Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №5
з дисципліни
«Технології Computer Vision»
на тему
«Дослідження технологій сегментації та кластеризації цифрових зображень для задач Computer Vision»

Виконала:
Студентка групи ІМ-21
Кривохата Марія Юріївна
Номер у списку групи: 12

Перевірив: Баран Д. Р.

Київ 2024

Мета: дослідити принципи та особливості практичного застосування технологій сегментації та кластеризації цифрових зображень для задач Computer Vision з використанням спеціалізованих програмних бібліотек.

Завдання:

Розробити програмний скрипт, що забезпечує цифрову обробку зображень для розрізнення та ідентифікації обраних об’єктів на цифровому знімку земної поверхні з низькою роздільною здатністю за цифровими зображеннями відкритих джерел даних дистанційного зондування Землі (ДЗЗ) із космосу.

Порядок організаційних дій та функціонал програмного скрипта:

1. Обрати район спостереження та об’єкти ідентифікації – однакові за оперативними та високоточними джерелами даних ДЗЗ – див. табл.

2. Отримати цифрові растрові знімки обраного району земної поверхні з оперативних та високоточних джерел даних ДЗЗ із збереженням їх у файлі відповідного типу.

3. За допомогою програмного скрипта провести кольорову корекцію та / або фільтрацію даних ДЗЗ від оперативних та високоточних джерел відносно об’єкта ідентифікації.

4. Реалізувати програмно кольорову кластерізацію покращених в п.3 зображень об’єкта ідентифікації на даних ДЗЗ від оперативних та високоточних джерел.

5. Здійснити сегментацію кластеризованих в п.4 цифрових зображень від оперативних та високоточних джерел даних ДЗЗ із виділенням контуру об’єкта ідентифікації.

6. Шляхом візуального та / або програмного порівняння контурів обраних об’єктів векторизованих зображень від оперативних та високоточних джерел даних ДЗЗ здійснити ідентифікацію цих об’єктів.

Вимоги та обмеження:

Об’єктами для ідентифікації можуть бути площадні або точкові об’єкти: лісові насадження, вирубки лісів, водойми, площі ерозії поверхні Землі, сільськогосподарські угіддя, посівні площі, будівлі, автівки, техногенні / критичні об’єкти.

Ідентифікація – полягає у встановленні лінгвістичної назви об’єкта та здійснюється за геометрією контура.

Всі процеси обробки повинні бути спрямовані та реалізовані відносно об’єкта ідентифікації.

7-12	1. Оперативні: https://livingatlas2.arcgis.com/landsatexplorer/ 2. Високоточні: https://www.bing.com/maps	Район спостереження – обрати самостійно. Об'єкти ідентифікації – обрати самостійно. Дата оперативних даних – обрати самостійно. Метод і технологія сегментації / кластеризації – повинні забезпечувати можливість розрізнення та ідентифікацію обраних об'єктів спостереження.

II рівень – максимально 9 балів, функціонал скрипта реалізовано у повному обсязі, п.6 технічних вимог (ідентифікація об'єкта) реалізовано шляхом програмного порівняння контурів.

Результат архітектурного проєктування:

Перший варіант програми (без k-means)

Другий варіант програми (із k-means)

Результат роботи програми:

У першому варіанті програми кластеризація та визначення маски здійснюється досить простим та наївним способом за допомогою маски по двох порогах та побітового «і»:

```
def color_clustering(image, lower_water, upper_water):

    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower_water, upper_water)
```

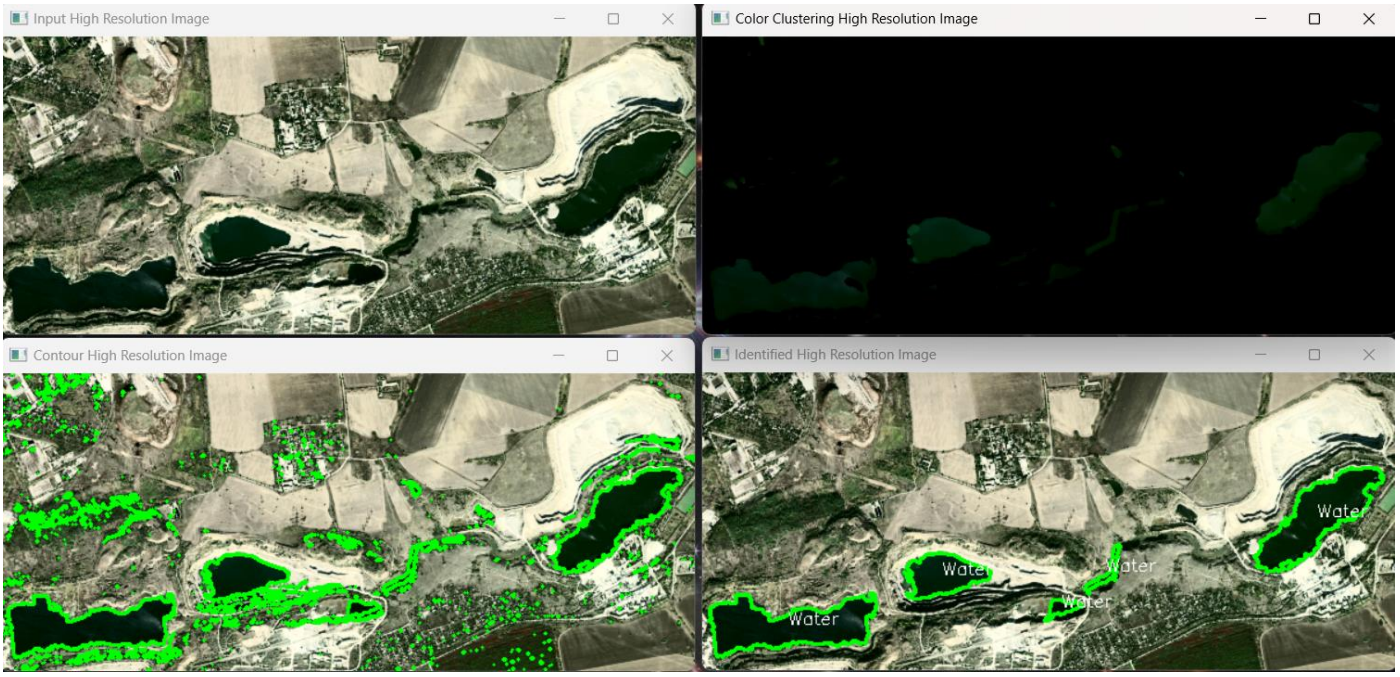
```
    img_bitwise_and = cv2.bitwise_and(image, image, mask=mask)

    # Median blurring
    img_median_blurred = cv2.medianBlur(img_bitwise_and, 5)

    return mask, img_median_blurred
```

Але із належним підбором нижнього та верхнього порогів така програма виявилась доволі дієвою в ідентифікації водойм і на фото з оперативного джерела даних, і на фото із високоточного джерела.

Результати ідентифікації водойм на фото з високоточного джерела:

Зліва зверху знаходиться оброблене (застосована колірна корекція) оригінальне зображення, справа зверху – кластеризоване зображення із накладеною маскою, зліва знизу – зображення із виділеним контуром, справа знизу – кінцеве зображення із ідентифікованими водоймами.

Після визначення кластерів та контуру було безпосередньо ідентифіковано самі водойми, але при цьому у програмі додатковою умовою була чистка «шуму», який утворився при створенні контуру. Його я прибрала за допомогою звичайного обмеження мінімального розміру об’єкта – саме тому невеликі неточності не потрапили на фінальне зображення у ролі ідентифікованих водойм.

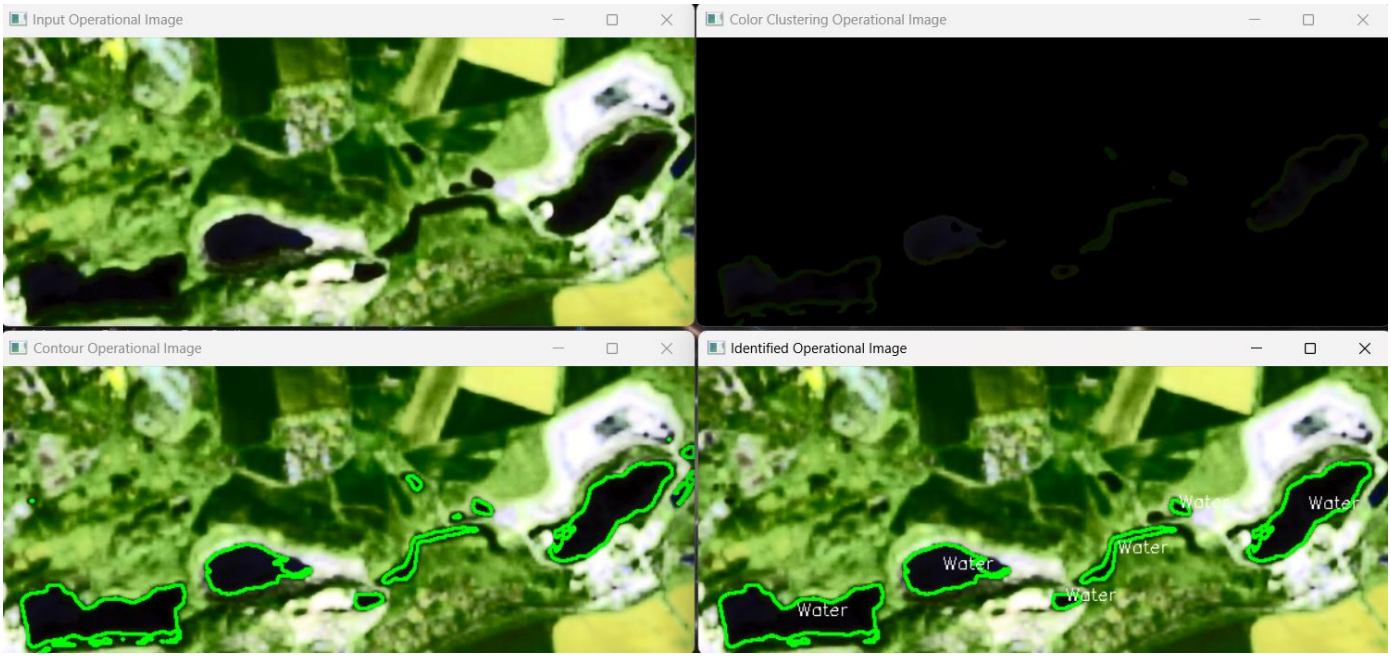


Якщо придивитись до кінцевого зображення, то можна побачити, що програмі навіть вдалось проігнорувати піщані виступи біля водойм і не ідентифікувати їх як воду, хоч вони і мають схожий характерний темний колір. Такий ефект був досягнутий повністю за рахунок намаганням підібрати найбільш точні нижній та верхній ліміті, які визначали цікавий нам проміжок відтінків, насиченості та яскравості.

```
hg_lower_water = np.array([55, 150, 11])
hg_upper_water = np.array([120, 255, 255])
```

Але все ж можна помітити, що невеличкі ділянки води були пропущені – це відбулося якраз через те, що подані ліміти не є інклюзивними до всіх, а всі їхні можливі комбінації фізично не можливо перебрати.

Результати ідентифікації водойм на фото з оперативного джерела:



Тут вже бачимо, що контур (зліва знизу) чіткіший порівняно із контуром на минулому високоточному зображенні. Це пояснюється особливістю картинки – цього разу вона має набагато менше деталей, які можуть перетворитись у «шум» під час роботи програми, але при цьому вона не настільки сильно заблюрена, щоб було складно відрізнити контури. Тут можна побачити, що одна із маленьких водойм тут також була розпізнана, на відміну від минулої програми.

Для ідентифікації водойм на цьому зображенні також використовувались вручну підібрані нижній та верхній поріг кольору, насиченості та яскравості.

```
op_lower_water = np.array([25, 0, 0])
op_upper_water = np.array([240, 255, 30])
```

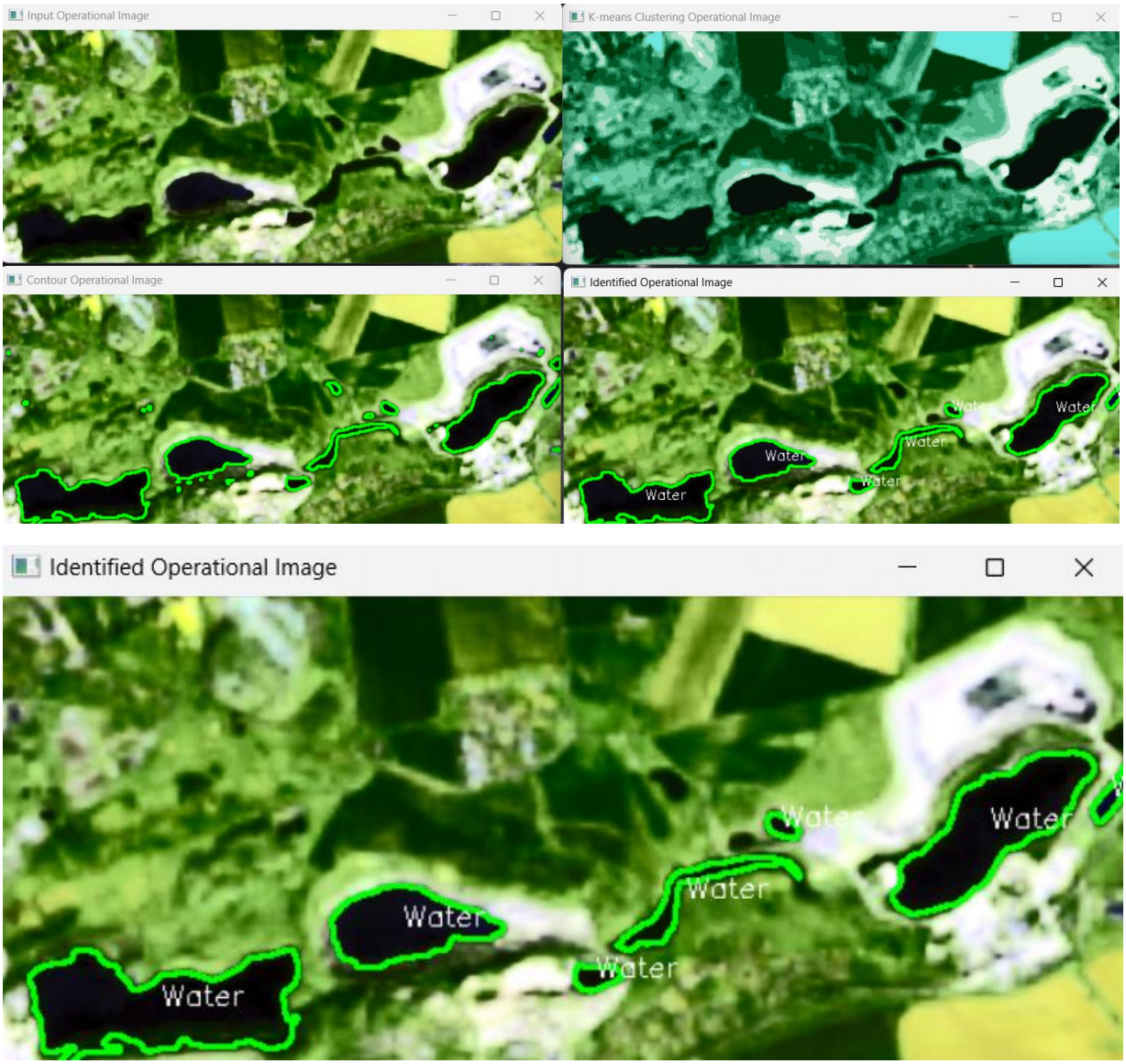
Такий підхід звісно працював, але підбір параметрів вручну виявився досить складною справою. Тому я вирішила спробувати зробити той самий функціонал, але **вже за допомогою кластеризації із k-means.**

Для максимальної точності я обрала достатньо велику кількість кластерів – 10

```
operational_clustered, operational_labels, operational_centers =
kmeans_clustering(operational_image, K=10)
```

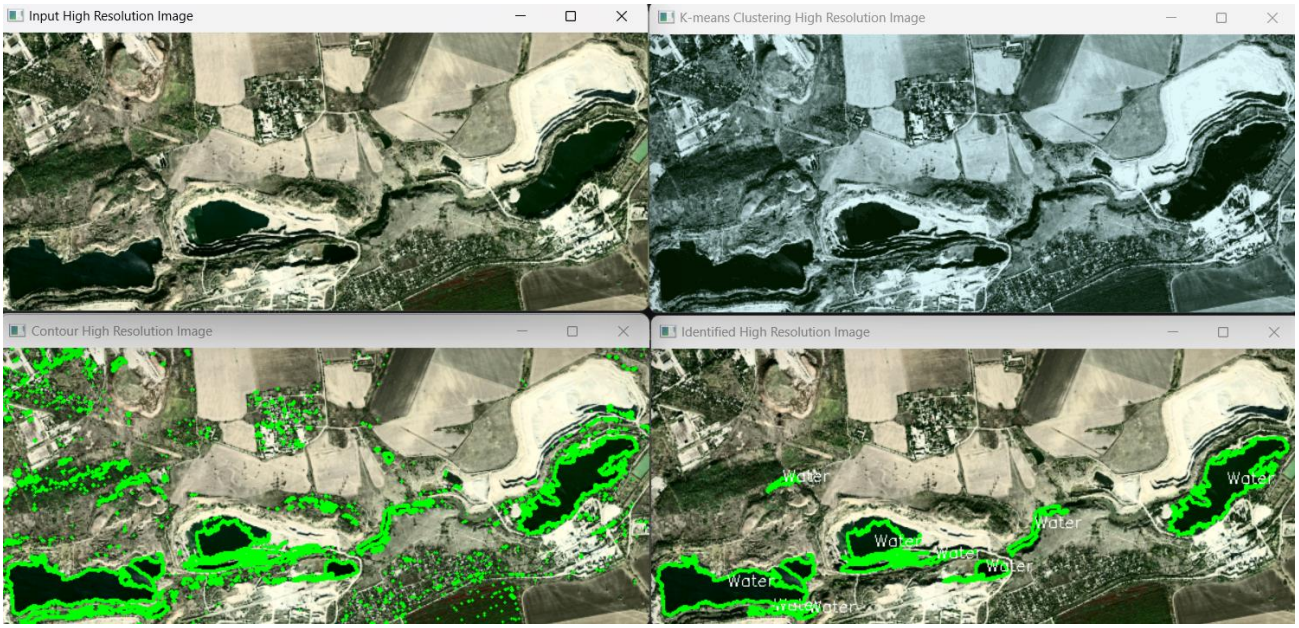
Як можна побачити на картинках знизу, ідентифікація водойм із **зображення з операційного джерела** тепер відбувається навіть ефективніше! Тепер ідентифікується навіть невеличка водойма справа.

Хочу також зазначити, що такий спосіб дає не тільки гарніший результат, але й набагато простіший у реалізації. Звісно, тут теж був елемент «експерименту», де потрібно було підбирати деякі параметри, але в загальному це все виглядає більш визначено та зрозуміло ніж у минулому способі.



Але якщо ми такий самий спосіб застосуємо до зображення із високоточного джерела, то бачимо, що результат тут вже менш точний, ніж був у попередньому способі.

На мою думку, це тому, що цей варіант все ж більш чутливий до відтінків та шумів. Тому можна зробити висновок, **що для зображення з високоточних джерел більш ефективним був попередній метод ідентифікації, НЕ за способом k-means.**



Програмний код, що забезпечує отримання результату:

Базова версія програми (без k-means)

```
import cv2
import numpy as np
import imutils


def color_correction(image_path):
    image = cv2.imread(image_path)
    image = imutils.resize(image, width=600)
    # color correction - simple histogram equalization in LAB color space
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    l = cv2.equalizeHist(l)
    lab = cv2.merge((l, a, b))
```

```
    corrected_image = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
    return corrected_image


def color_clustering(image, lower_water, upper_water):

    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower_water, upper_water)

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

    # Median blurring
    img_median_blurred = cv2.medianBlur(img_bitwise_and, 5)

    return mask, img_median_blurred


def find_contours(image, mask):
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contour_image = image.copy()
    cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
    return contour_image, contours


def identify_objects(image, contours):
    identified_image = image.copy()
    for i, contour in enumerate(contours):
        area = cv2.contourArea(contour)
        if area > 100:  # Filter small areas
            cv2.drawContours(identified_image, [contour], -1, (0, 255, 0), 2)
            M = cv2.moments(contour)
            if M['m00'] != 0:
                cX = int(M['m10'] / M['m00'])
                cY = int(M['m01'] / M['m00'])
                cv2.putText(identified_image, 'Water', (cX, cY),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    return identified_image


def show_results(original, segment, contour, identified, img_type):
    original_text = 'Color corrected Input ' + img_type + ' Image'
    segment_text = 'Color Clustering ' + img_type + ' Image'
    contour_text = 'Contour ' + img_type + ' Image'
    identified_text = 'Identified ' + img_type + ' Image'

    cv2.imshow(original_text, original)
    cv2.imshow(segment_text, segment)
    cv2.imshow(contour_text, contour)
    cv2.imshow(identified_text, identified)


def process_images(operational_image_path, high_res_image_path):
    # Color correction
    operational_image = color_correction(operational_image_path)
    high_res_image = color_correction(high_res_image_path)

    hg_lower_water = np.array([55, 150, 11])
    hg_upper_water = np.array([120, 255, 255])
```

```
    op_lower_water = np.array([25, 0, 0])
    op_upper_water = np.array([240, 255, 30])

    # Color clustering
    operational_mask, op_img_median_blurred =
color_clustering(operational_image, op_lower_water, op_upper_water)
    high_res_mask, hg_img_median_blurred = color_clustering(high_res_image,
hg_lower_water, hg_upper_water)

    # Contours
    operational_contour_image, operational_contours =
find_contours(operational_image, operational_mask)
    high_res_contour_image, high_res_contours = find_contours(high_res_image,
high_res_mask)

    # Identifying objects
    operational_ident_image = identify_objects(operational_image,
operational_contours)
    high_res_ident_image = identify_objects(high_res_image, high_res_contours)

    show_results(operational_image, op_img_median_blurred,
operational_contour_image, operational_ident_image, 'Operational')
    show_results(high_res_image, hg_img_median_blurred, high_res_contour_image,
high_res_ident_image, 'High Resolution')

    cv2.waitKey(0)
    cv2.destroyAllWindows()


if __name__ == "__main__":
    operational_image_path = 'landsat_picture.png'
    high_res_image_path = 'bing_picture.png'
    process_images(operational_image_path, high_res_image_path)
```

Більш просунута версія програми (із k-means)

```
import cv2
import numpy as np
import imutils


def color_correction(image_path):
    image = cv2.imread(image_path)
    image = imutils.resize(image, width=600)
    # color correction - simple histogram equalization in LAB color space
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    l = cv2.equalizeHist(l)
    lab = cv2.merge((l, a, b))
    corrected_image = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
    return corrected_image


def kmeans_clustering(image, K=2):
    img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    twoDimage = img.reshape((-1, 3))
    twoDimage = np.float32(twoDimage)
```

```
    # defining criteria and applying kmeans
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    ret, labels, centers = cv2.kmeans(twoDimage, K, None, criteria, 10,
cv2.KMEANS_PP_CENTERS)

    centers = np.uint8(centers)
    clustered_image = centers[labels.flatten()]
    clustered_image = clustered_image.reshape((img.shape))

    return clustered_image, labels, centers


def create_binary_mask(clustered_image, labels, cluster_index):
    mask = np.zeros(clustered_image.shape[:2], dtype=np.uint8)
    mask[labels.reshape(clustered_image.shape[:2]) == cluster_index] = 255
    return mask


def find_water_cluster(centers, target_color):
    # calculating the Euclidean distance between each cluster center and the
target color
    distances = np.linalg.norm(centers - np.array(target_color), axis=1)
    water_cluster = np.argmin(distances)  # the cluster with the smallest
distance
    print(f"Water cluster index: {water_cluster}, center color:
{centers[water_cluster]}")
    return water_cluster


def find_contours(image, mask):
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contour_image = image.copy()
    cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
    return contour_image, contours


def identify_objects(image, contours):
    identified_image = image.copy()
    for i, contour in enumerate(contours):
        area = cv2.contourArea(contour)
        if area > 100:  # Filter small areas
            cv2.drawContours(identified_image, [contour], -1, (0, 255, 0), 2)
            M = cv2.moments(contour)
            if M['m00'] != 0:
                cX = int(M['m10'] / M['m00'])
                cY = int(M['m01'] / M['m00'])
                cv2.putText(identified_image, 'Water', (cX, cY),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    return identified_image


def show_results(original, clustered, contour, identified, img_type):
    original_text = 'Color corrected Input ' + img_type + ' Image'
    segment_text = 'K-means Clustering ' + img_type + ' Image'
    contour_text = 'Contour ' + img_type + ' Image'
    identified_text = 'Identified ' + img_type + ' Image'
```

```
    cv2.imshow(original_text, original)
    cv2.imshow(segment_text, clustered)
    cv2.imshow(contour_text, contour)
    cv2.imshow(identified_text, identified)


def process_images(operational_image_path, high_res_image_path):
    # Color correction
    operational_image = color_correction(operational_image_path)
    high_res_image = color_correction(high_res_image_path)

    # defining water color (BGR for #08100b)
    water_color = (11, 16, 8)

    # K-means clustering
    operational_clustered, operational_labels, operational_centers =
kmeans_clustering(operational_image, K=10)
    high_res_clustered, high_res_labels, high_res_centers =
kmeans_clustering(high_res_image, K=20)

    # Clustering
    operational_water_cluster = find_water_cluster(operational_centers,
water_color)
    high_res_water_cluster = find_water_cluster(high_res_centers, water_color)

    # Binary masks for the water clusters
    operational_mask = create_binary_mask(operational_clustered,
operational_labels, cluster_index=operational_water_cluster)
    high_res_mask = create_binary_mask(high_res_clustered, high_res_labels,
cluster_index=high_res_water_cluster)

    # Contours
    operational_contour_image, operational_contours =
find_contours(operational_image, operational_mask)
    high_res_contour_image, high_res_contours = find_contours(high_res_image,
high_res_mask)

    # Identifying objects
    operational_ident_image = identify_objects(operational_image,
operational_contours)
    high_res_ident_image = identify_objects(high_res_image, high_res_contours)

    show_results(operational_image, operational_clustered,
operational_contour_image, operational_ident_image, 'Operational')
    show_results(high_res_image, high_res_clustered, high_res_contour_image,
high_res_ident_image, 'High Resolution')

    cv2.waitKey(0)
    cv2.destroyAllWindows()


if __name__ == "__main__":
    operational_image_path = 'landsat_picture.png'
    high_res_image_path = 'bing_picture.png'
    process_images(operational_image_path, high_res_image_path)
```

Висновки: Виконавши цю лабораторну роботу, я навчилась працювати із кластеризацією за кольором, k-means, сегментацією та ідентифікацією об’єктів. Я написала дві версії програми,

що ідентифікує водойми на супутникових знімках із різних джерел. Методом багаточисленних експериментів я дійшла до висновку, що в моєму випадку для оперативних джерел найефективнішим способом кластеризації виявився k-means та вже подальша обробка для ідентифікації, а для високоточних джерел найкраще спрацювала звичайна колірна кластеризація і подальша ідентична обробка як і в способі з k-means.