Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Романов В.В.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Крамаренко А.А.

**Цель работы:** Реализация трекинга.

**Ход работы:**

Задание 1. Прочитать изображение с камеры и перевести его в формат HSV.

Перевести изображение в формат HSV можно с помощью команды cv2.cvtColor(), указав в аргументах флаг HSV формата (cv2.COLOR\_BGR2HSV).

RGB и HSV – это 2 различных цветовых пространства, которые используются для представления цветов. RGB – это аддитивное цветовое пространство, которое используется для представления цветов на экране. Оно состоит из красного, зеленого и синего цветов. HSV – это цветовое пространство, которое используется для представления цветов в виде оттенка, насыщенности и значения. Он также называется HSB (оттенок, насыщенность, яркость). Он позволяет легче управлять цветом, чем RGB. В аддитивном цветовом пространстве цвет создается путем смешивания света разных цветовых компонентов. Например, красный, зеленый и синий свет смешиваются вместе, чтобы создать белый свет. Это отличается от вычитательного цветового пространства, которое используется для представления цветов в печати и других физических процессах. В вычитательном цветовом пространстве цвет создается путем вычитания определенных цветовых компонентов из белого света.

Для перевода цвета из RGB в HSV выполняются следующие действия:

1)Необходимо нормализовать значения R, G и B, чтобы они находились в диапазоне от 0 до 1 – поделить значения на 255.

2)Поиск максимального и минимального значений из R, G и B.

3)Вычисляется разность между максимальным и минимальным значением.

4)Вычисляется значение яркости как Max значение из R, G и B.

5)Вычисляется насыщенность как разность между Max значением и значением яркости, деленную на Max значение.

6) Оттенок вычисляется следующим образом:

Если максимальным значением является R, то H = 60 \* ((G-B)/delta) (если G < B, то H += 360). Если максимальным значением является G, то H = 60 \* ((B-R)/delta + 2). Если максимальным значением является B, то H = 60 \* ((R-G)/delta + 4). Здесь delta = max(R, G, B) - min(R, G, B).

Обратный перевод выполняется следующим образом:

1) Нормализация значения H, S и V, чтобы они находились в диапазоне от 0 до 1.

2) Вычисление следующих значений: C = V \* S, X = C \* (1 - |(H / 60°) mod 2 - 1|) и m = V - C.

3) Вычисление значения R’, G’ и B’ следующим образом:

• Если 0 ≤ H < 60, то R’ = C, G’ = X, B’ = 0.

• Если 60 ≤ H < 120, то R’ = X, G’ = C, B’ = 0.

• Если 120 ≤ H < 180, то R’ = 0, G’ = C, B’ = X.

• Если 180 ≤ H < 240, то R’ = 0, G’ = X, B’ = C.

• Если 240 ≤ H < 300, то R’ = X, G’ = 0, B’ = C.

• Если 300 ≤ H < 360, то R’ = C, G’ = 0, B’ = X.

4) Вычисление значения R, G и B следующим образом: R = (R’ + m) \* 255; G = (G’ + m) \* 255; и B = (B’ + m) \* 255.

Например, если есть пиксель с HSV-значениями (120°, 0.5, 0.5), то его RGB-значения будут следующими: R = 64, G = 128 и B = 64 – красный цвет.

Теперь преобразуем видеопоток в цветовой формат HSV. Результат данного действия представлен на рисунке 1.



Рисунок 1 – HSV формат.

**Задание 2.** Применить фильтрацию изображения с помощью команды inRange и оставить только красную часть, вывести получившееся изображение на экран(treshold), выбрать красный объект и протестировать параметры фильтрации, подобрав их нужного уровня.

Преобразуем кадр в цветовое пространство HSV: hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV): Конвертирует BGR-изображение в HSV (оттенок, насыщенность, значение), что облегчает фильтрацию по цвету.

Определим цветовую маску:

lower\_red и upper\_red: Определяют диапазон красного цвета в формате HSV.

mask = cv2.inRange(hsv, lower\_red, upper\_red): Создает маску, которая выделяет области, соответствующие заданному диапазону красного цвета.

Применим маску к кадру:

onlyRed\_frame = cv2.bitwise\_and(frame, frame, mask=mask): Применяет маску к исходному кадру, оставляя только пиксели, соответствующие красному цвету.

Отображение фильтрованного изображения:

cv2.imshow('Red Filtered Image', onlyRed\_frame): Отображает фильтрованное изображение, на котором видны только красные объекты. На рисунке 2 представлен результат работы программы.



Рисунок 2 – Выделение красного цвета.

**Задание 3.** Провести морфологические преобразования (открытие и закрытие) фильтрованного изображения, вывести результаты на экран, посмотреть смысл подобного применения операций erode и dilate.

Морфологические преобразования – это операции, применяемые к изображениям с целью изменения их формы и структуры на основе формы и структуры объектов на изображении. Два из наиболее распространенного морфологического преобразования – это "открытие" и "закрытие". Эти операции часто используются в обработке изображений для удаления шума, заполнения дыр в объектах и изменения размера объектов. Рассмотрим каждую из них более подробно:

Открытие (Opening):

1. Операция открытия состоит из двух шагов: сначала применяется эрозия (erode), а затем – дилатация (dilate).
2. Эрозия удаляет маленькие объекты и "шум" на изображении, уменьшая объекты и заполняя небольшие прорехи.
3. Дилатация восстанавливает объекты близкой к их исходному размеру.
4. Открытие полезно для удаления шума, разделения объектов, связанных друг с другом, и выделения объектов, близких к заданной форме.

Закрытие (Closing):

1. Операция закрытия также состоит из двух шагов: сначала применяется дилатация (dilate), а затем – эрозия (erode).
2. Сначала дилатация увеличивает объекты и заполняет небольшие отверстия в объектах.
3. Затем эрозия уменьшает объекты обратно к их исходному размеру.
4. Закрытие полезно для заполнения небольших отверстий в объектах и объединения близко расположенных объектов.

Применение открытия и закрытия может быть полезным при обработке изображений в различных задачах компьютерного зрения, таких как сегментация объектов, удаление шума и анализ текстур. Эти операции особенно эффективны в тех случаях, когда объекты на изображении имеют различные размеры или, когда на изображении присутствует некоторый уровень шума.

Использование морфологических преобразований требует выбора правильных структурирующих элементов (ядро) и настройки их размера в зависимости от конкретной задачи.

Ниже на рисунке 3 представлена работа программы с добавлением операций открытия и закрытия соответственно.

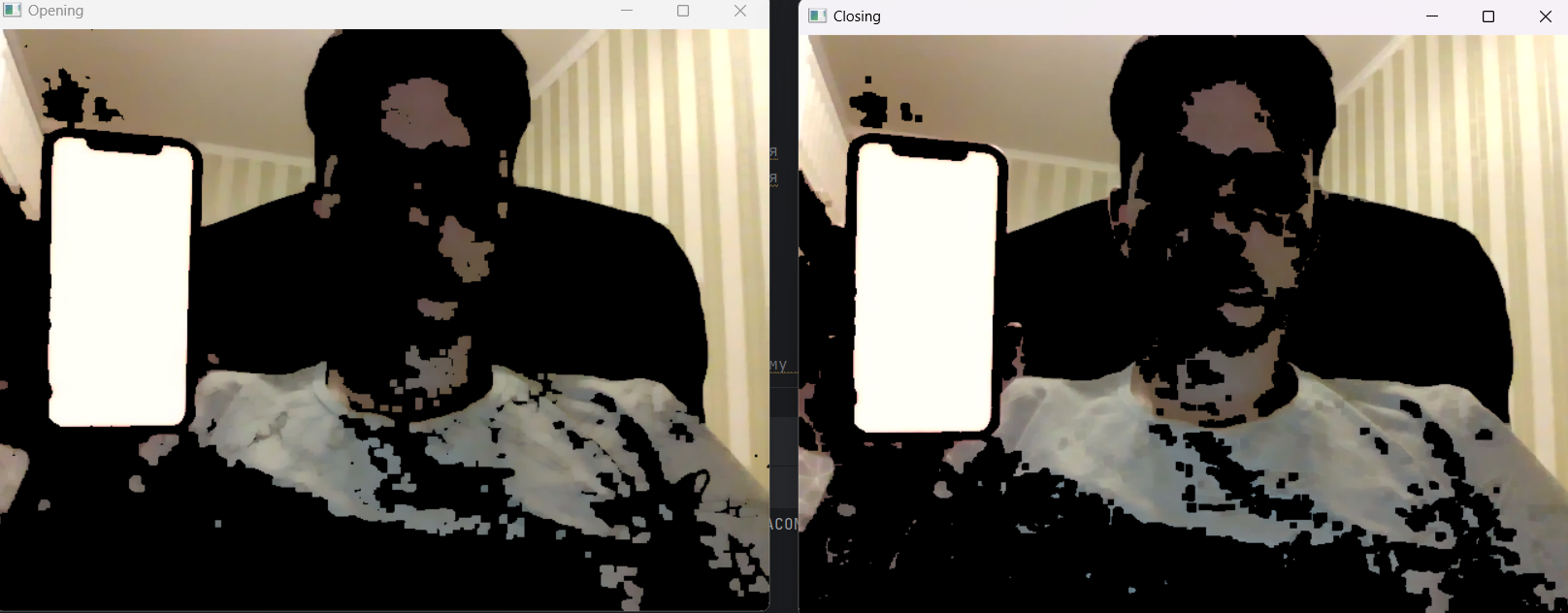


Рисунок 3 – Операции open и close.

**Задание 5.** На основе анализа площади объекта найти его центр и построить черный прямоугольник вокруг объекта. Сделать так, чтобы на видео выводился полученный черный прямоугольник, причем на новом кадре.

Моменты изображения – это статистические показатели, которые описывают форму, структуру и распределение пикселей в изображении.

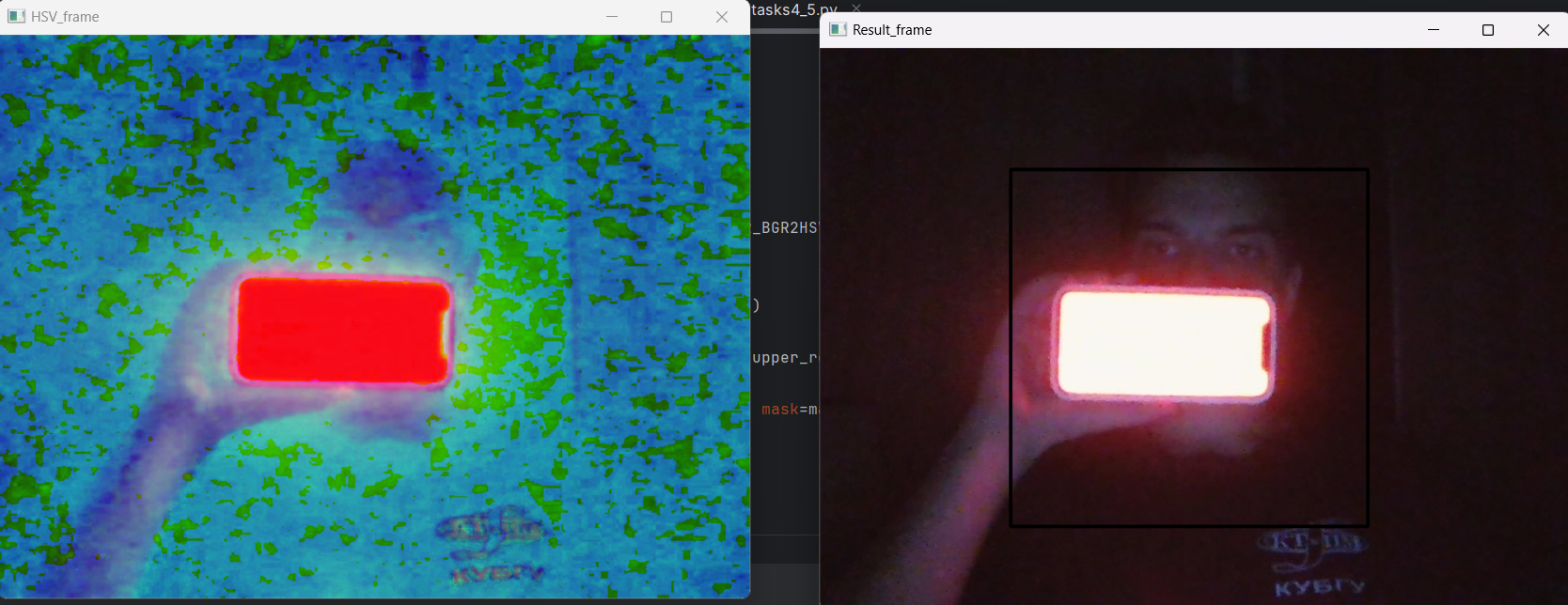
Рассмотрим их составляющие подробнее:

1. m00 – это один из моментов изображения, который называется нулевым моментом. Нулевой момент представляет собой интегральную яркость (или массу) всего изображения и используется для вычисления центра масс объекта.
2. area – представляет собой значение нулевого момента, которое соответствует площади объекта на изображении. Это значение используется для определения площади объекта.
3. Центроид объекта на изображении – это центр масс или среднее положение всех пикселей, составляющих объект.

По координатам m10 и m01 можно вычислить координаты центра масс объекта (c\_x, c\_y).

Далее рисуется черный прямоугольник вокруг центра масс объекта.

На рисунке 4 представлен результат работы программы.



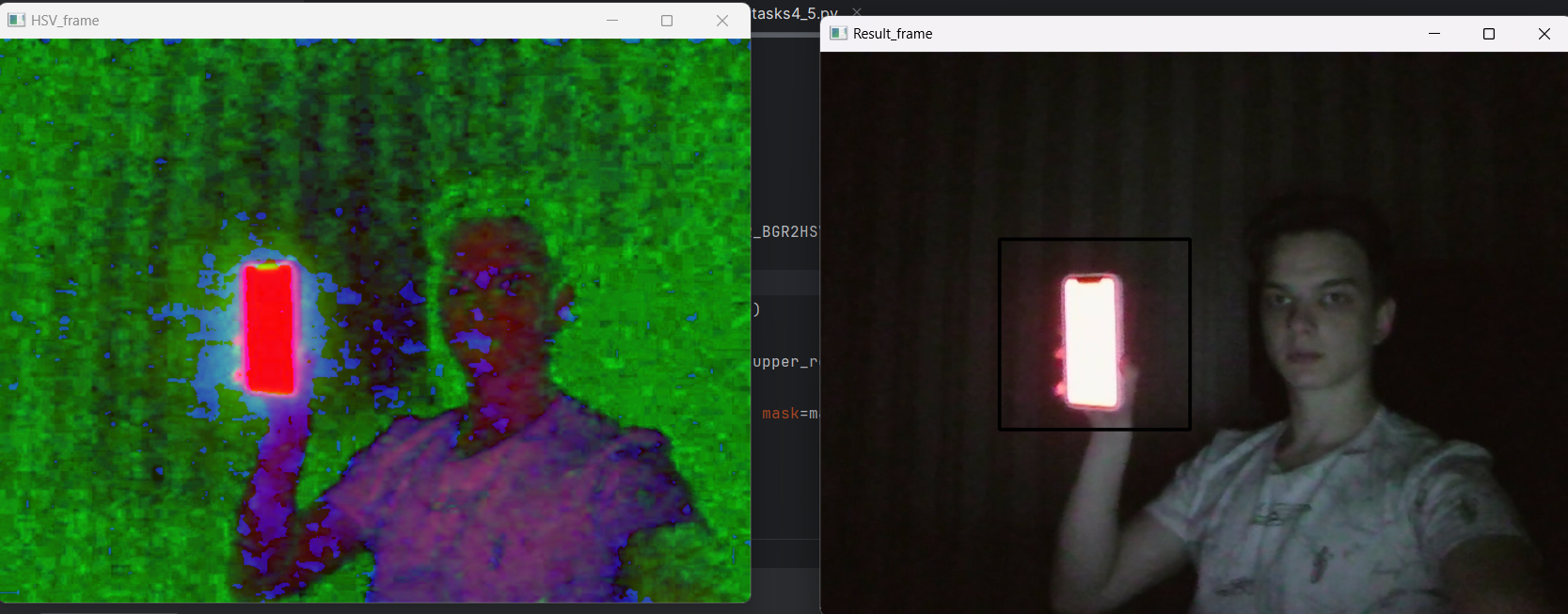


Рисунок 4 – Трекинг красного цвета с прямоугольником.

**Листинг программы**

**Файл task1.py:**

import cv2  
import numpy as np  
  
cap = cv2.VideoCapture(0)  
  
while True:  
 ret, frame = cap.read()  
  
 hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)  
  
 cv2.imshow("HSV\_image", hsv)  
  
 cv2.imwrite("task\_1.png", hsv)  
  
 if cv2.waitKey(1) & 0xFF == 27:  
 break  
  
cap.release()  
cv2.destroyAllWindows()

**Файл task2.py:**

import cv2  
import numpy as np  
  
cap = cv2.VideoCapture(0)  
  
while True:  
 ret, frame = cap.read()

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)  
  
 lower\_red = np.array([0, 120, 220])  
 upper\_red = np.array([10, 255, 255])  
  
 mask = cv2.inRange(hsv, lower\_red, upper\_red)  
 onlyRed\_frame = cv2.bitwise\_and(frame, frame, mask = mask)  
  
 cv2.imshow('Red Filtered Image', onlyRed\_frame)  
  
 if cv2.waitKey(1) & 0xFF == 27:  
 break  
  
cap.release()  
cv2.destroyAllWindows()

**Файл task3.py:**

import cv2  
import numpy as np  
  
cap = cv2.VideoCapture(0)  
  
while True:  
 ret, frame = cap.read()  
 hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)  
  
 *#Определяем диапазон красного цвета в HSV*  
lower\_red = np.array([0, 0, 100]) *#Min значения*  
upper\_red = np.array([100, 100, 255]) *#Max значения*  
  
mask = cv2.inRange(hsv, lower\_red, upper\_red)  
  
 *#Применяем маску на изображение*  
res = cv2.bitwise\_and(frame, frame, mask=mask)  
  
 *#Структурирующий элемент, определяющий размер и форму области*  
kernel = np.ones((5, 5), np.uint8)  
  
 *#Применяем операцию открытия, чтобы удалить шумы и мелкие объекты на изображении*  
opening = cv2.morphologyEx(res, cv2.MORPH\_OPEN, kernel)  
  
 *#Применяем операцию закрытия, чтобы заполнить маленькие пробелы и разрывы в объектах на изображении*  
closing = cv2.morphologyEx(res, cv2.MORPH\_CLOSE, kernel)  
  
 cv2.imshow('Opening', opening)  
 cv2.imshow('Closing', closing)  
  
 if cv2.waitKey(1) & 0xFF == 27:  
 break  
  
cap.release()  
cv2.destroyAllWindows()

**Файл tasks4\_5.py:**

import cv2  
import numpy as np  
  
cap = cv2.VideoCapture(0)  
  
while True:  
 ret, frame = cap.read()  
 hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)  
  
 lower\_red = np.array([0, 50, 50])  
 upper\_red = np.array([10, 255, 255])  
  
 mask = cv2.inRange(hsv, lower\_red, upper\_red)  
  
 res = cv2.bitwise\_and(frame, frame, mask=mask)  
  
 *#Вычисляем момент на основе маски*  
moments = cv2.moments(mask)  
  
 *#Поиск момента первого порядка*  
area = moments['m00']  
  
 if area > 0:  
 width = height = int(np.sqrt(area))  
  
 *#Вычисляем координаты центра объекта на изображении с использованием момент первого порядка*  
c\_x = int(moments["m10"] / moments["m00"])  
 c\_y = int(moments["m01"] / moments["m00"])  
  
 *#Отрисовка прямоугольника*  
color = (0, 0, 0)  
 thickness = 2  
 cv2.rectangle(frame,  
 (c\_x - (width // 8), c\_y - (height // 8)),  
 (c\_x + (width // 8), c\_y + (height // 8)),  
 color, thickness)  
  
 cv2.imshow('HSV\_frame', hsv)  
 cv2.imshow('Result\_frame', frame)  
  
 if cv2.waitKey(1) & 0xFF == 27:  
 break  
  
print("Площадь объекта:", area)  
  
cap.release()  
cv2.destroyAllWindows()