame_hsv,
                           low_Red_V, high_Red_V)
```

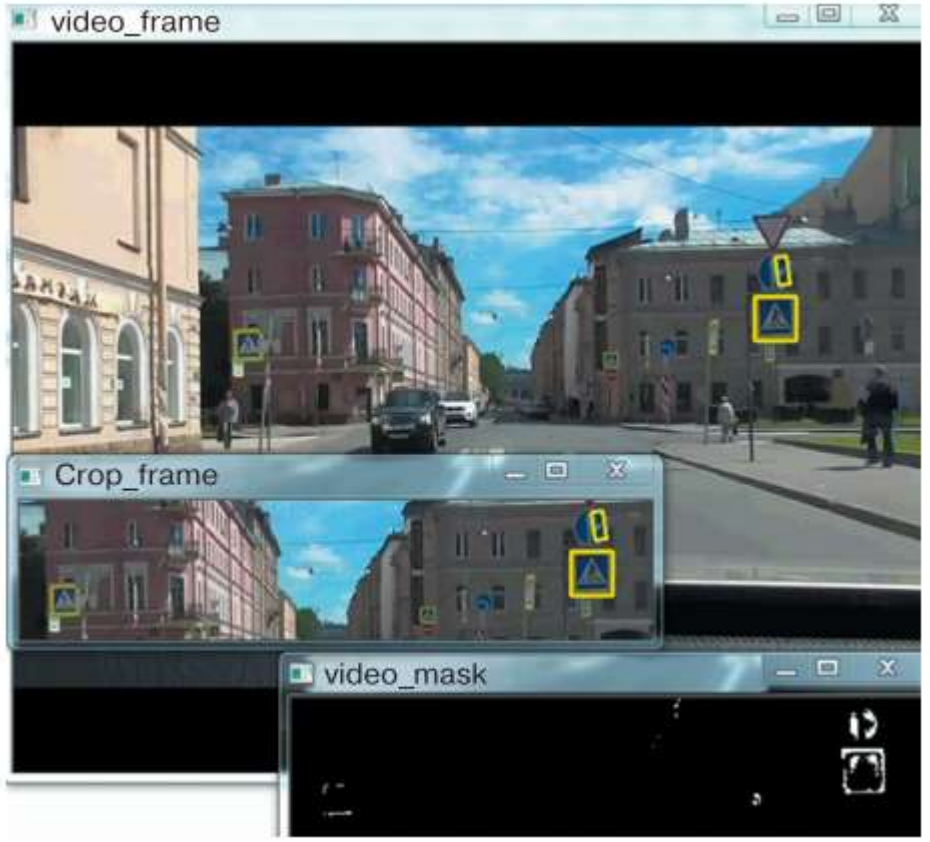
# ОПТИМИЗАЦИЯ НЕПРЕРЫВНЫХ ФУНКЦИЙ

```
#    вычисляем полную маску
#    полная маска представляет собой сумму всех масок
     full_mask = red1_mask + red2_mask + blue_mask +
     yel_mask

# ЗАКЛЮЧАЕМ В ПРЯМОУГОЛЬНЫЕ КОНТУРЫ НАЙДЕННЫЕ ОБЪЕКТЫ
#    ищем контуры и записываем их в переменную
#    contours

#    в режиме поиска всех контуров без группировки
#    cv2.RETR_LIST, для хранения контуров используем
#    метод cv2.CHAIN_APPROX_SIMPLE
     contours, hierarchy = cv2.findContours(full_mask.
                                            copy(),
                                          cv2.RETR_TREE,
                                 cv2.CHAIN_APPROX_SIMPLE)
#    перебираем все найденные контуры в цикле
     for icontour in contours:
#        ищем прямоугольник, результат записываем
#        в rect
         rect = cv2.minAreaRect(icontour)
#        вычисление площади прямоугольного контура
         area = int(rect[1][0]*rect[1][1])
```

```
# ФИЛЬТРУЕМ МЕЛКИЕ КОНТУРЫ
#        отсекаем ложные контуры, если они вдруг
#        появятся
         if area > 100:
#        поиск вершин прямоугольника, результат
#        записываем в box
             box = cv2.boxPoints(rect)
#        округление координат вершин, результат
#        записываем в box
             box = np.int0(box)
#            рисуем прямоугольник
             cv2.drawContours(crop_frame,[box],0,
                              (0,255,255),2)

# ВЫВОДИМ РЕЗУЛЬТАТЫ НА ЭКРАН
#   выводим изображения в окнах на экран
     cv2.imshow("video_mask", full_mask)
     cv2.imshow("video_frame", frame)
     cv2.imshow("Crop_frame", crop_frame)
#    задержка по времени, аргумент подбираем
#    экспериментально
     time.sleep(0.1)
#    организуем выход из цикла по нажатию клавиши,
#    ждем 30 миллисекунд нажатия, записываем код
#    нажатой клавиши
     key_press = cv2.waitKey(30)
```

# ОПТИМИЗАЦИЯ НЕПРЕРЫВНЫХ ФУНКЦИЙ

```
#   если код нажатой клавиши совпадает с кодом «q»,
    if key_press == ord('q'):
        break
# освобождаем память от переменной cap
cap.release()
# закрываем все окна opencv
cv2.destroyAllWindows()
```




Результат работы программы, где на одном из кадров запечатлен момент с обнаружением дорожного знака «Пешеходный переход». Сразу видно, что даже простейшая фильтрация контуров по размерам позволяет игнорировать мелкие артефакты, которые появляются при поиске цветных объектов. Чтобы повысить качество поиска дорожных знаков или других объектов на видео, надо понимать, что оно напрямую зависит от качества самого видео. Поэтому вполне допустимо, что на видео не очень хорошего качества наша программа, кроме дорожных знаков, найдет что-то еще.

# ГЛОССАРИЙ

**ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) — название кодировочной таблицы, в которой распространенным печатным и непечатным символам сопоставлены числовые коды.

**AVI** (**A**udio **V**ideo **I**nterleave) — медиаконтейнер, разработанный компанией Microsoft для пакета Video for Windows. Файл AVI может содержать видео- и аудиоданные, сжатые с помощью разных комбинаций видео- и аудиокодеков, что позволяет синхронно воспроизводить видео со звуком.

**BGR** — цветовое пространство: синий (**B**lue), зеленый (**G**reen), красный (**R**ed); каждая компонента может принимать значение в диапазоне 0–255.

**break** — команда, прерывает выполнение любого цикла.

**cv2.boxPoints()** — функция, возвращает массив из вещественных координат четырех точек — вершин прямоугольника.

**cv2.createTrackbar()** — функция, создает графический компонент «ползунок с указателем».

**cv2.cvtColor()** — функция, конвертирует исходное изображение из одного цветового пространства в другое и возвращает сконвертированное изображение.

**cv2.destroyAllWindows()** — функция, закрывает все графические окна.

**cv2.drawContours()** — функция, отображает контуры, найденные функцией **cv2.findContours()**.

**cv2.ellipse()** — функция, отображает найденный эллиптический контур.

**cv2.findContours()** — функция, возвращает замкнутые контуры графических областей в виде массива точек.

**cv2.fitEllipse()** — функция, возвращает повернутый прямоугольник, в который вписан эллипс длиной не менее 5 точек.

**cv2.flip()** — функция, возвращает зеркальную копию исходного изображения.

**cv2.getRotationMatrix2D()** — функция, возвращает двумерный объект, в котором будет размещено графическое изображение.

**cv2.getTrackbarPos()** — функция, считывает положение указателя ползунка.

**cv2.imread()** — функция, считывает массив данных из файла с изображением.

**cv2.imshow()** — функция, отображает графические данные на экран.

**cv2.imwrite()** — функция, сохраняет изображение в графический файл.

**cv2.inRange()** — функция, возвращает черно-белое изображение, где белыми пикселями обозначены области, цвета которых попали в заданный цветовой фильтр.

**cv2.minAreaRect()** — функция, ищет прямоугольный контур минимальной площади, способный описать заданный замкнутый контур.

**cv2.moments()** — функция, вычисляет моменты графической области белого цвета на черно-белом изображении.

# ГЛОССАРИЙ

**cv2.namedWindow()** — функция, создает графическое окно.

**cv2.putText()** — функция, выводит в графическое окно OpenCV текстовое сообщение.

**cv2.resize()** — функция, возвращает трансформированное изображение.

**cv2.VideoCapture()** — функция, обеспечивает идентификацию и доступ к потоку видеоданных.

**cv2.waitKey()** — функция, ожидает нажатия любой клавиши с клавиатуры при пустом аргументе или нажатия клавиши, соответствующей символу в одинарных кавычках, например 'n'.

**cv2.warpAffine()** — функция, выполняет поворот изображения и возвращает новое преобразованное изображение (в случае трансформации и поворота).

**HSV(HSB)** — цветовое пространство: цветовой тон (**H**ue), насыщенность (**S**aturation) и значение или яркость (**V**alue или **B**rightness). Под цветовым тоном понимается именно цвет (0–180). Насыщенность характеризует близость цвета к белому (0–255). Значение или яркость представляет как общую яркость точки или цвета (0–255).

**image.shape[i]** — метод, команда, возвращает ширину (при i = 1) или высоту (при i = 0) исходного изображения **image**.

**ord()** — функция, возвращает код указанного символа.

**pass** — оператор-заглушка, не выполняет никаких действий.

**Region Of Interest (ROI)** — определенная пользователем прямоугольная область на рисунке.

**Арккосинус** — тригонометрическая функция, обратная функции косинуса и возвращающая значение угла, если известен косинус.

**Библиотека OpenCV (Open Source Computer Vision Library)** — библиотека компьютерного зрения с открытым исходным кодом.

**Косинус** острого угла $\alpha$ в прямоугольном треугольнике — это отношение прилежащего катета $AC$ к гипотенузе $AB$: $\cos\alpha = AC/AB$.

**Нейронная сеть** — в нашем случае это искусственная нейронная сеть, представляет собой математическую модель, которая может быть реализована в программном коде или в аппаратном устройстве. Нейронные сети используются для решения задач адаптивного управления, распознавания образов, в системах искусственного интеллекта и т. д.

# РАСПОЗНАВАНИЕ ОБЪЕКТОВ

Модели нейронных сетей

```python
import cv2

img = cv2.imread('images/people_2.jpeg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = cv2.CascadeClassifier('faces.xml')

results = faces.detectMultiScale(gray, scaleFactor=1.8, minNeighbors=4)

for (x, y, w, h) in results:
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), thickness=3)

cv2.imshow("Result", img)
cv2.waitKey(0)
```

| | |
|---|---|
| haarcascade_eye.xml | some attempts to tune the performance |
| haarcascade_eye_tree_eyeglasses.xml | some attempts to tune the performance |
| haarcascade_frontalcatface.xml | fix files permissions |
| haarcascade_frontalcatface_extended.xml | fix files permissions |
| haarcascade_frontalface_alt.xml | some attempts to tune the performance |
| haarcascade_frontalface_alt2.xml | some attempts to tune the performance |
| haarcascade_frontalface_alt_tree.xml | some attempts to tune the performance |
| haarcascade_frontalface_default.xml | some attempts to tune the performance |
| haarcascade_fullbody.xml | Some mist. typo fixes |
| haarcascade_lefteye_2splits.xml | some attempts to tune the performance |
| haarcascade_license_plate_rus_16stages.xml | Merge pull request #22727 from su77ungr:patch-1 |
| haarcascade_lowerbody.xml | Some mist. typo fixes |
| haarcascade_profileface.xml | some attempts to tune the performance |
| haarcascade_righteye_2splits.xml | some attempts to tune the performance |
| haarcascade_russian_plate_number.xml | Create haarcascade_russian_plate_number.xml |
| haarcascade_smile.xml | fixing models to resolve XML violation issue |
| haarcascade_upperbody.xml | Some mist. typo fixes |

# ЗАДАНИЕ ДЛЯ САМОКОНТРОЛЯ

Используя любую готовую модель распознать объекты с помощью библиотеки **Python OpenCV**

# СПАСИБО ЗА ВНИМАНИЕ!