به نام خدا

مهدی فیروزبخت

تمرین سری اول

درس بینایی کامپیوتر

۱.

ابتدا با توجه به صورت سوال ، تصویر MeeseeksHQ را بار گذاری مینماییم. طبق صورت سوال این تصویر را به صورت رنگی بار گذاری کرده ایم و در قدم بعدی آن را به نمایش گذاشته ایم.

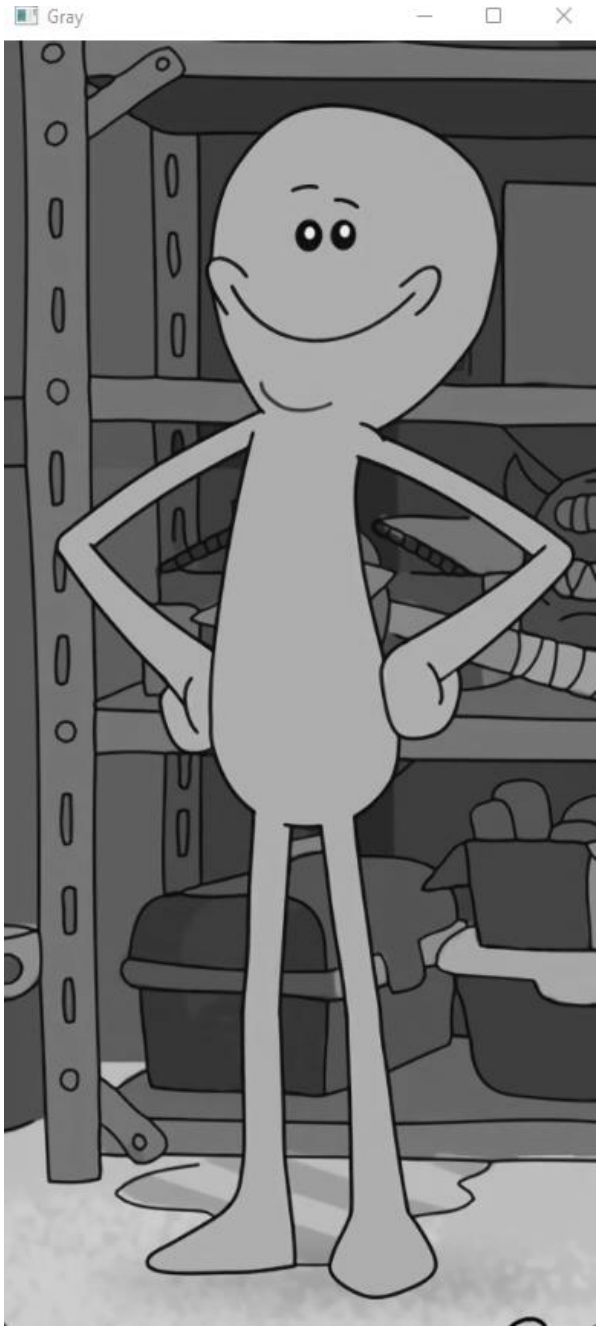
```
# importing the opencv module
import cv2

# using imread('path') and 1 denotes read as  color image
src = cv2.imread('MeeseeksHQ.png', 1)

# This is using for display the image
cv2.imshow('BGR', src)
cv2.waitKey()  # This is necessary to be required so that the image doesn't close immediately.
```

در قدم بعدی با استفاده از تابع cvtColor و ورودی cv2.COLOR_BGR2GRAY تصویر را به سطح خاکستری میبریم.

```
# Using cv2.cvtColor() method
# Using cv2.COLOR_BGR2GRAY color space
# conversion code
src2gry = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

# Displaying the image
cv2.imshow('Gray', src2gry)

cv2.waitKey()  # This is necessary to be required so that the image doesn't close immediately.
# It will run continuously until the key press.
cv2.destroyAllWindows()
```

در ادامه نیز تصاویر به دست آمده را قرار داده ایم :

تصویر اصلی



تصویر سطح خاکستری

.۲

در ابتدا با توجه به خواسته مسئله ، تصویر Mr. Meeseeks را بار گذاری کرده و آن را نمایش میدهیم که به صورت زیر میباشد:

```
# importing the opencv module
import cv2
import numpy as np

# using imread('path') and 1 denotes read as  color image
image = cv2.imread('mr meeskees.png', 1)

# This is using for display the image
cv2.imshow('BGR', image)
cv2.waitKey()  # This is necessary to be required so that the image doesn't close immediately.
# It will run continuously until the key press.
# cv2.destroyAllWindows()
```

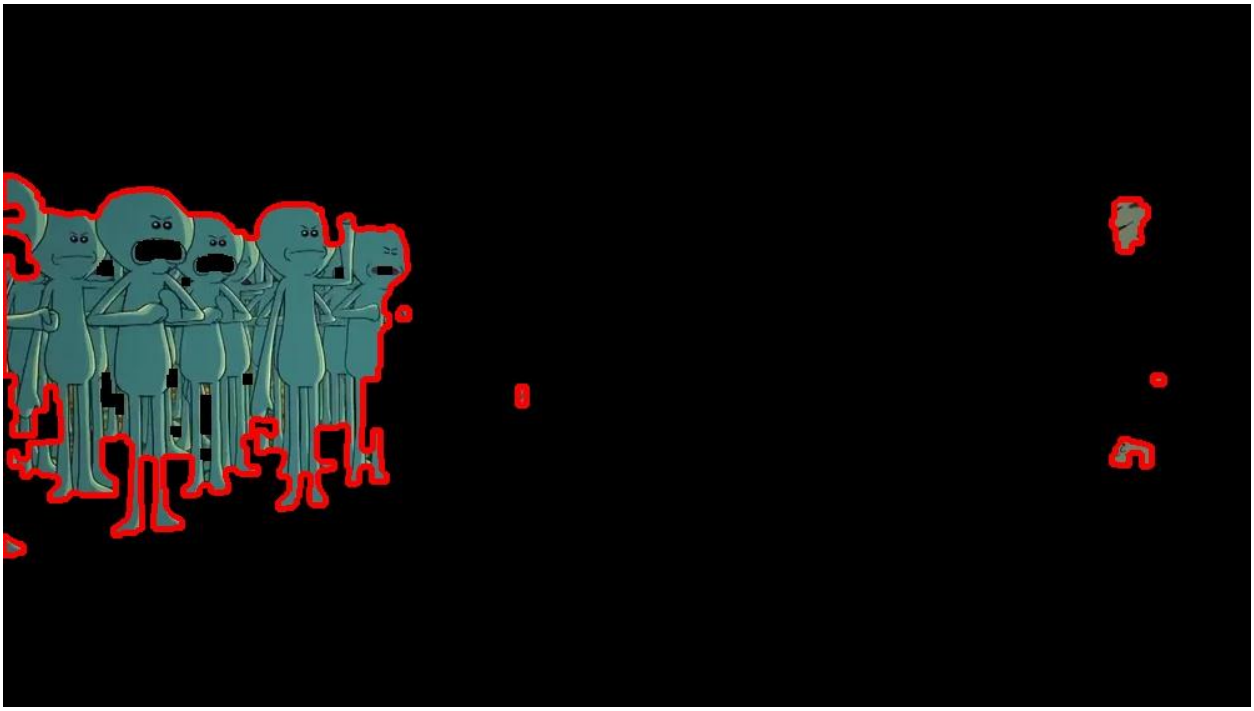
در ادامه برای تشخیص شخصیت مورد نظر ، از ۲ روش کار خود را انجام میدهیم . ابتدا روش اول:

در این روش از تصاویر به صورت BGR استفاده میکنیم. برای اینکار ابتدا محدوده پیکسلی که شامل رنگ Mr. Meeseeks است را در نظر گرفته و پیکسل هایی که در این محدوده هستند را روشن کرده (یعنی رنگ آن را بدون تغییر قرار داده) و پیکسل هایی که در این محدوده نیستند را خاموش میکنیم (یعنی رنگ آن را به سیاه تبدیل میکنیم). سپس یک ماسک از این محدوده ایجاد میکنیم.

برای پیدا کردن محدوده ها از روش آزمون خطا استفاده کرده ایم . در ماسک می بینیم که نویز غیر ضروری زیادی وجود دارد. بنابراین باید آن را حذف کنیم تا نتیجه بهتری بگیریم. برای اینکار یک کرنل ۷*۷ تعریف کرده و آن را بر تصویر اعمال میکنیم. سپس ماسک را بر تصویر اصلی اعمال میکنیم. در انتها نیز با استفاده از ماسک بدست آمده کانتوری را دور شخصیت های مورد نظر میکشیم. برای این کار از کد زیر استفاده میکنیم :

```
# first method
# define the boundaries
lower = np.array([103, 80, 50])
upper = np.array([150, 170, 120])
# find the colors within the specified boundaries and apply
# the mask
mask = cv2.inRange(img, lower, upper)

# define kernel size
kernel = np.ones((7, 7), np.uint8)
# Remove unnecessary noise from mask
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

segmented_img = cv2.bitwise_and(img, img, mask=mask)
# Find contours from the mask
contours, hierarchy = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
output = cv2.drawContours(segmented_img, contours, -1, (0, 0, 255), 3)
# Showing the output
# show the images
cv2.imshow("images", output)
cv2.waitKey(0)
```

تصویر به دست آمده از این روش به صورت زیر میباشد :



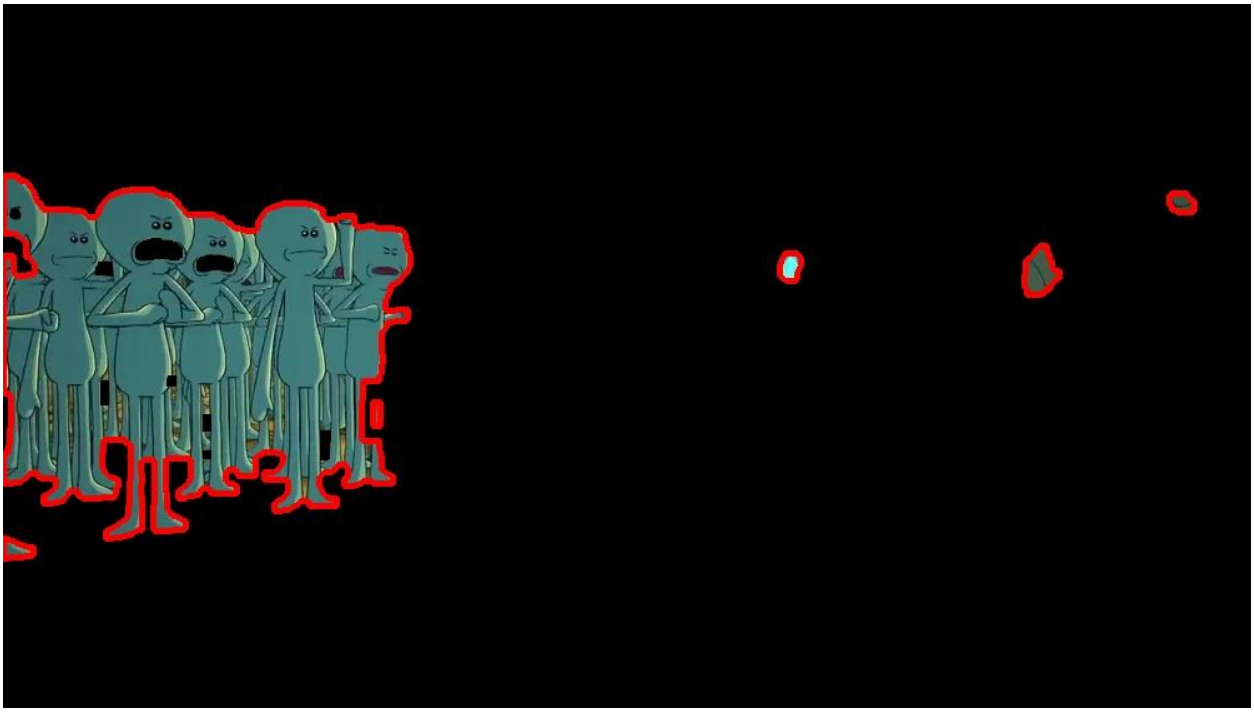
در این حالت مشاهده میکنیم که تصاویر شخصیت های مورد نظر واضح هستند هر چند به طور کل خطا در تشخیص پیکسل ها نیز وجود دارد.

در روش دوم از فضای رنگی HSV استفاده میشود زیرا فضای رنگی HSV زمانی مفید است که با اطلاعات رنگ کار می کنیم. HSV مخفف HUE، SATURATION و VALUE (یا روشنایی) است. این یک فضای رنگی استوانه ای است.

برای اینکار ابتدا تصاویر را به فضای HSV می‌آوریم. سپس یک محدوده جدید در این فضا تعیین میکنیم. تشکیل محدوده در این فضا نیز با آزمون و خطا پیدا شده است .سپس مراحل ساخت ماسک ، کرنل و اعمال آنها به تصویر همانند حالت قبلی میباشد :

```
# second method
# convert to hsv colorspace
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# lower bound and upper bound for Green color
lower_bound = np.array([50, 60, 70])
upper_bound = np.array([100, 255, 255])
# find the colors within the boundaries
mask = cv2.inRange(hsv, lower_bound, upper_bound)
# define kernel size
kernel = np.ones((7, 7), np.uint8)
# Remove unnecessary noise from mask
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
# Segment only the detected region
segmented_img = cv2.bitwise_and(img, img, mask=mask)
# Find contours from the mask
contours, hierarchy = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
output = cv2.drawContours(segmented_img, contours, -1, (0, 0, 255), 3)
# Showing the output
cv2.imshow("Output", output)
cv2.waitKey(0)

# Filename
filename = 'second_method.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, output)
```

در این حالت تصویر به دست آمده به صورت زیر میباشد :



در این حالت نیز مشاهده میشود که شخصیت مورد نظر واضح است هر چند خطایی نیز در تصویر قابل مشاهده است.

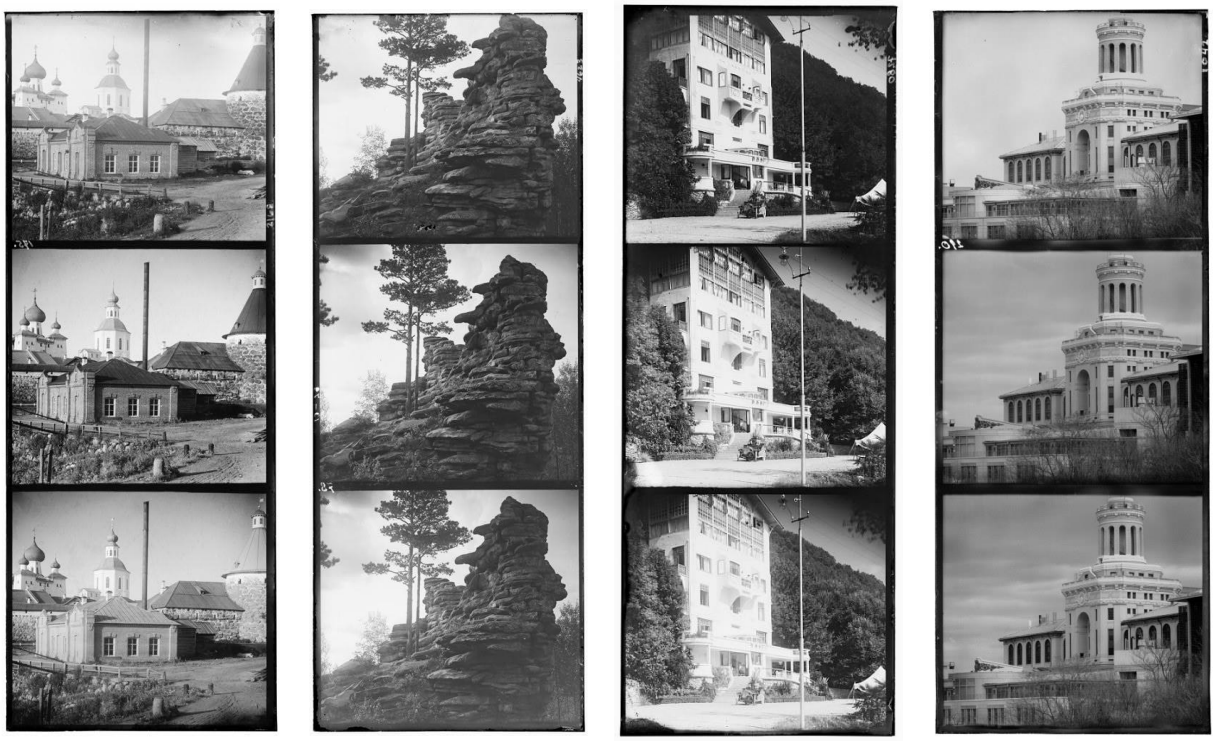
در مرحله آخر نیز کانتور را بر روی تصویر اصلی قرار میدهیم تا نتیجه را در تصویر اصلی نیز مشاهده کنیم :

.۳

ابتدا تصاویر مد نظر را بارگذاری کرده و درون آرایه ای قرار میدهیم :

```python
import cv2
import numpy as np

# using imread('path') and 1 denotes read as  color image
image01 = cv2.imread('01.jpg', 0)
image02 = cv2.imread('02.jpg', 0)
image03 = cv2.imread('03.jpg', 0)
image04 = cv2.imread('04.jpg', 0)

images = [image01, image02, image03, image04]


def show_images(images):
    for image in images:
        cv2.imshow("image", image)
        cv2.waitKey()


show_images(images)
```

که تصاویر به صورت زیر میباشند :



سپس در ادامه به سوالات هر بخش پاسخ داده شده است :

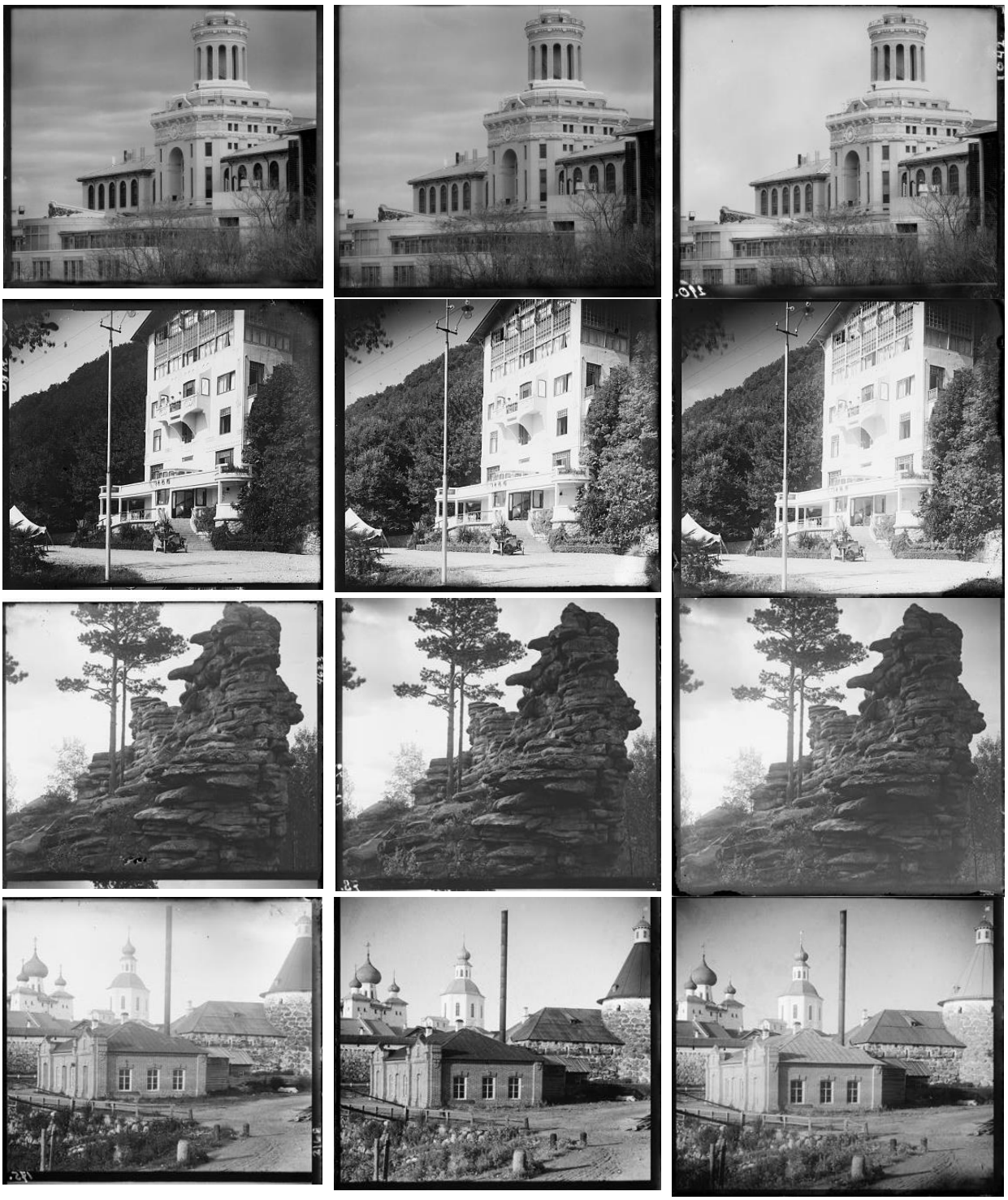
الف . برای ساخت این تابع نیاز است ابتدا کادر دور تصاویر را حذف نماییم با استفاده از تابع زیر این کار را انجام میدهیم . در این کد از هر طرف پیکسل ها را بررسی مینماییم و اگر سفید بود به معنای کادر است و آن را حذف مینماییم . تمام این کد ها را در تابع show_filters قرار میدهیم :

```
def show_filters(item):
    x = item.shape[0]
    y = item.shape[1]

    margins = [0, 0, 0, 0]

    # 0 for left , 1 for right , 2 for top and 3 for bottom
    for margin in range(4):
        if margin < 2:
            plc = np.squeeze(item[int(x / 2):int((x / 2) + 1), :])
        else:
            plc = np.squeeze(np.transpose(item[:, int(y / 2):int((y / 2) + 1)]))

        if margin % 2 != 0:
            plc = np.flip(plc)

        for pixel in plc:
            if pixel > 245:
                margins[margin] += 1
            else:
                break

    new_image = item[int(margins[2] + 2): int(item.shape[0] - margins[3] - 2),
                int(margins[0] + 2): int(item.shape[1] - margins[1] - 2)]
```

سپس تصاویر را با کد زیر به ۳ قسمت تقسیم مینماییم :

```
new_x = new_image.shape[0]
new_y = new_image.shape[1]

cropped_x = new_x / 3
cropped_y = new_y

cv2.imshow("", new_image)
cv2.waitKey()

image1 = new_image[0: int(cropped_x), 0: int(cropped_y)]
image2 = new_image[int(cropped_x): 2 * int(cropped_x), 0: int(cropped_y)]
image3 = new_image[2 * int(cropped_x): 3 * int(cropped_x), 0: int(cropped_y)]

cv2.imshow("",image1)
cv2.waitKey()
cv2.imshow("",image2)
cv2.waitKey()
cv2.imshow("",image3)
cv2.waitKey()

return image1, image2, image3
```

تصاویر به صورت زیر تبدیل میشوند :

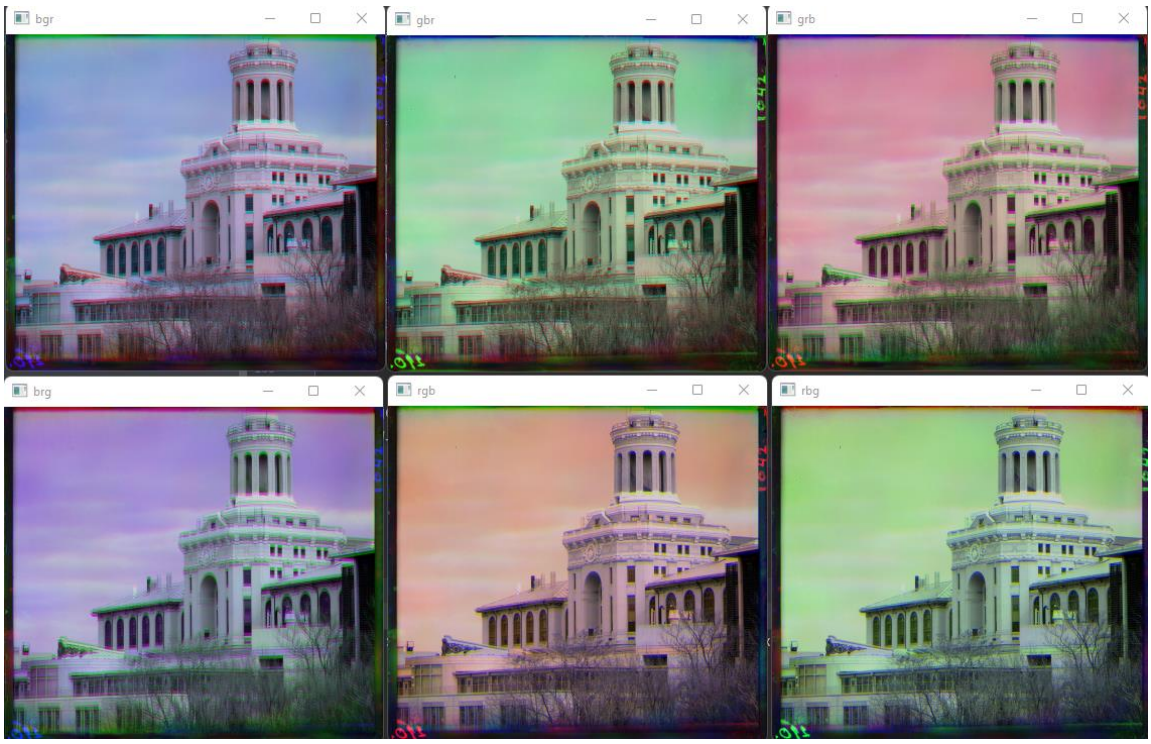


ب. برای این قسمت تصاویر را در حالت های مختلف به روی یکدیگر قرار میدهیم تا بهترین حالت ممکن را به دست آوریم. در این قسمت برای تصویر بالا نام b برای تصویر دوم نام g و برای تصویر آخر نام r را انتخاب میکنیم تا با بررسی آن ها بهترین تصویر را به دست آوریم. کد این بخش به صورت زیر میباشد :

```
def show_colorful_image(image):
    b = image[0]
    g = image[1]
    r = image[2]
    img = cv2.merge((b, g, r))
    cv2.imshow("bgr", img)
    cv2.waitKey()

    img = cv2.merge((g, b, r))
    cv2.imshow("gbr", img)
    cv2.waitKey()

    img = cv2.merge((g, r, b))
    cv2.imshow("grb", img)
    cv2.waitKey()

    img = cv2.merge((b, r, g))
    cv2.imshow("brg", img)
    cv2.waitKey()

    img = cv2.merge((r, g, b))
    cv2.imshow("rgb", img)
    cv2.waitKey()

    img = cv2.merge((r, b, g))
    cv2.imshow("rbg", img)
    cv2.waitKey()


for image in seperated_images:
    show_colorful_image(image)
```

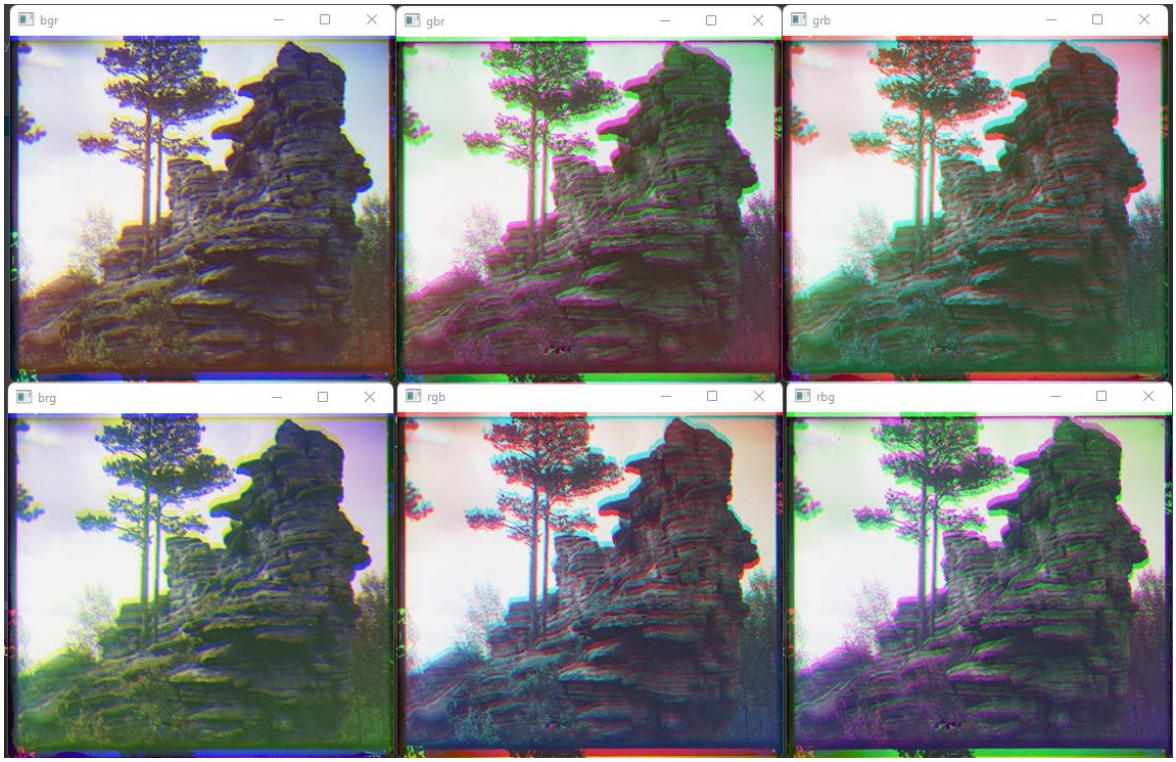
ترکیب این تصاویر برای تصاویر مختلف به صورت زیر میباشد :

تصویر ۱ :



تصویر ۲ :

تصویر ۳ :



تصویر ۴ :

با توجه به تصاویر به دست آمده میتوان فهمید که تصاویر bgr بهترین تصاویر است . با توجه به کتابخانه opencv که تصاویر در آن به صورت فضایی BGR هستند پس تصویر بالا به عنوان تصویر کانال Blue تصویر وسط به عنوان ورودی Green و تصویر پایین به عنوان ورودی Red میباشد.

پس تصاویر زیر به صورت تصویر اصلی هستند :

ج. برای این سوال نیازمند این هستیم که چند تابع بنویسیم. ابتدا تابع align_images که عکس ها را گرفته و به تابع align وارد میکند. در این تابع کانال آبی را به صورت ثابت در نظر گرفته و ۲ کانال دیگر را برای جابه‌جایی انتخاب میکنیم. همچنین از time.time() برای دریافت زمان و بررسی آن استفاده میکنیم . تابع مدنظر به صورت زیر میباشد :

```
def align_images():
    for j in range(2):
        aligend = []
        for i, item in enumerate(colorful_image):
            start_time = time.time()
            ag, g_shift = align(item[:, :, 1], item[:, :, 0], methods[j])
            ar, r_shift = align(item[:, :, 2], item[:, :, 0], methods[j])

            img = cv2.merge((item[:, :, 0], ag, ar))
            end_time = time.time() - start_time
            aligend.append([str(i), img, end_time])

            # Filename
            # filename = 'image'+str(i)+'method'+methods[j]+'.jpg'
            #
            # # Using cv2.imwrite() method
            # # Saving the image
            # cv2.imwrite(filename, img)

            # cv2.imshow(str(i), img)
            # cv2.waitKey()
        aligend_images.append([methods[j], aligend])


align_images()
```

در این تابع یک تابع با نام align آورده شده است که کار اصلی را انجام میدهد. این تابع به صورت زیر میباشد :

```
def align(img1, img2, method, off_x=(-15, 15), off_y=(-15, 15)):
    best_score = -1
    best_shift = [0, 0]

    # loop over all the different displacement permutations
    for i in range(off_x[0], off_x[1] + 1):
        for j in range(off_y[0], off_y[1] + 1):
            temp_score = score(np.roll(img1, (i, j), (0, 1)), img2, method)
            if temp_score > best_score:
                best_score = temp_score
                best_shift = [i, j]

    # return the best displaced image along with the displacement vector
    return np.roll(img1, best_shift, (0, 1)), np.array(best_shift)
```

در این تابع ۲ فیلتر (یک فیلتر ثابت و یک فیلتر متحرک) و همچنین بازه جابه‌جایی به عنوان ورودی دریافت میشود. سپس در این بازه ، کانال متحرک را جابه‌جا میکنیم و ۲ کانال جدید را به

یک تابع امتیازدهی میدهیم و در این کانال بسته به روش امتیازی به نسبت این ۲ کانال داده میشود. سپس بهترین امتیاز را پیدا کرده و درمیابیم که چه حدی از کانال جابه‌جا شوند بهترین امتیاز را میگیرند پس آن کانال را به این تعداد واحد جابه‌جا میکنیم. سپس همین کار را برای کانال دوم انجام میدهیم و در انتها نتیجه را به نمایش میگذاریم. تابع امتیاز دهی به صورت زیر میباشد:

```
def score(im1, im2, method):
    if method == 'SSD':
        return -np.sum(np.sum((im1 - im2) ** 2))
    elif method == 'NCC':
        im1 = np.ndarray.flatten(im1)
        im2 = np.ndarray.flatten(im2)
        return np.dot(im1 / np.linalg.norm(im1), im2 / np.linalg.norm(im2))
```

در این روش از ۲ روش برای امتیازدهی استفاده شده است که یک مدل حالت SSD میباشد و روش دم به صورت NCC میباشد.

تصاویر به دست آمده در هر روش به صورت زیر میباشد. ابتدا روش SSD را نمایش میدهیم :




سپس روش NCC :

در مورد سرعت روش و پیچیدگی زمانی آن با توجه به زمان به دست آمده به صورت زیر میباشد :

```
['SSD', '0', 0.733722448348999]
['SSD', '1', 0.7031199932098389]
['SSD', '2', 0.734424352645874]
['SSD', '3', 0.7187483310699463]
['NCC', '0', 6.015578985214233]
['NCC', '1', 5.453125715255737]
['NCC', '2', 4.250349998474121]
['NCC', '3', 5.294970989227295]
```

در این روش ۲ حلقه داریم که هر کدام در ۳۰ جهت حرکت میکنند پس در کل ۹۰۰ بار چرخش در آن داریم. در ادامه درون تابع امتیازدهی در ۲ روش تفاوت وجود دارد. در روش NCC پیچیده‌ترین و طولانی ترین مرحله ضرب داخلی میباشد.

در این حالت روش دوم یعنی SSD بسیار سریعتر از روش NCC است.

روش سومی نیز وجود دارد که میتوانیم از هرم وضوح استفاده کرد که کد آن درون تابع pyramid_aling قرار داده شده است :

```
def pyramid_aling():
    for j in range(2):
        aligend = []
        for i, item in enumerate(colorful_image):
            start_time = time.time()
            ag, g_shift = pyramid(item[:, :, 1], item[:, :, 0], methods[j])
            ar, r_shift = pyramid(item[:, :, 2], item[:, :, 0], methods[j])

            img = cv2.merge((item[:, :, 0], ag, ar))
            end_time = time.time() - start_time
            aligend.append([str(i), img, end_time])
            print([methods[j], str(i), end_time])

            # # Filename
            # filename = 'image pyramid '+str(i)+'method'+methods[j]+'.jpg'
            #
            # # Using cv2.imwrite() method
            # # Saving the image
            # cv2.imwrite(filename, img)
            #
            # cv2.imshow(str(i)+str(j), img)
            # cv2.waitKey()
        aligend_images.append([methods[j], aligend])
```

که در این تابع همانند روش قبل عمل شده است که به جای تابع align از تابع pyramid استفاده شده است که به صورت زیر میباشد :

```
def pyramid(im1, im2, method, off_x=(-4, 4), off_y=(-4, 4), depth=5):
    if im1.shape[0] < 400 or depth == 0:
        return align(im1, im2, method)
    else:
        _, best_shift = pyramid(
            cv2.resize(im1, dsize=(int((im1.shape[0]) / 2), int((im1.shape[1]) / 2)), interpolation=cv2.INTER_CUBIC),
            cv2.resize(im1, dsize=(int((im2.shape[0]) / 2), int((im2.shape[1]) / 2)), interpolation=cv2.INTER_CUBIC),
            method, depth=depth - 1)
        best_shift *= 2
        result, new_shift = align(np.roll(im1, best_shift, (0, 1)), im2, method, off_x, off_y)
        best_shift += new_shift
        return result, best_shift
```

که در این تابع ابتدا سطح وضوح را به کمترین حد رسانده و تابع align را برای کمترین وضوح محاسبه کرده سپس از آن برای سطوح بالاتر استفاده میشود. برای تابع align دوباره از ۲ روش میتوانیم استفاده کنیم که نتیجه آن ها در زیر آورده شده است.

ابتدا روش SSD :




سپس روش دوم یعنی NCC :

سرعت این روش نیز در ادامه آورده شده است :

```
['SSD', '0', 0.7025902271270752]
['SSD', '1', 0.7031219005584717]
['SSD', '2', 0.7343780994415283]
['SSD', '3', 0.7187519073486328]
['NCC', '0', 5.437872648239136]
['NCC', '1', 4.794828176498413]
['NCC', '2', 5.615850925445557]
['NCC', '3', 5.468754768371582]
```

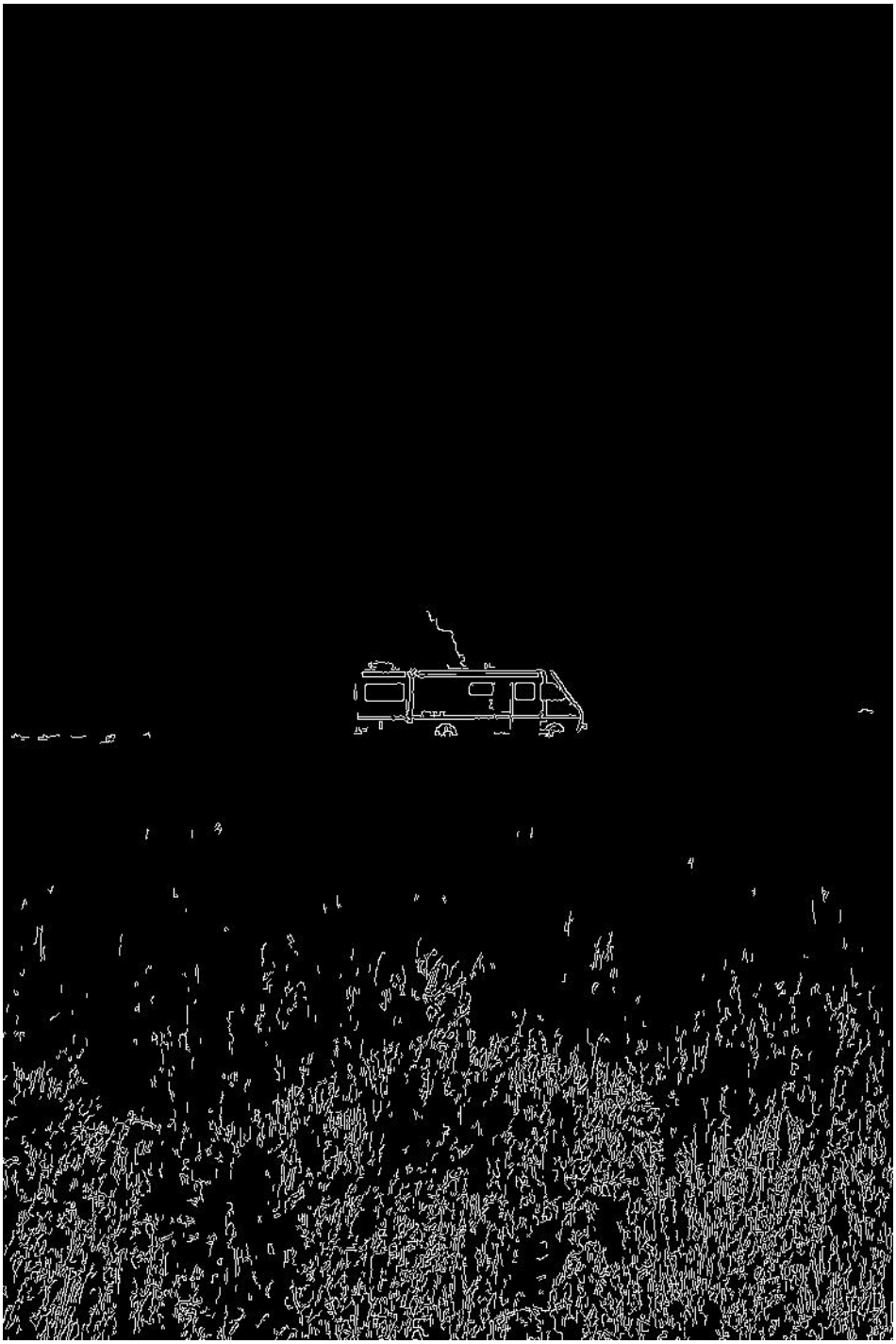
با توجه سرعت به دست آمده و مقایسه آن با ۲ روش اول متوجه میشویم این روش سرعت بیشتری دارد. در این حالت پیچیدگی زمانی از مرتبه ((O(log(n*m میباشد. اما ۲ روش اول پیچیدگی زمانی از مرتبه (O(n*m میباشند. (روش ساده)

.۴

در این سوال ابتدا تصویر edge را بارگذاری میکنیم. سپس از طریق کتابخانه opencv و تابع canny با حدهای پایین و بالا به ترتیب ۲۰۰ و ۴۰۰ لبه های تصویر مورد نظر را پیدا میکنیم. کد این بخش ها به صورت زیر میباشد :

```python
import cv2
import numpy as np

# using imread('path') and 1 denotes read as  color image
img = cv2.imread('edge.jpg', 1)

cv2.imshow("edge_image", img)
cv2.waitKey()

edges = cv2.Canny(img, 200, 400, True)
cv2.imshow("Edge Detected Image", edges)
cv2.imshow("Original Image", img)
cv2.waitKey(0)  # waits until a key is pressed
cv2.destroyAllWindows()  # destroys the window showing image

# Filename
filename = 'edge_canny_before.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, edges)
```

و نتیجه تصویر به صورت زیر میباشد :



در این حالت مشاهده میکنیم لبه های گندم زار نیز در تصویر نتیجه به حضور دارند که برای این سوال به صورت خطا هستند و باید برای حل این مشکل راه حلی ارائه نماییم. در این مسئله میتوانیم تصاویر گندم زار را به صورت نویز در نظر بگیریم. در این حالت میتوانیم از ۲ طریق این مشکل را حل نماییم. ۱. ابتدا یک فیلتر Median استفاده کرد سپس از طریق تابع Canny لبه ها را پیدا کنیم. ۲. از طریق استفاده از یک فیلتر گوسی سپس استفاده از تابع Canny.

برای اینکار ابتدا از روش اول استفاده میکنیم :

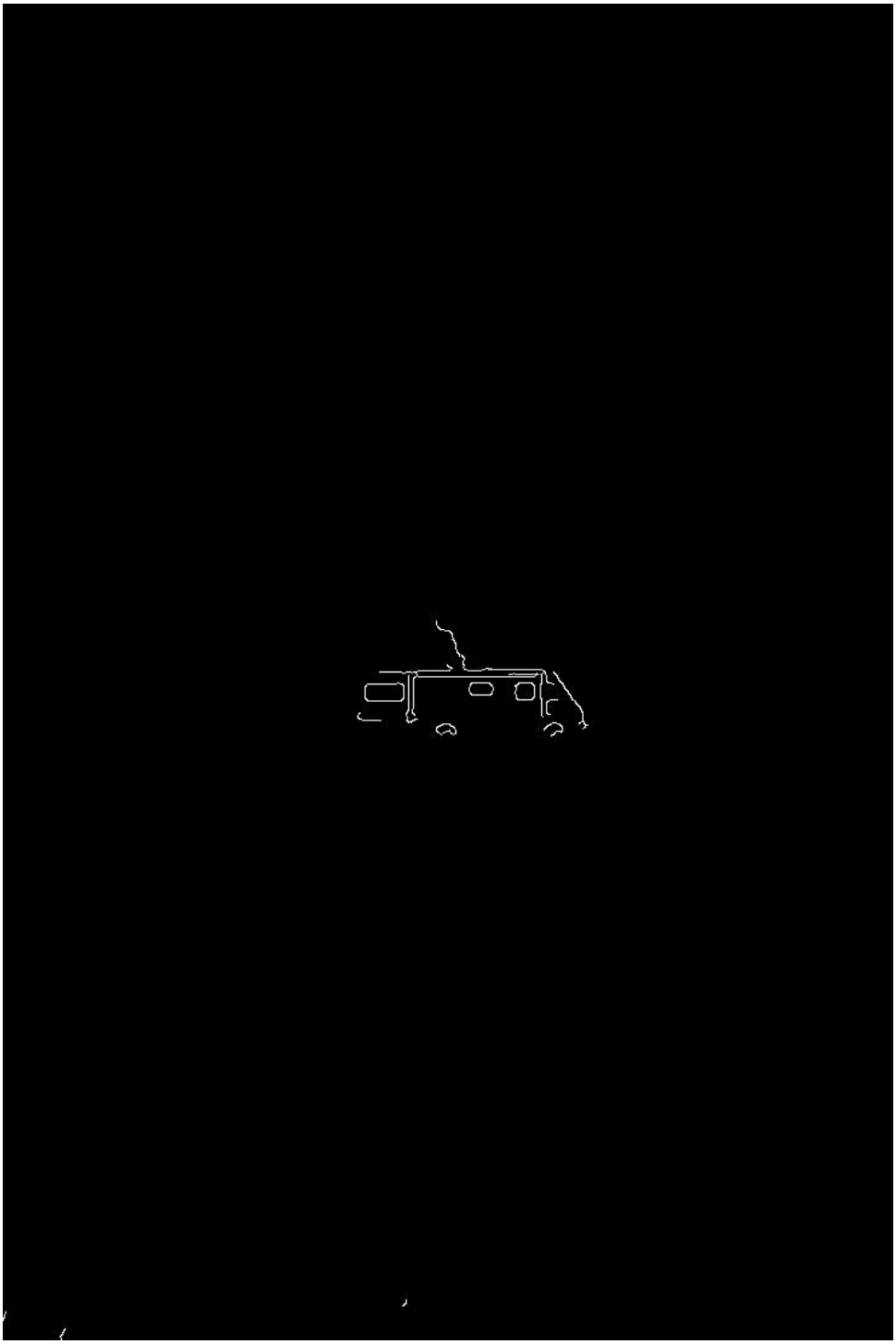
ابتدا یک فیلتر median را ایجاد میکنیم سپس تابع Canny با حدود ۲۰۰ و ۴۰۰ استفاده میکنیم . فیلتر میانه یک تکنیک فیلتر دیجیتال غیر خطی است که اغلب برای حذف نویز از تصویر یا سیگنال استفاده می شود. فیلتر میانی به طور گسترده ای در پردازش تصویر دیجیتال استفاده می شود، زیرا در شرایط خاص، لبه ها را حفظ می کند و نویز را حذف می کند. این یکی از بهترین الگوریتم ها برای حذف نویز نمک و فلفل است. کد آن به صورت زیر میباشد :

```
img_median = cv2.medianBlur(img, 5)
cv2.imshow("new", img_median)
cv2.waitKey(0)  # waits until a key is pressed

edges = cv2.Canny(img_median, 200, 400, True)
cv2.imshow("Edge Detected Image", edges)
cv2.imshow("Original Image", img_median)
cv2.waitKey(0)  # waits until a key is pressed
cv2.destroyAllWindows()  # destroys the window showing image
# Filename
filename = 'edge_canny_after_median.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, edges)
```

که نتیجه فیلتر به روی تصویر و همچنین نتیجه آن برای لبه ها در ادامه قرار داده شده است.

اعمال فیلتر Median بر تصویر اولیه :

اعمال تابع Canny بر روی تصویر فیلتر شده :



قابل مشاهده است که این فیلتر اکثر لبه های مربوط به گندم زار را حذف کرده و نتیجه مطلوبی ایجاد کرده است.

در حالت دوم از یک فیلتر گاسی با کرنل ۵ در ۵ استفاده کرده ایم. سپس از تابع Canny استفاده کرده و تصاویر مربوط به هر بخش را ایجاد کرده ایم. این فیلتر یک افکت پرکاربرد در نرم افزارهای گرافیکی است که معمولاً برای کاهش نویز تصویر و کاهش جزئیات استفاده می شود. همچنین قبل از اعمال مدل های یادگیری ماشینی یا یادگیری عمیق، به عنوان مرحله پیش پردازش استفاده می شود.

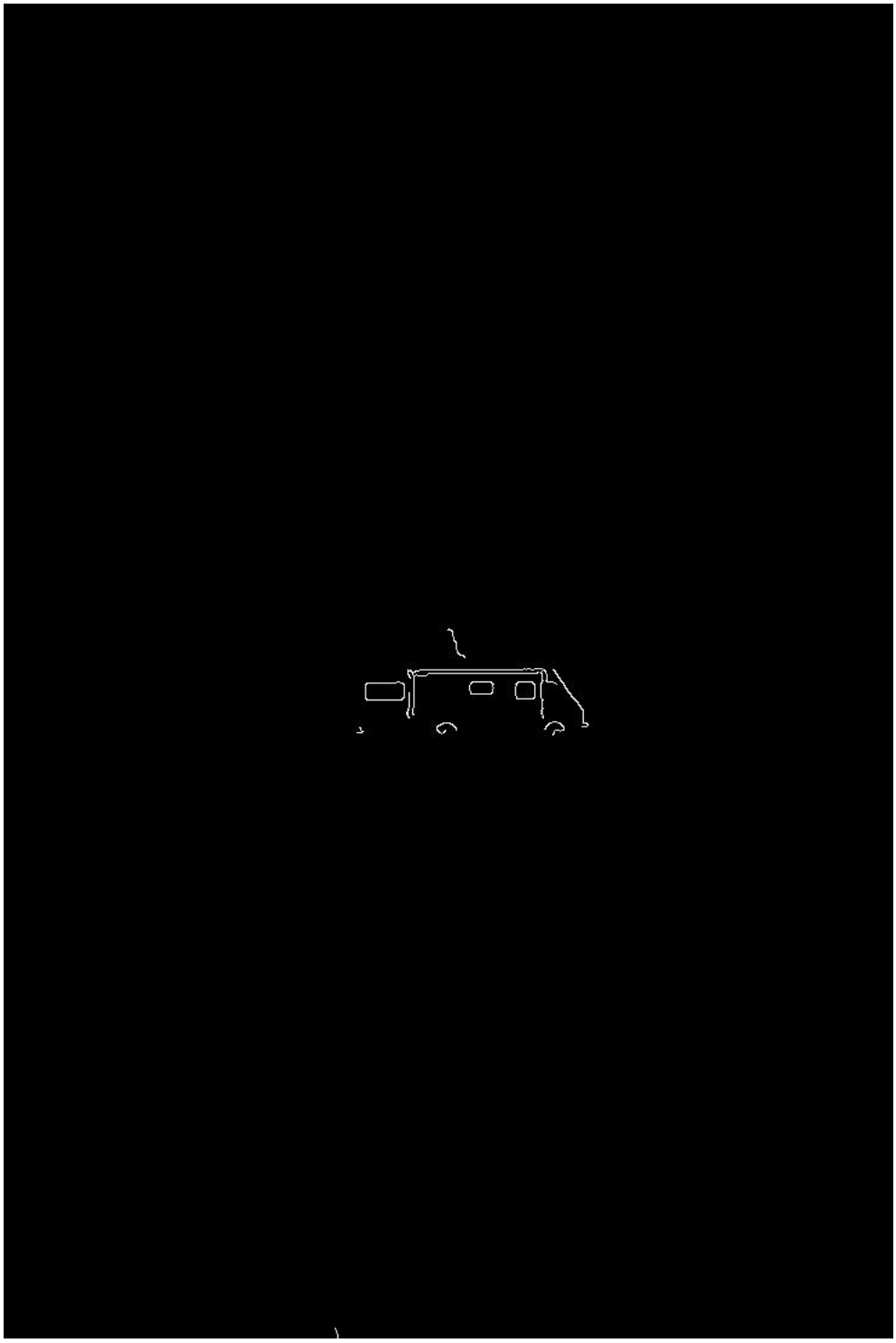
کد آن به صورت زیر نوشته شده است :

```
img_gauss = cv2.GaussianBlur(img, (5, 5), 0)
cv2.imshow("new", img_gauss)
cv2.waitKey(0)  # waits until a key is pressed

edges = cv2.Canny(img_gauss, 200, 400, True)
cv2.imshow("Edge Detected Image", edges)
cv2.imshow("Original Image", img_gauss)
cv2.waitKey(0)  # waits until a key is pressed
# Filename
filename = 'edge_canny_after_gauss.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, edges)

# Filename
filename = 'image_gauss.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, img_gauss)
```

در ادامه تصاویر مربوط به تصویر اصلی به همراه فیلتر گاسی و تصویر لبه ها قرار داده شده است.

تصویر اصلی با اعمال فیلتر گاسی به کرنل ۵ در ۵ :

تصویر لبه های مربوط به تصویر اصلی که فیلتر گاسی بر آن اعمال شده است:



مشاهده میشود این فیلتر به خوبی جزئیات مربوط به گندم زار را حذف کرده است و لبه های اصلی مربوط به ماشین ون را به خوبی حفظ کرده است.

۵.

در حالتی که تصاویر به صورت Gray Level میباشند ، با اعمال تابع cv2.equalizeHist() به سادگی میتوان هموارسازی هیستوگرام را انجام داد. اما برای تصاویر رنگی این امکان وجود ندارد. برای همین میتوانیم از ایده های مختلفی استفاده کرد. یکی از ایده ها این است که تصاویر مربوط به هر کانال رنگی را جدا کرده و هیستوگرام مربوط به آن را به دست آوریم. سپس هموارسازی هر هیستوگرام را با تابع cv2.equalizeHist() به دست آورده و تصویر مربوط به هر کانال را ذخیره کنیم. در مرحله آخر نیز کانال ها را با یکدیگر ترکیب کرده و تصویر جدید با هموارسازی هیستوگرام را به دست آوریم.

برای پیاده سازی این ایده ابتدا ، تصویر خود را در ورودی فراخوانی میکنیم. برای این سوال از تصویر تعدادی برگ چنار خشک شده استفاده کرده ایم و آن را نمایش میدهیم :

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('hcont.jpg', 1)

cv2.imshow("main_image", img)
cv2.waitKey()
```

تصویر ورودی به صورت زیر میباشد :

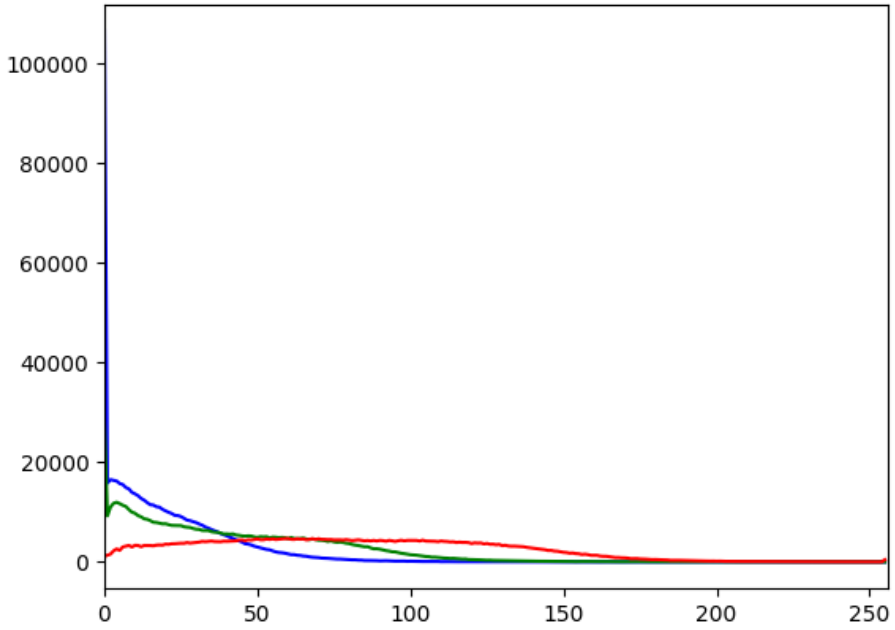
سپس تابعی به صورت زیر برای آن نوشته ایم :

```python
def histogram_equalization(img_in):
    # segregate color streams
    b, g, r = cv2.split(img_in)
    color = ('b', 'g', 'r')
    fig, ax = plt.subplots()
    for i, col in enumerate(color):
        histr = cv2.calcHist([img_in], [i], None, [256], [0, 256])
        ax.plot(histr, color=col)
        ax.set_xlim([0, 256])
    fig.show()
    fig.savefig('histogram_image_before.png')
    plt.close(fig)

    equ_b = cv2.equalizeHist(b)
    equ_g = cv2.equalizeHist(g)
    equ_r = cv2.equalizeHist(r)
    img_out = cv2.merge((equ_b, equ_g, equ_r))
    fig1, ax1 = plt.subplots()
    for i, col in enumerate(color):
        histr = cv2.calcHist([img_out], [i], None, [256], [0, 256])
        ax1.plot(histr, color=col)
        ax1.set_xlim([0, 256])
    fig1.show()
    fig1.savefig('histogram_image_after.png')
    plt.close(fig1)
    return img_out
```

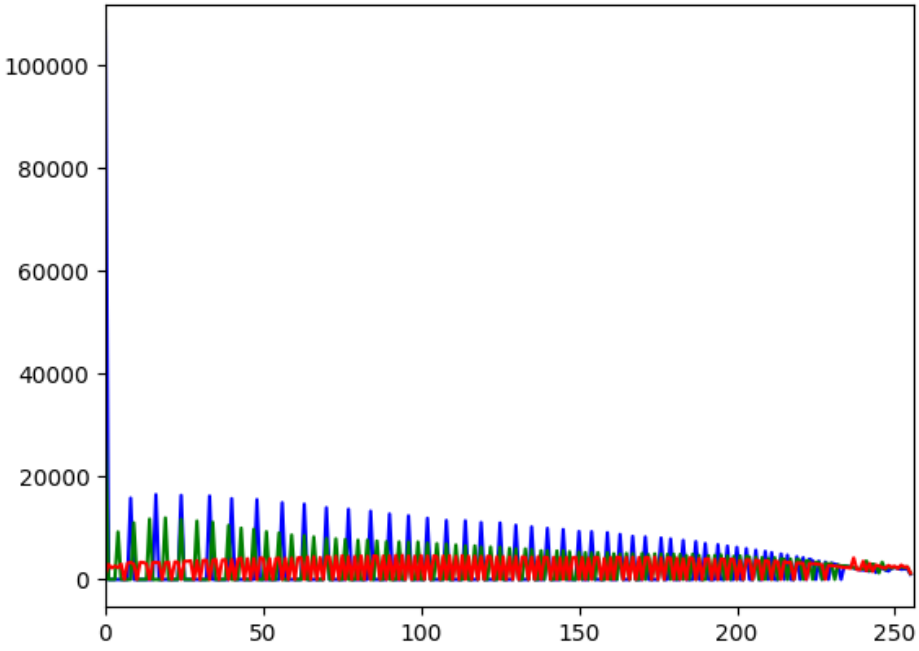
در این تابع ابتدا تصویر را گرفته ، کانال ها را جدا میکنیم. سپس هیستوگرام مربوط به هر تابع را نمایش میدهیم. سپس با تابع cv2.equalizeHist() برای هر کانال هموارسازی کرده و هیستوگرام بعد از هموار سازی را نمایش میدهیم. در انتها تصاویر هر کانال را با یکدیگر ادغام کرده و تصویر جدید را باز میگردانیم .

تصویر هیستوگرام مربوط به این عکس قبل و بعد از هموار سازی به صورت زیر است :

هیستوگرام پیش از هموارسازی :



هیستوگرام پس از هموارسازی :



سپس تصاویر کانال ها را ادغام کرده و تصویر خروجی زیر ایجاد شده است :

قابل مشاهده است که تصویر خروجی نسبت به تصویر ورودی کاملا دچار تغییر شده و از لحاظ رنگی در همه ابعاد به حالت متعادلی رسیده است.

هیستوگرام تصویر در بسیاری از خدمات مدرن وجود دارد. عکاسان می‌توانند از آن‌ها به عنوان کمکی برای نشان دادن توزیع تون‌های گرفته‌شده و اینکه آیا جزئیات تصویر به‌خاطر نقاط برجسته یا سایه‌های سیاه‌شده از بین رفته است، استفاده کنند. هموارسازی هیستوگرام یک تکنیک پردازش تصویر کامپیوتری است که برای بهبود کنتراست در تصاویر استفاده می شود. این کار را با پخش مؤثرترین مقادیر شدت انجام می دهد، یعنی باز کردن دامنه شدت تصویر. لزومی ندارد که کنتراست همیشه در این مورد افزایش یابد. ممکن است مواردی وجود داشته باشد که هموارسازی هیستوگرام می تواند بدتر باشد. در این موارد کنتراست کاهش می یابد.

در مورد تصویر بررسی شده با دقت در تصویر قابل مشاهده است بخش هایی از تصویر مانند زمین یا گیاهان سبزی که در تصویر زیر دور آن دایره کشیده شده است ، در تصویر اصلی به سختی قابل مشاهده است که با این هموارسازی این بخش ها به سادگی قابل مشاهده بوده و همچنین جزییات هر برگ و بخش های مختلف تصویر کاملا قابل جداسازی هستند .