Лабораторна робота №8

Тема: ДОСЛІДЖЕННЯ МЕТОДІВ КОМП’ЮТЕРНОГО ЗОРУ

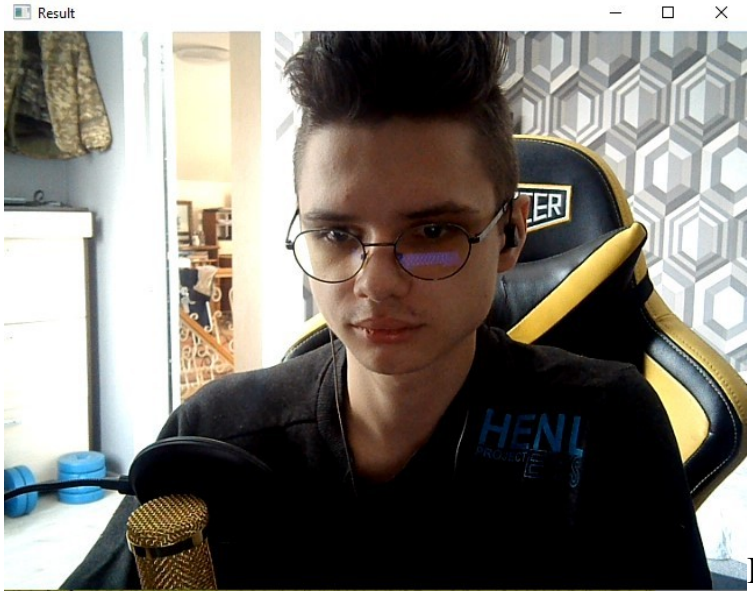
Мета: *використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися обробляти зображення за допомогою бібліотеки OpenCV.*

Хід роботи

GitHub репозиторій:
https://github.com/BashmanivskiyMaxim/Artificial_intelligence_labs

Завдання 2.1: Завантаження зображень та відео в OpenCV

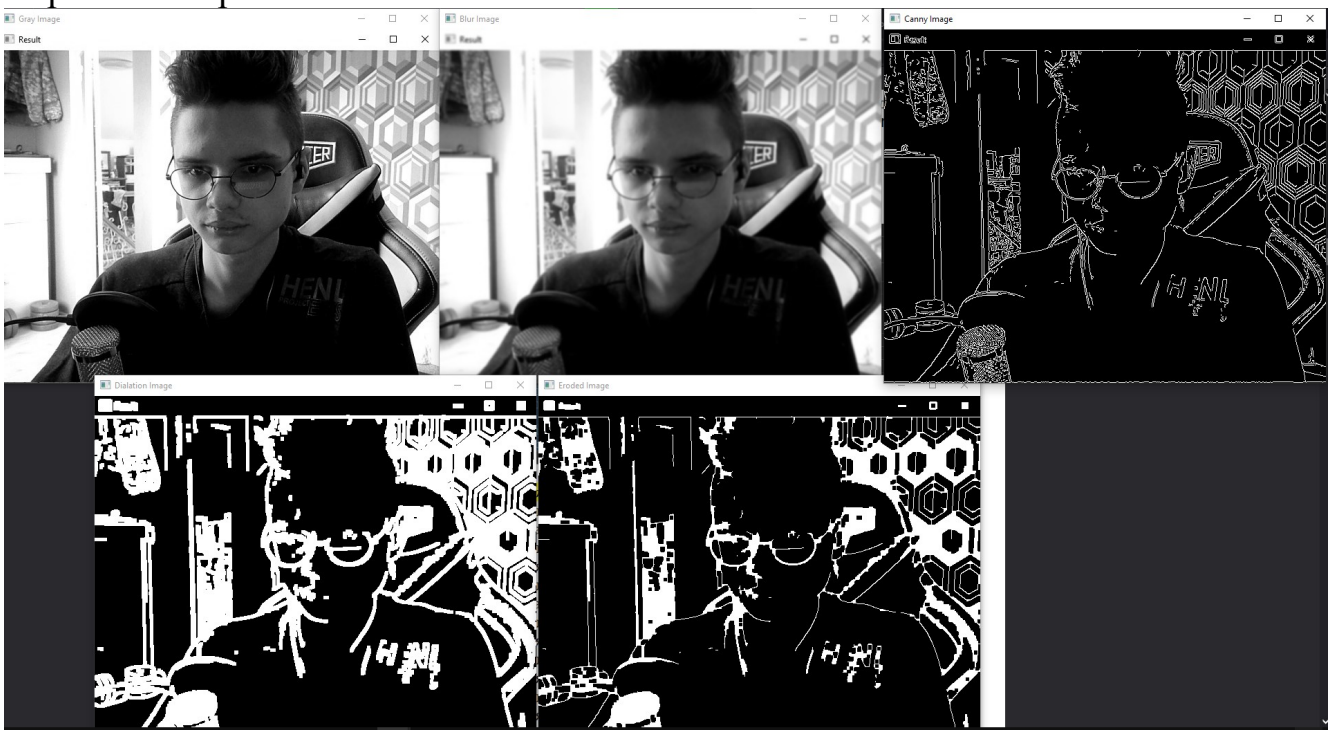
Отримане зображення:



Висновок: Цей код демонструє базовий функціонал бібліотеки OpenCV для роботи з відеопотоком та зображеннями. Він дозволяє захоплювати відеопотік з вебкамери та відображати зображення з файлу на екрані.

Завдання 2.2: Дослідження перетворень зображення

Отримані зображення:



Висновок:

1. **Метод** cvtColor **(колірне перетворення)**:

Застосування методу cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) перетворює кольорове зображення img у відтінки сірого. Іншими словами, воно видаляє колірну інформацію та залишає лише яскравість. В результаті отримуємо сірошкальне зображення (градації сірого), яке зазвичай використовується для спрощення обробки та аналізу зображень.

2. **Метод** GaussianBlur **(розмиття Гауса)**:

Застосування cv2.GaussianBlur(imgGray, (7, 7), 0) виконує розмиття Гауса на сірому зображенні imgGray. Розмиття Гауса використовується для зменшення шуму та видалення деталей. У результаті отримуємо зображення, в якому дрібні деталі та шум стають менш помітними.

3. **Метод** Canny **(детектор границь Canny)**:

Використання cv2.Canny(img, 150, 200) дозволяє виявити границі на вхідному зображенні. Детектор Canny ідентифікує різницю в яскравості між сусідніми пікселями та виділяє границі об'єктів на зображенні. Операція Canny генерує бінарне зображення з виявленими границями, де білим позначені границі, а чорним фон.

4. **Метод** dilate **(діляція)**:

cv2.dilate(imgCanny, kernel, iterations=1) виконує морфологічну операцію діляції на бінарному зображенні imgCanny з використанням заданого ядра kernel. Діляція використовується для збільшення об'єму об'єктів на зображенні, розширюючи їх. У результаті об'єкти на зображенні стають більшими та більш помітними.

5. **Метод** erode **(ерозія)**:

cv2.erode(imgDialation, kernel, iterations=1) виконує морфологічну операцію ерозії на бінарному зображенні imgDialation з використанням заданого ядра kernel. Ерозія використовується для зменшення об'єму об'єктів на зображенні, стираючи частини границь об'єктів. У результаті об'єкти стають меншими та менш помітними.

Завдання 2.3. Вирізання частини зображення

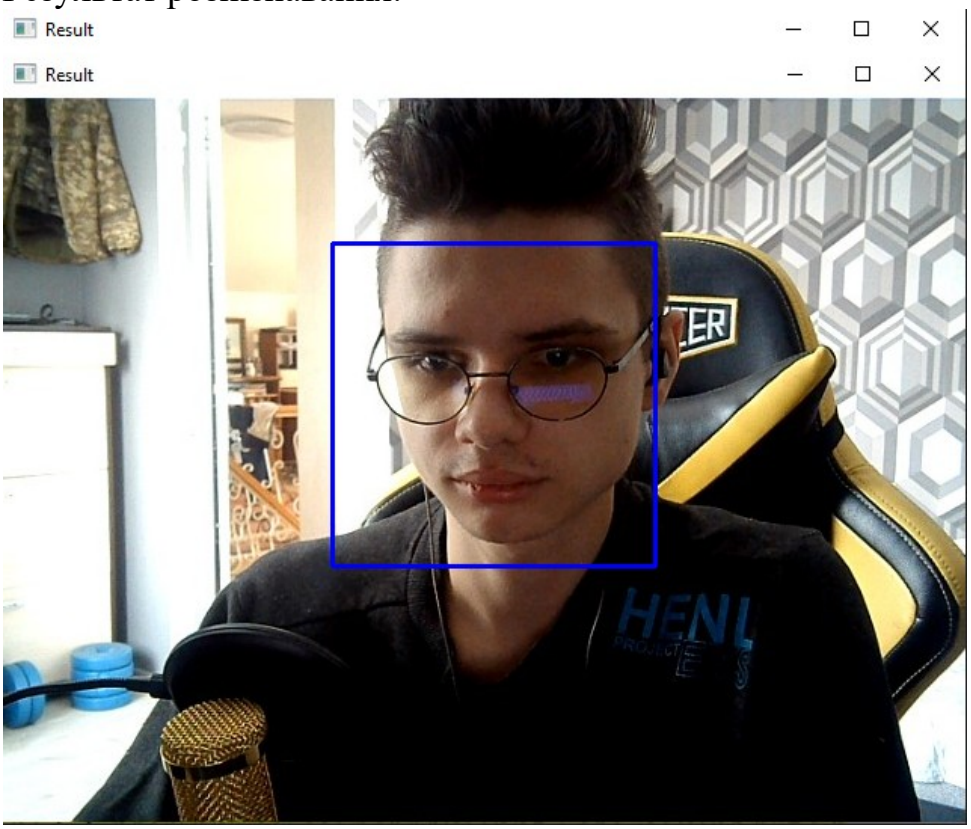
Лістинг програми:

```
import cv2
import numpy as np
img = cv2.imread("Bashmanivskiy.jpg")
print(img.shape)
imgResize = cv2.resize(img,(1000,500))
print(imgResize.shape)
imgCropped = img[106:339,252:395]
cv2.imshow("Image",img)
#cv2.imshow("Image Resize",imgResize)
cv2.imshow("Image Cropped",imgCropped)
cv2.waitKey(0)
```

Обрізане фото:

Завдання 2.4: Розпізнавання обличчя на зображенні

Результат розпізнавання:



Лістинг програми:

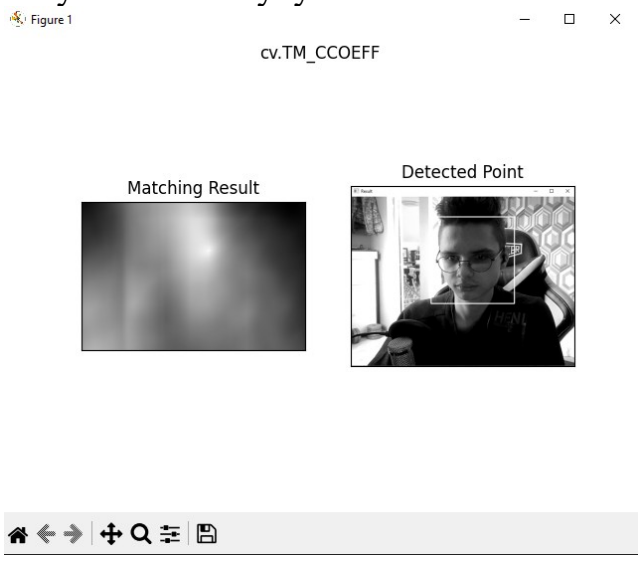
```
import cv2

# Завантажуємо класифікатор для виявлення обличчя (haarcascade_frontalface_default.xml).
faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# Завантажуємо вхідне зображення "Bashmanivskiy.jpg".
img = cv2.imread("Bashmanivskiy.jpg")

# Перетворюємо кольорове зображення в відтінки сірого (зменшуємо обсяг кольорової інформації).
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Використовуючи класифікатор, виявляємо обличчя на зображенні.
# Цей метод `detectMultiScale` шукає обличчя на сірому зображенні з параметрами (масштаб, мінімальні сусіди).
faces = faceCascade.detectMultiScale(imgGray, 1.1, 4)

# Для кожного виявленого обличчя обводимо його прямокутником на оригінальному зображенні.
```

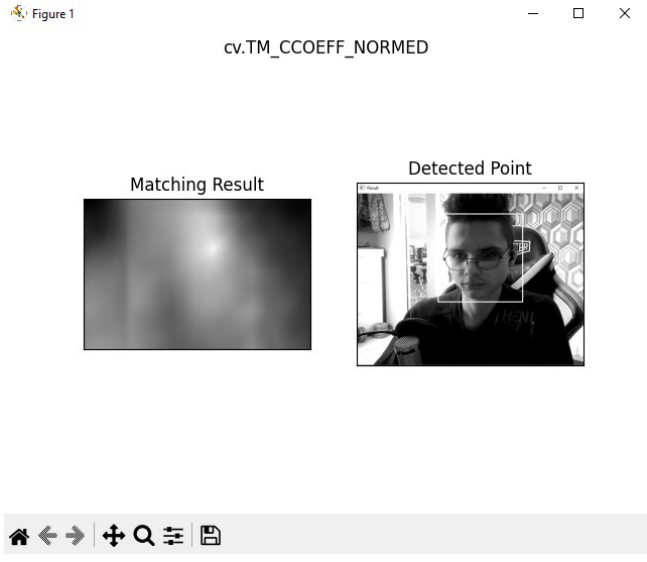
```
for x, y, w, h in faces:
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Відображаємо зображення з виділеними обличчями.
cv2.imshow("Result", img)

# Чекаємо на натискання будь-якої клавіші перед закриттям вікна.
cv2.waitKey(0)
```

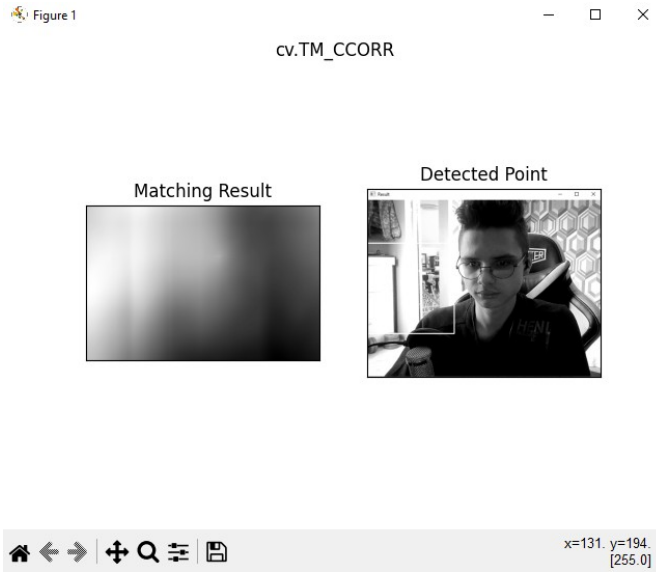
Висновок: Для виявлення об'єктів, використовуються функції Хаара, які є окремими значеннями, отриманими шляхом віднімання суми пікселів під білим прямокутником від суми пікселів під чорним прямокутником. Для зменшення обчислювальної складності функцій Хаара, вони групуються в різні етапи класифікаторів. Замість застосування всіх функцій для кожного вікна, функції застосовуються послідовно, і вікно відкидається, якщо не проходить перший етап. Алгоритм використовує Adaboost для вибору найкращих ознак (функцій Хаара) з великої кількості доступних ознак. Ознаки вибираються на основі їх точності в класифікації облич та необличчя. За допомогою цього методу можна досягти високої точності виявлення об'єктів при значно зменшеній кількості функцій Хаара, які фактично обчислюються.

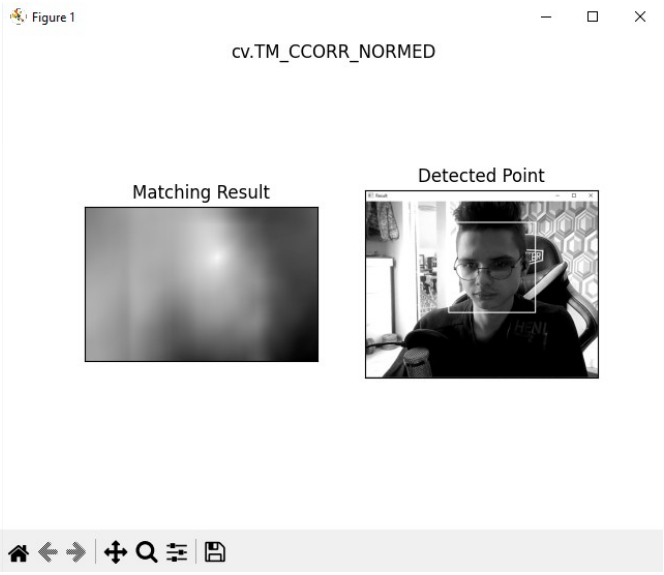
Завдання 2.5: Розпізнавання об’єктів на зображенні за допомогою методів зіставлення шаблонів (Template Matching)

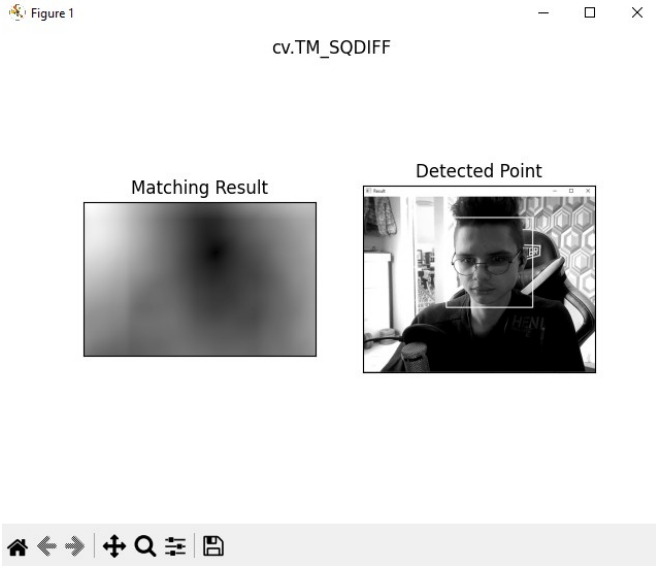
Результати пошуку:

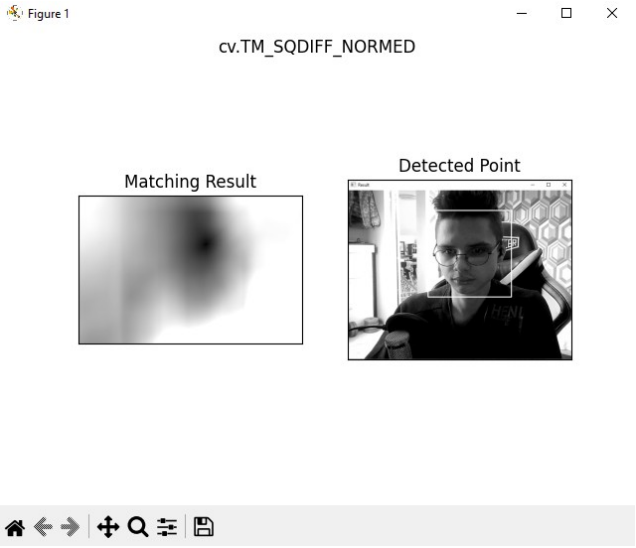




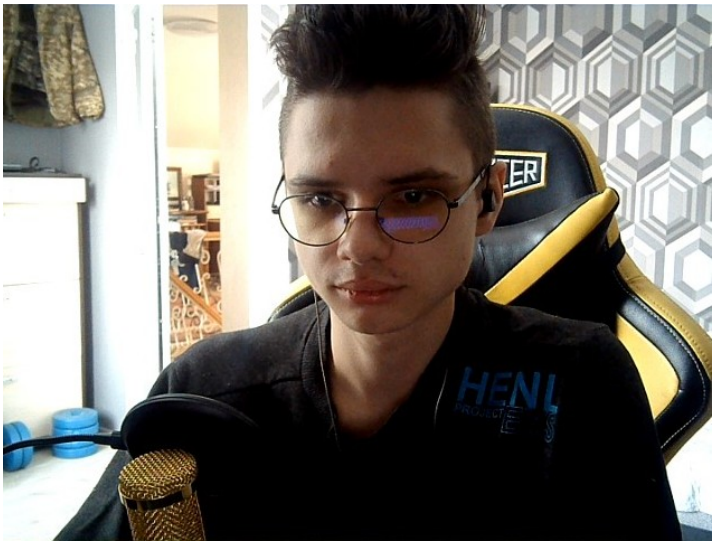








Шаблон та зображення:



Лістинг програми:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
# Завантаження основного та зображення-шаблону
img = cv.imread("Bashmanivskiy.JPG", 0)
img2 = img.copy()
template = cv.imread("Bashmanivskiy_face.JPG", 0)
w, h = template.shape[::-1]
# Список методів порівняння
methods = [
  "cv.TM_CCOEFF",
  "cv.TM_CCOEFF_NORMED",
  "cv.TM_CCORR",
  "cv.TM_CCORR_NORMED",
  "cv.TM_SQDIFF",
  "cv.TM_SQDIFF_NORMED",
]
# Перебираємо методи порівняння
for meth in methods:
  img = img2.copy()
  method = eval(meth)
  # Застосовуємо метод шаблонного виявлення
  res = cv.matchTemplate(img, template, method)
  min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
  # Визначаємо позицію результату в залежності від методу
  if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
    top_left = min_loc
  else:
    top_left = max_loc
  bottom_right = (top_left[0] + w, top_left[1] + h)
  # Малюємо прямокутник навколо виявленого регіону
  cv.rectangle(img, top_left, bottom_right, 255, 2)
  # Відображаємо результати порівняння
  plt.subplot(121), plt.imshow(res, cmap="gray")
  plt.title("Matching Result"), plt.xticks([]), plt.yticks([])
  plt.subplot(122), plt.imshow(img, cmap="gray")
  plt.title("Detected Point"), plt.xticks([]), plt.yticks([])
  plt.suptitle(meth)
  plt.show()
```

Висновок:
cv.TM_CCOEFF: Цей метод використовує коефіцієнт кореляції. Він відзначаєся високою чутливістю до змін масштабу та освітлення, і може виявити подібні образи навіть при їхніх зміщеннях. Якщо ви шукаєте подібність зображень, це добрий метод.
cv.TM_CCOEFF_NORMED: Цей метод є нормалізованою версією попереднього методу. Він також корисний для виявлення подібних образів та вважається найкращим варіантом для багатьох випадків.

cv.TM_CCORR: Цей метод використовує коефіцієнт кореляції. Він підходить для точного виявлення шаблонів, але не є таким стійким до змін освітлення та масштабу.

cv.TM_CCORR_NORMED: Це нормалізована версія попереднього методу. Вона також добре підходить для точного виявлення шаблонів.

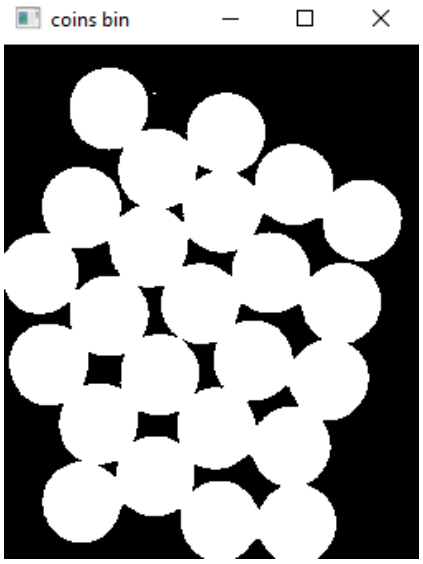
cv.TM_SQDIFF: Цей метод використовує квадрат різниці. Він найменше значення для точних відповідностей та зазвичай використовується для виявлення точних відповідностей.

cv.TM_SQDIFF_NORMED: Це нормалізована версія попереднього методу, яка також відзначається високою чутливістю до точних відповідностей.

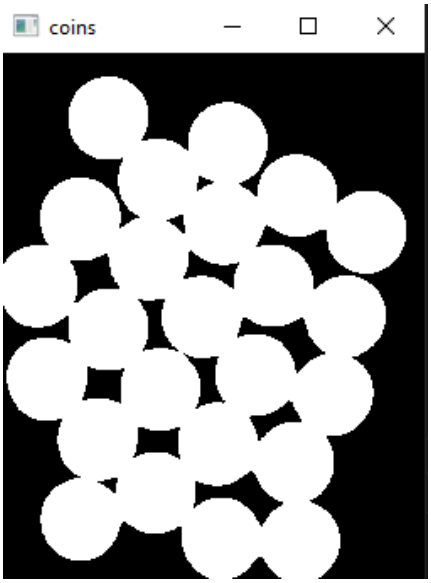
Вибір методу залежить від конкретних вимог завдання. Якщо потрібно точне виявлення шаблону без змін масштабу та освітлення, можна використовувати методи cv.TM_CCORR або cv.TM_CCORR_NORMED. Якщо потрібна більша робастість до змін, cv.TM_CCOEFF або cv.TM_CCOEFF_NORMED можуть бути кращими виборами.

Завдання 2.6: Сегментація зображення алгоритмом водорозподілу
Отримане зображення:

Отримане зображення:



Отримане зображення:



Отримане зображення:

Висновок:
На зображенні, отриманому в результаті коду, монети розділені на окремі сегменти. Це було досягнуто за допомогою алгоритму вододілу, який використовує ієрархію областей, щоб розділити зображення на сегменти.

Алгоритм вододілу працює, створюючи гіпотетичну поверхню вододілу, яка проходить між областями з різними значеннями яскравості. Потім він знаходить точки, де поверхня вододілу стикається з областями. Ці точки називаються вершинами вододілу.

У цьому коді алгоритм вододілу був використаний для сегментації монет на зображенні. Для цього спочатку було створено бінарне зображення, на якому монети були представлені як світлі області на темному фоні. Потім був використаний алгоритм вододілу для сегментації зображення на окремі сегменти.

Результат сегментації показаний на зображенні. Кожна монета має свій власний сегмент, а межі між монетами позначені синім кольором.

Можна було б використовувати більш складний алгоритм вододілу, який враховує форму монет. Це могло б призвести до більш точного сегментування монет.

Завдання 2.7: Сегментація зображення

Лістинг програми:

```
import cv2
import numpy as np
import random


def process_image(image_path):
    frame = cv2.imread(image_path)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    gray_blur = cv2.GaussianBlur(gray, (15, 15), 0)
    thresh = cv2.adaptiveThreshold(
        gray_blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 1
    )

    kernel = np.ones((3, 3), np.uint8)
    closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=1)

    cont_img = closing.copy()
    contours, hierarchy = cv2.findContours(
```

```
    cont_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
  )

  area_groups = {
    "group_1": [],
    "group_2": [],
    "group_3": [],
    "group_4": [],
    "group_5": [],
    "group_6": [],
    "group_7": [],
    "group_8": [],
  }

  for cnt in contours:
    area = cv2.contourArea(cnt)
    if area < 2000 or area > 12000:
      continue

    if len(cnt) < 5:
      continue

    if 2000 < area < 3000:
      color = (255, 0, 0)
      area_groups["group_1"].append(area)
    elif 3000 < area < 4000:
      color = (34, 139, 34)
      area_groups["group_2"].append(area)
    elif 4000 < area < 5000:
      color = (0, 0, 255)
      area_groups["group_3"].append(area)
    elif 5000 < area < 6000:
      color = (255, 255, 0)
      area_groups["group_4"].append(area)
    elif 6000 < area < 7000:
      color = (0, 255, 255)
      area_groups["group_5"].append(area)
    elif 7000 < area < 8000:
      color = (255, 0, 255)
      area_groups["group_6"].append(area)
    elif 8000 < area < 9000:
      color = (128, 0, 0)
      area_groups["group_7"].append(area)
    else:
      color = (0, 128, 0)
      area_groups["group_8"].append(area)
    # color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
    ellipse = cv2.fitEllipse(cnt)
    cv2.ellipse(frame, ellipse, color, 2)

  cv2.imshow("coins bin ", cont_img)
  cv2.imshow("Morphological Closing", closing)
```

```
    cv2.imshow("Adaptive Thresholding", thresh)
    cv2.imshow("Contours", frame)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


if __name__ == "__main__":
    image_path = "coins_3.png"
    process_image(image_path)
```

Результат виконання програми:

Початкове зображення:

Змінив алгоритм обробки зображення в порівнянні з 2.6 завданням. Основні кроки мого алгоритму:
Перетворює зображення кольорового типу на відтінки сірого за допомогою функції cv2.cvtColor.

- Використовуючи фільтрацію Гаусса (cv2.GaussianBlur), створює розмиту версію сірого зображення для покращення обробки.
- Застосовує адаптивне порогування (cv2.adaptiveThreshold) для отримання бінарного зображення, де монети відокремлені від фону.
- Використовує морфологічну операцію закриття (cv2.morphologyEx) для видалення невеликих шумів і підсилення з'єднаних областей.
- Знаходить контури на бінарному зображенні (cv2.findContours) та зберігає їх у змінну contours.
- Ділить знайдені контури на групи відповідно до їхньої площі. Якщо площа контуру потрапляє в певний діапазон, то відповідний колір і площа додаються до відповідної групи (area_groups).
- Рисує еліпси на кольоровому зображенні монет, використовуючи cv2.ellipse. Кожному еліпсу надається колір відповідно до групи, до якої він належить.

- Відображає оброблені зображення на екрані за допомогою cv2.imshow.

Висновок: У завданні я тестував різні методи сегментації зображень — за кольором, за областю, водорозподілом. Найгірше показав себе метод сегментації за кольором, так як на зображенні дуже погане освітлення. У цьому коді був використаний метод сегментації зображення на основі адаптивного порогування (Adaptive Thresholding). Адаптивне порогування визначає поріг для кожного пікселя на основі локальних значень яскравості в околі, що дозволяє ефективно виділити об'єкти на зображенні, незалежно від зміни освітлення. Однак більшість об’єктів на зображенні торкаються один одного, тому це викликало деякі проблеми. Як видно на фото деякі монети об’єднались і мають спільну площу. Також труднощі виникли при визначенні площі, так як монети різної вартості мають однакову площу(або дуже схожу). Проте, на зображенні видно що деякі монети вдалося класифікувати в одну групу. На мою думку це завдання потребує використанню нейронної мережі, яка навчилася розрізняти монети на фотографіях цих монет.