تمرین ۷ سوال ۱-۱-۷

چکیده

در این تمرین الگوریتم تشخیص گوشه Harris را پیاده سازی می کنیم و نتیجه را در حالات مختلف و با Scale های مختلف بدست می آوریم.

مقدمه

برای پیاده سازی الگوریتم Harris Corner Detetor کافیست تا مراحل زیر را طی کنیم:

۱. در صورت نیاز تصویر را Blur کنیم.

۲. Derivation های I_x و I_y را بدست آوریم.

۳. با به توان رساندن و ضرب مشتق های مرحله قبل I_{x2}، I_{y2} و I_{xy} را بدست می آوریم.

۴. بار دیگر فیلتر گوسی را روی هر کدام از مشتق های بدست آمده اعمال می کنیم و ماتریس Second Moment را می سازیم.

۵. مقدار Harris Response را با فرمول $harr = det(M) - \alpha * trace(M)^2$ محاسبه می کنیم.

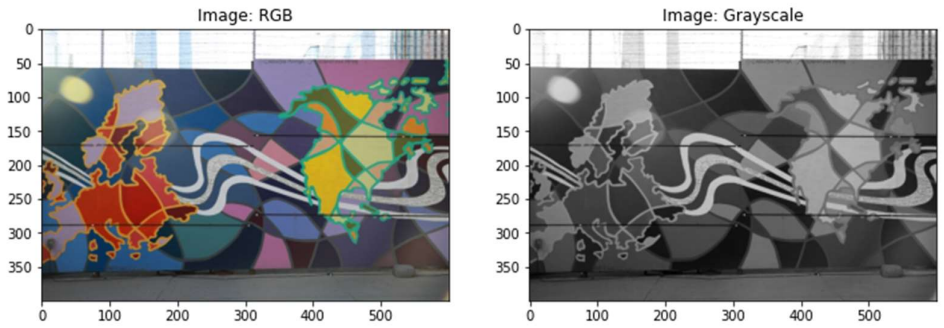
۶. با توجه به Threshold تعیین شده، بررسی می کنیم که آیا نقاط Corner هستند یا خیر.

بنابراین در کد نیز به همین صورت مراحل را پیاده سازی می کنیم و با تشکیل Second Moment Matrix و بررسی پنجره های مختلف روی آن، مقادیر Harris Response را بدست آورده و نرمال می کنیم و به مرحله انتخاب Corner بر اساس Threshold تعیین شده ادامه می دهیم. در نهایت ویژگی های انتخاب شده را با رنگ قرمز در تصویر مشخص می کنیم.

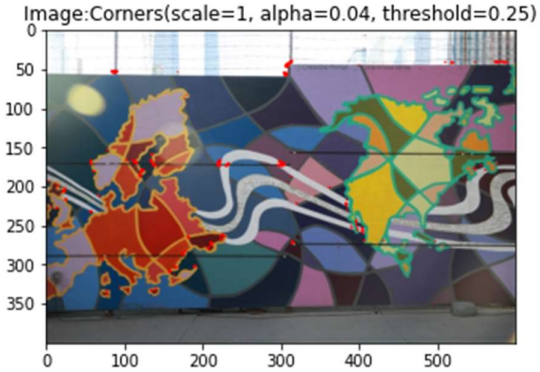
شرح نتایج

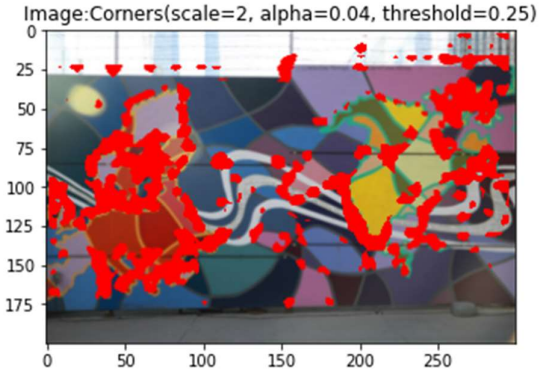
نتایج را برای Threshold های مختلف ۰,۲۵، ۰,۵ و ۰,۷۵ به همراه مقادیر آلفا مختلف ۰,۰۴، ۰,۰۵ و ۰,۰۶ بررسی می کنیم و همه این حالات را برای Scale های مختلف (Downsample کردن تصویر اصلی به تعداد ۳ بار و استفاده از ۴ Scale مختلف) محاسبه می کنیم.

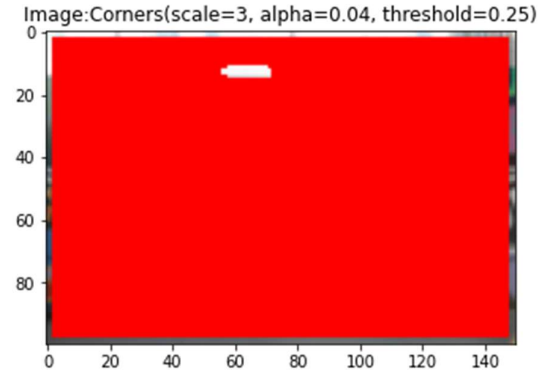
تصویر اصلی

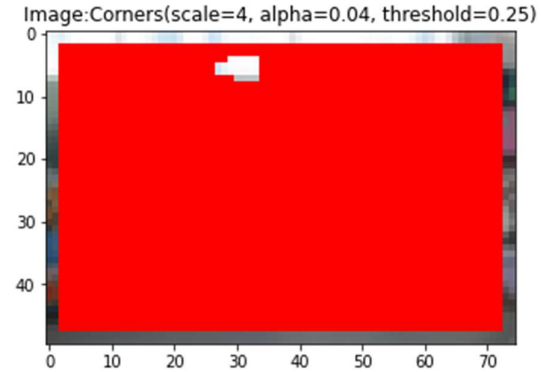


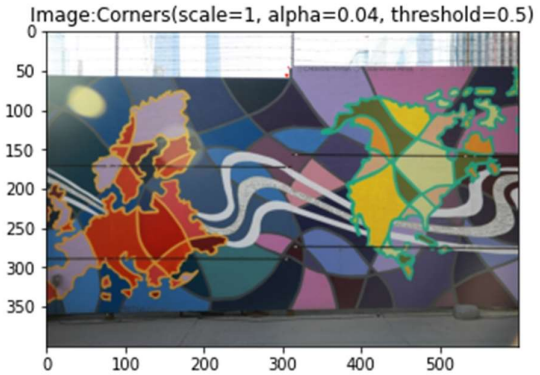
نتایج اعمال الگوریتم

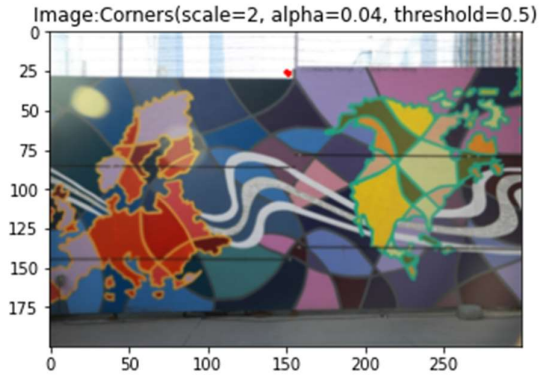


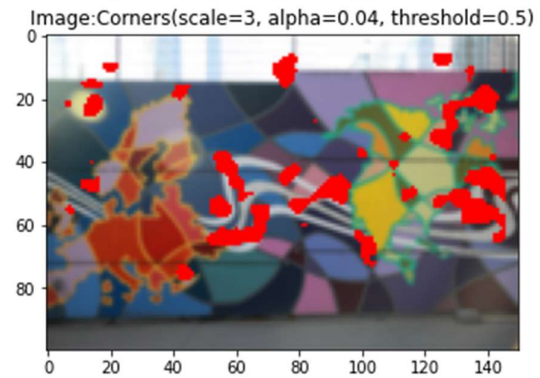


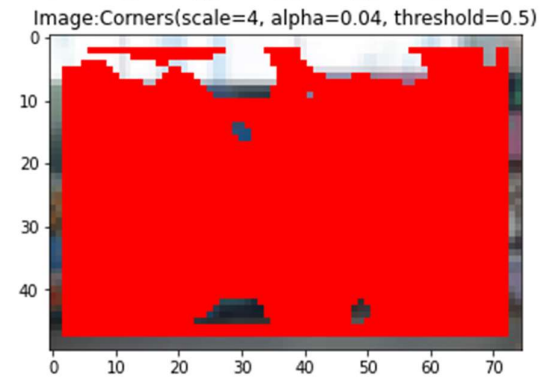












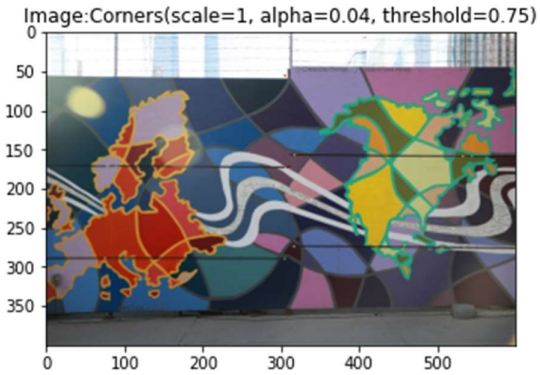


Image:Corners(scale=1, alpha=0.04, threshold=0.75)

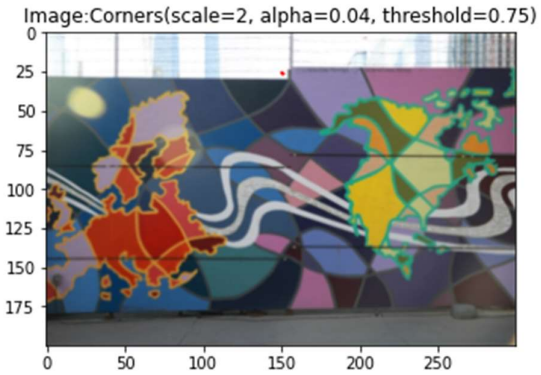


Image:Corners(scale=2, alpha=0.04, threshold=0.75)

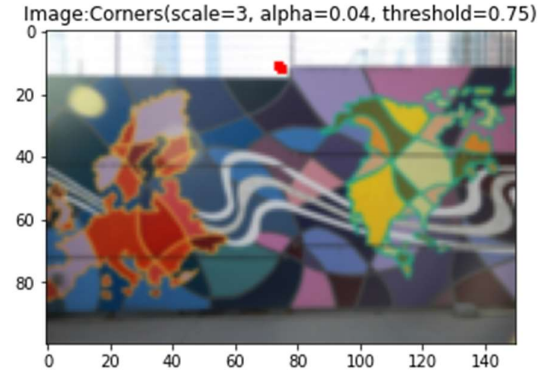


Image:Corners(scale=3, alpha=0.04, threshold=0.75)

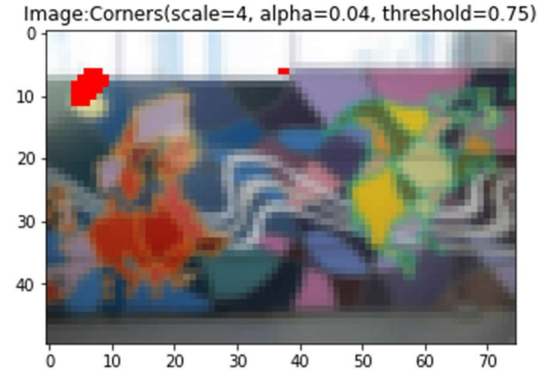


Image:Corners(scale=4, alpha=0.04, threshold=0.75)

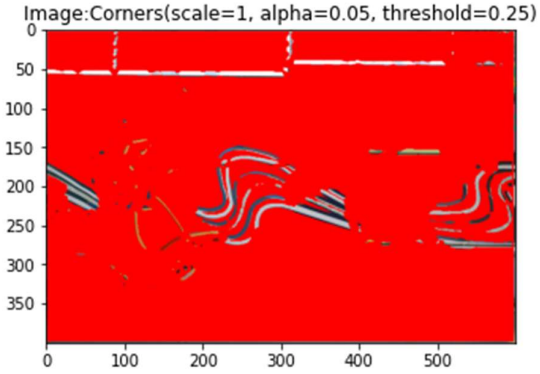


Image:Corners(scale=1, alpha=0.05, threshold=0.25)

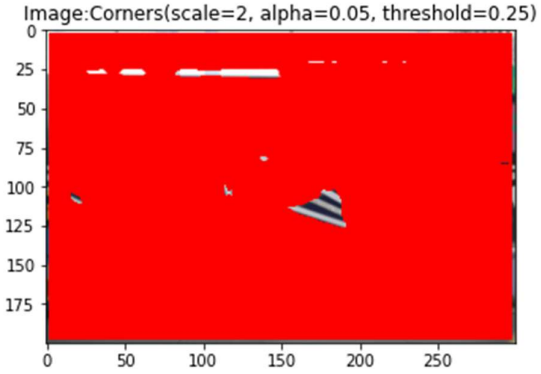


Image:Corners(scale=2, alpha=0.05, threshold=0.25)

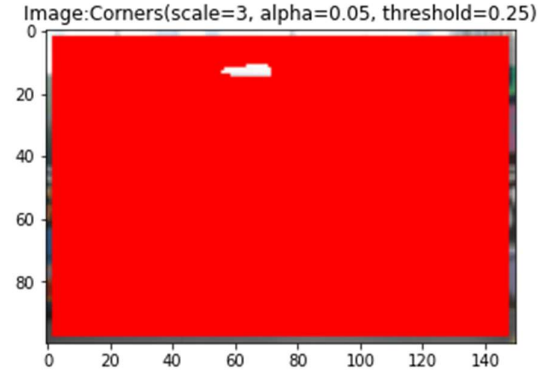


Image:Corners(scale=3, alpha=0.05, threshold=0.25)

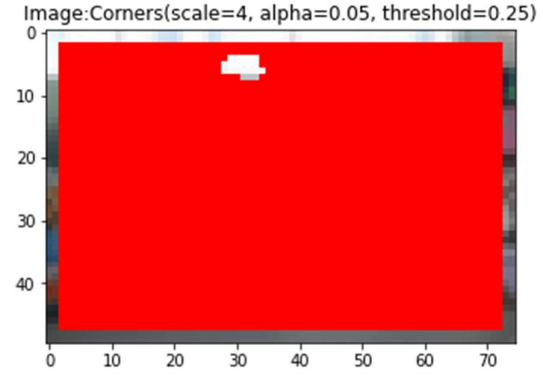


Image:Corners(scale=4, alpha=0.05, threshold=0.25)

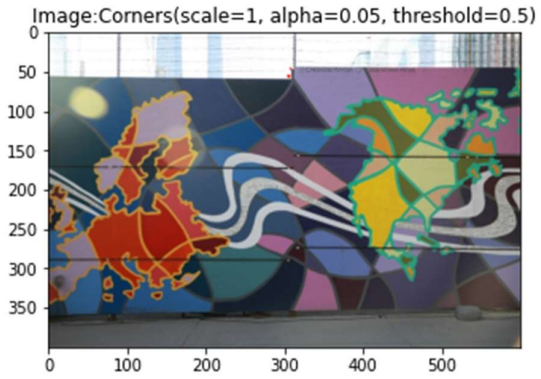


Image:Corners(scale=1, alpha=0.05, threshold=0.5)

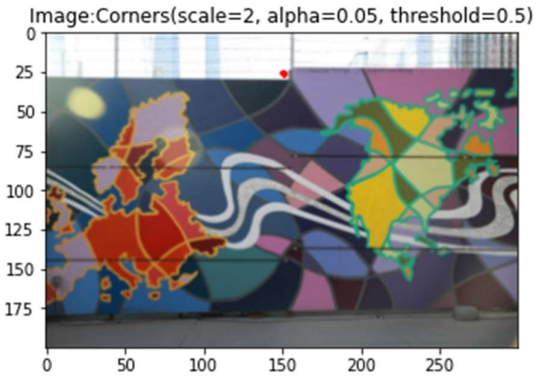


Image:Corners(scale=2, alpha=0.05, threshold=0.5)

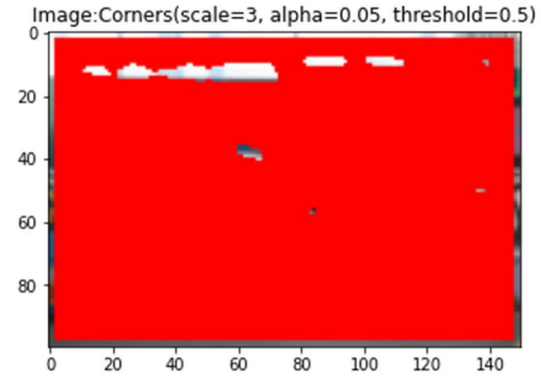


Image:Corners(scale=3, alpha=0.05, threshold=0.5)

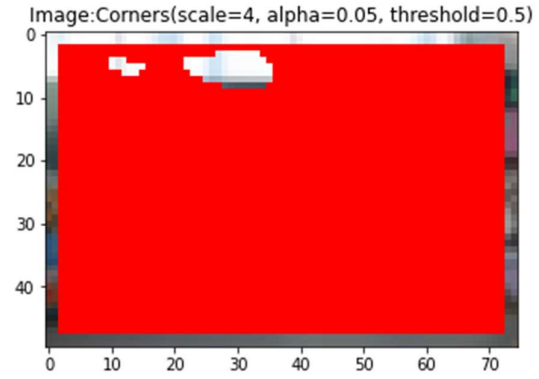


Image:Corners(scale=4, alpha=0.05, threshold=0.5)

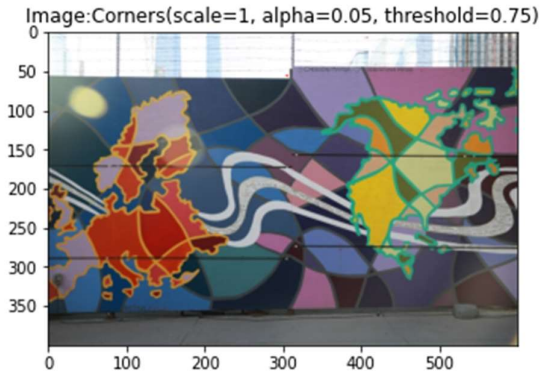


Image:Corners(scale=1, alpha=0.05, threshold=0.75)

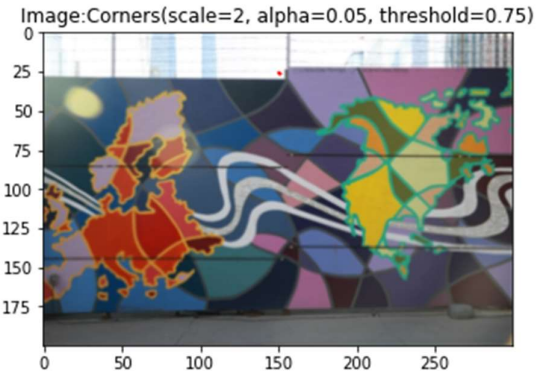


Image:Corners(scale=2, alpha=0.05, threshold=0.75)

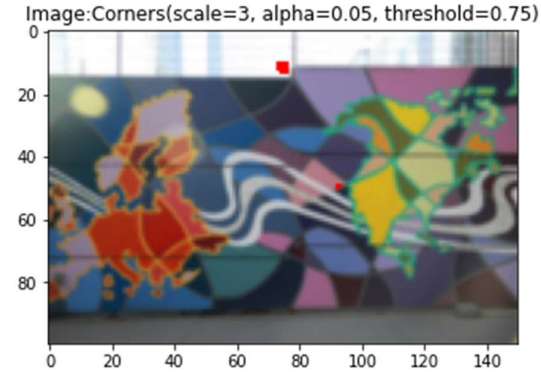


Image:Corners(scale=3, alpha=0.05, threshold=0.75)

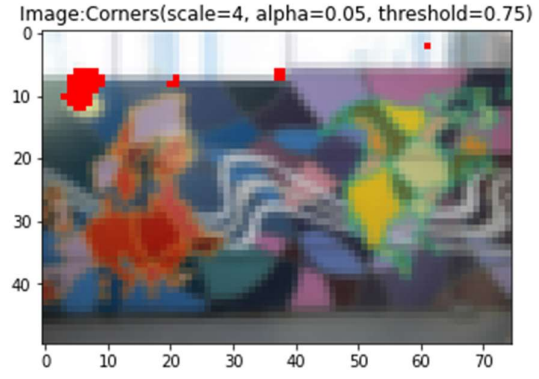


Image:Corners(scale=4, alpha=0.05, threshold=0.75)

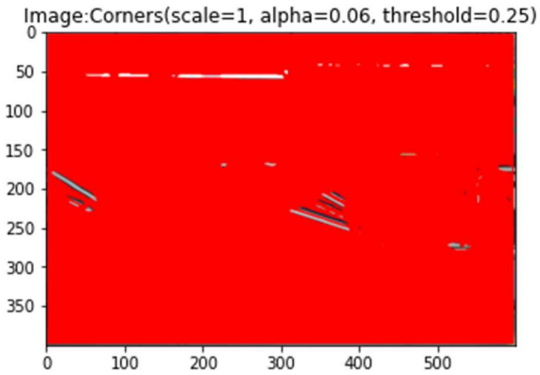


Image:Corners(scale=1, alpha=0.06, threshold=0.25)

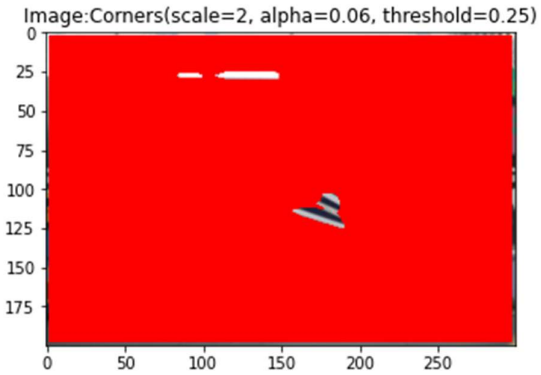


Image:Corners(scale=2, alpha=0.06, threshold=0.25)

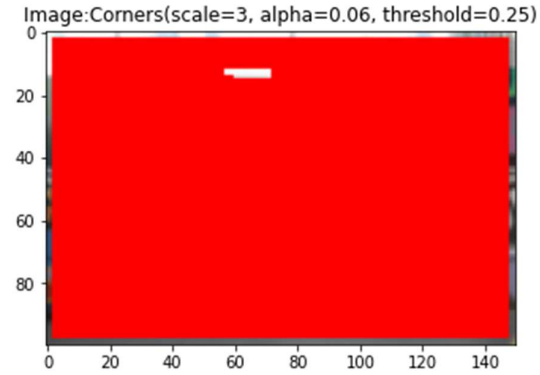


Image:Corners(scale=3, alpha=0.06, threshold=0.25)

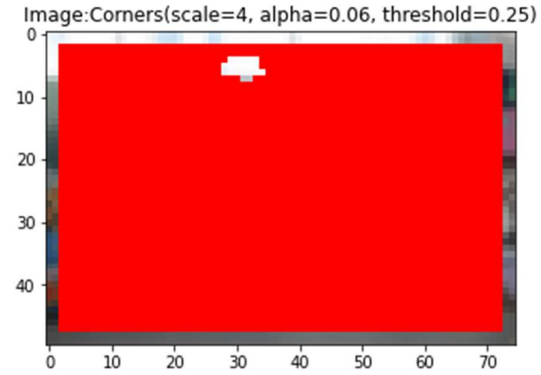


Image:Corners(scale=4, alpha=0.06, threshold=0.25)

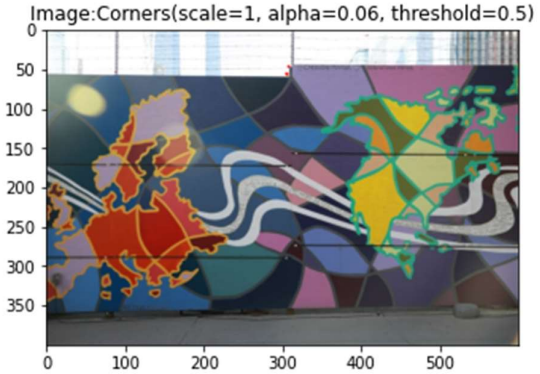


Image:Corners(scale=1, alpha=0.06, threshold=0.5)

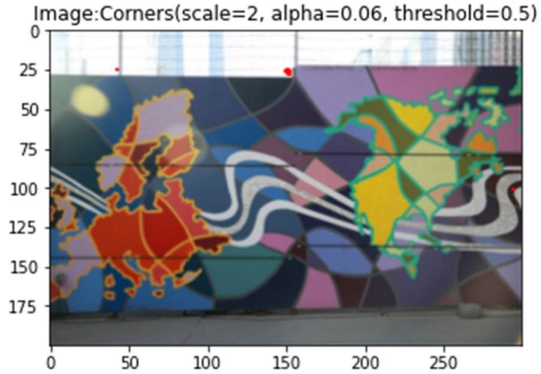


Image:Corners(scale=2, alpha=0.06, threshold=0.5)

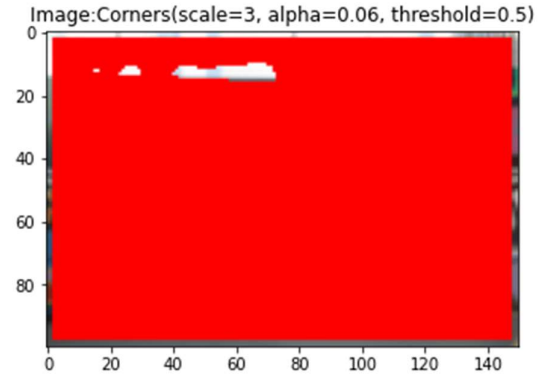


Image:Corners(scale=3, alpha=0.06, threshold=0.5)

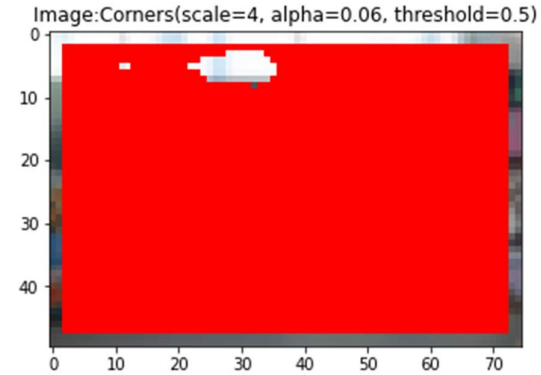
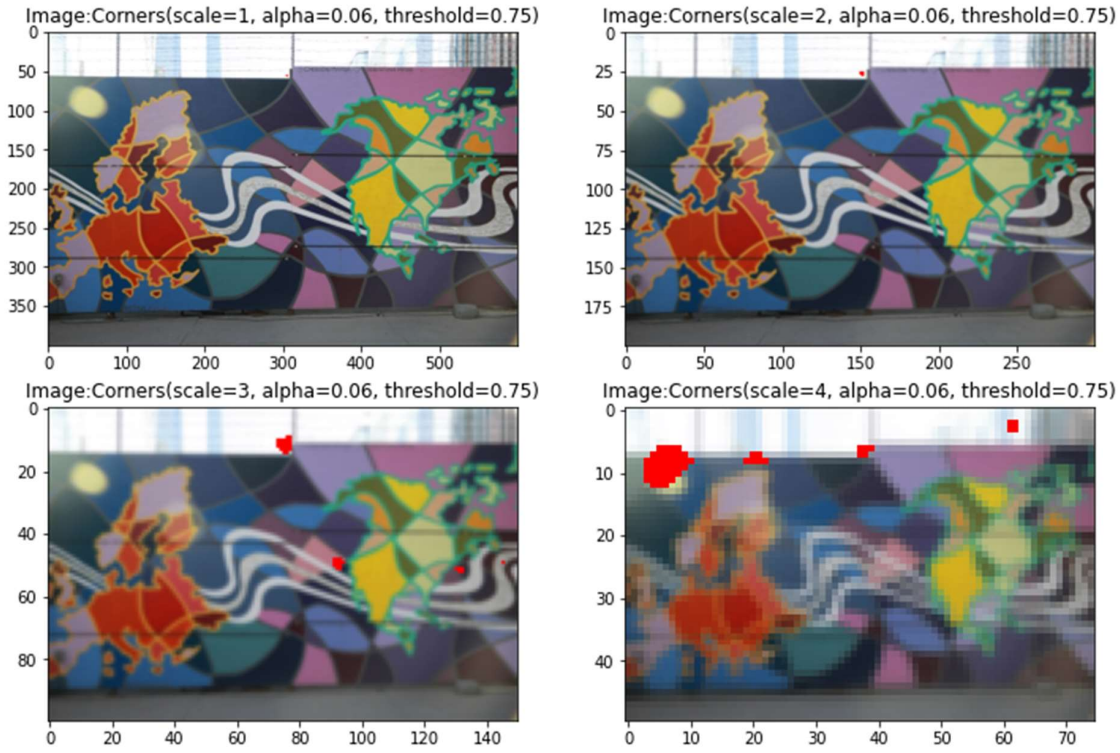


Image:Corners(scale=4, alpha=0.06, threshold=0.5)



همانطور که مشاهده می شود، برخی Threshold ها در برخی Scale ها به خوبی پاسخ داده و برخی از آنها نیز با Threshold های متفاوتی به بهترین حالت خود می رسند. بنابراین برای هر Scale مختلف، Threshold های متفاوتی را انتخاب کرده و بهترین انتخاب ها را در زیر می بینیم:

Scale 1: Alpha = 0.04 و Threshold = 0.25

Scale 2: Alpha = 0.04 و Threshold = 0.25

Scale 3: Alpha = 0.04 و Threshold = 0.5

Scale 4: Alpha = 0.06 و Threshold = 0.75

تعداد Interest Point ها را در هر مرحله بدست می آوریم و می بینیم که چه تعدادی به صورت مشترک بدست آمده اند:

```
Feature Counts:
With Alpha = 0.04

With Threshold = 0.25
Scale = 1 -> Feature Count = 1141
Scale = 2 -> Feature Count = 10396
Scale = 3 -> Feature Count = 13957
Scale = 4 -> Feature Count = 3243

With Threshold = 0.5
Scale = 1 -> Feature Count = 17
Scale = 2 -> Feature Count = 18
Scale = 3 -> Feature Count = 1062
Scale = 4 -> Feature Count = 2987

With Threshold = 0.75
Scale = 1 -> Feature Count = 6
```

```
Scale = 2 -> Feature Count = 6
Scale = 3 -> Feature Count = 14
Scale = 4 -> Feature Count = 28

With Alpha = 0.05

With Threshold = 0.25
Scale = 1 -> Feature Count = 220825
Scale = 2 -> Feature Count = 57090
Scale = 3 -> Feature Count = 13965
Scale = 4 -> Feature Count = 3245

With Threshold = 0.5
Scale = 1 -> Feature Count = 20
Scale = 2 -> Feature Count = 19
Scale = 3 -> Feature Count = 13668
Scale = 4 -> Feature Count = 3189

With Threshold = 0.75
Scale = 1 -> Feature Count = 6
Scale = 2 -> Feature Count = 6
Scale = 3 -> Feature Count = 18
Scale = 4 -> Feature Count = 40

With Alpha = 0.06

With Threshold = 0.25
Scale = 1 -> Feature Count = 233085
Scale = 2 -> Feature Count = 57440
Scale = 3 -> Feature Count = 13974
Scale = 4 -> Feature Count = 3246

With Threshold = 0.5
Scale = 1 -> Feature Count = 26
Scale = 2 -> Feature Count = 28
Scale = 3 -> Feature Count = 13857
Scale = 4 -> Feature Count = 3210

With Threshold = 0.75
Scale = 1 -> Feature Count = 6
Scale = 2 -> Feature Count = 8
Scale = 3 -> Feature Count = 41
Scale = 4 -> Feature Count = 54
```

تمرین ۷ سوال ۲-۷

چکیده

در این تمرین به پیاده سازی عملیات Feature Selection و ساخت تصویر Panorama توسط الگوریتم SIFT می پردازیم.

مقدمه

برای پیاده سازی این تمرین، انتخاب بین الگوریتم های SURF و SIFT بود که از استفاده از الگوریتم SURF به علت عدم قابلیت استفاده تجاری منصرف شده و از الگوریتم SIFT استفاده کردیم.

با توجه به مراحل زیر، توابع مورد نیاز درکد ارائه شده است که از مثال موجود در این لینک برای کمک گرفتن استفاده شده است.

۱. اجرای الگوریتم Feature Selection و بدست آوردن Key Point ها و Desriptor ها برای تصاویر انتخابی.

۲. پیدا کردن نقاط Key Point مشترک بین دو تصویر (نگاه Vector-Based).

۳. محاسبه ماتریس Homography.

۴. دوختن و Stitching دو تصویر.

در هر بخش از الگوریتم ها و مراحل زیر استفاده شده است:

۱. اجرای الگوریتم SIFT به عنوان الگوریتم Feature Selection.

۲. یافتن نقاط مشترک با استفاده از BFMatcher و knnMatch.

۳. محاسبه ماتریس Homography با پیاده سازی توسط RANSAC.

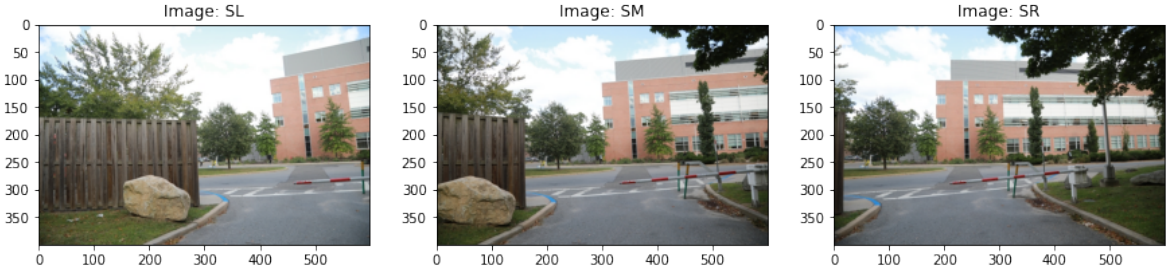
۴. دوخت دو تصویر با استفاده از ماتریس Homography و ساخت mask برای جدا کردن بخش های چپ و وسط و راست تصویر و ساخت جداگانه هر کدام.

برای اتصال سه تصویر نیز ابتدا تصاویر راست و وسط را به هم و وسط و چپ را به هم به صورت جداگانه متصل کرده و بعد دو تصویر بدست آمده را به هم وصل می کنیم.

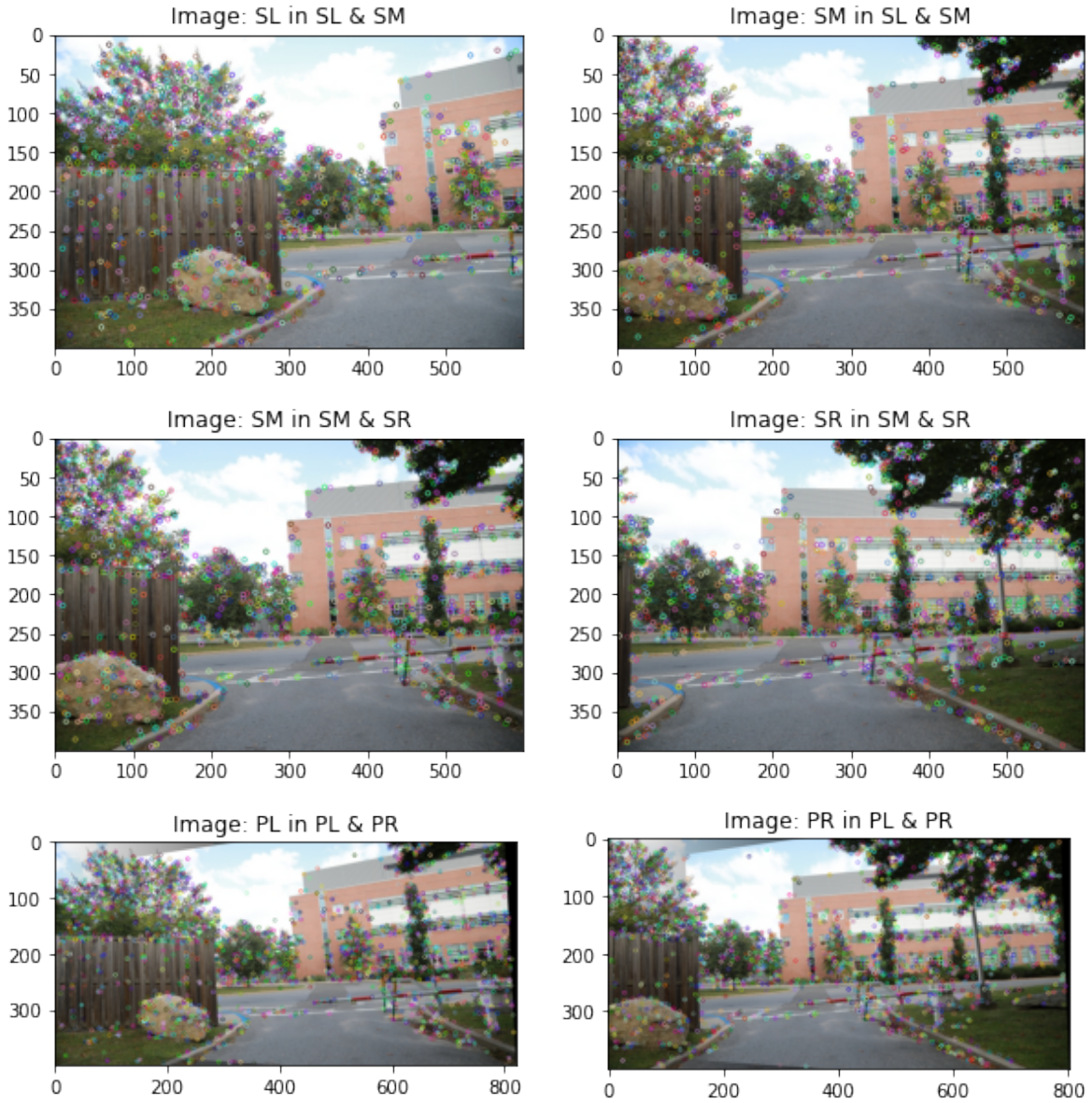
در بخش نتیجه گیری، نتیجه انتخاب ویژگی ها و ویژگی های مشترک بین دو تصویر و در نهایت تصویر پانوراما را برای تصاویر آموزشی داده شده مشاهده نموده و برای پاسخگویی به تمرین بخش ۱-۲-۷ نیز همین الگوریتم ها را برای تصاویر گرفته شده توسط دوربین خودم پیاده سازی کرده و مشاهده می کنیم.

شرح نتایج

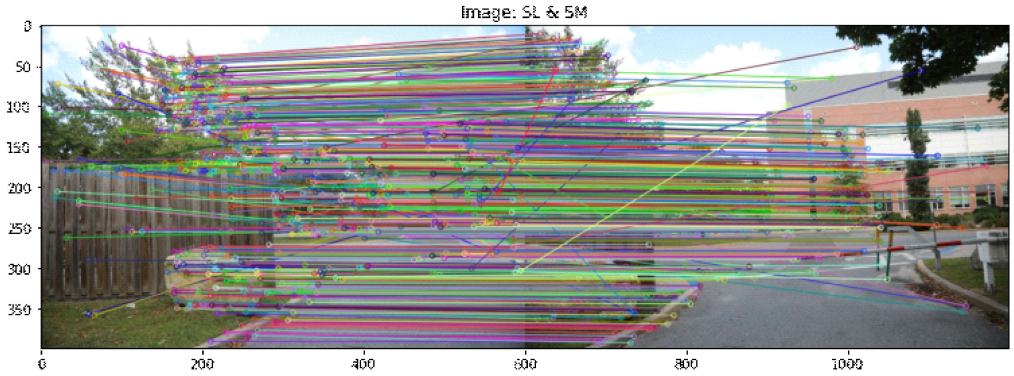
ابتدا تصاویر آموزشی داده شده را کنار هم مشاهده می کنیم.

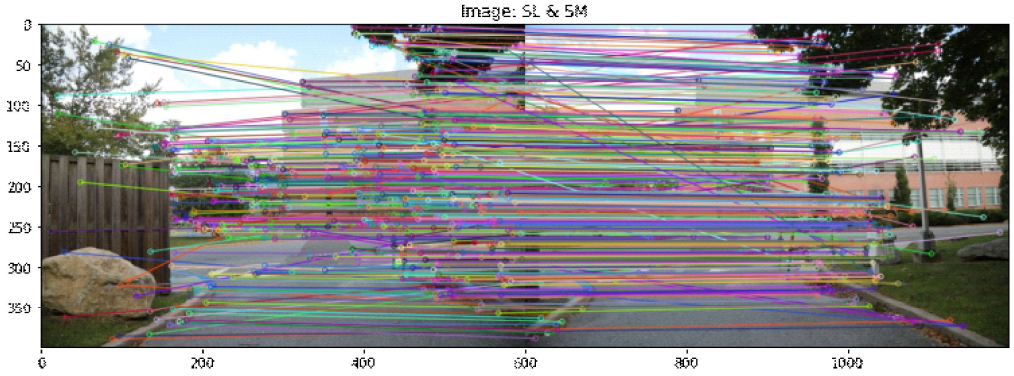


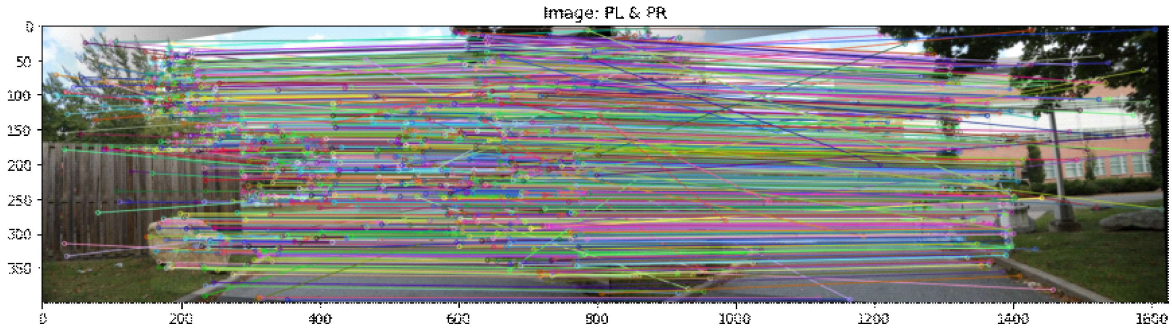
در ادامه ویژگی های انتخاب شده روی هر تصویر در هر مرحله را مشاهده می کنیم.



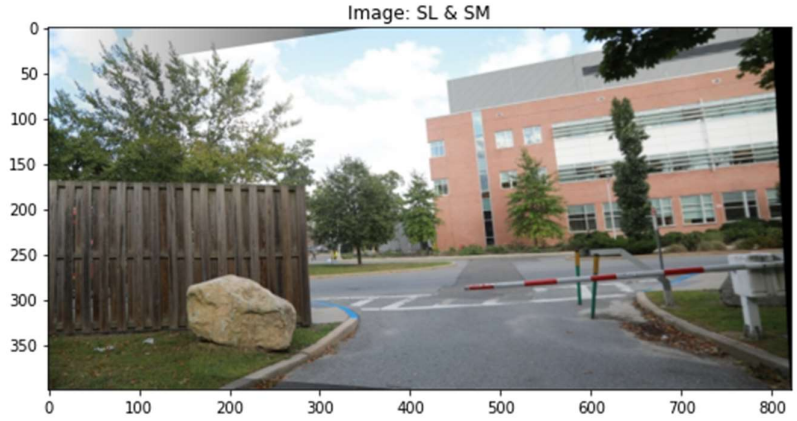
سپس ویژگی های Match شده بین تصاویر را مشاهده می کنیم.







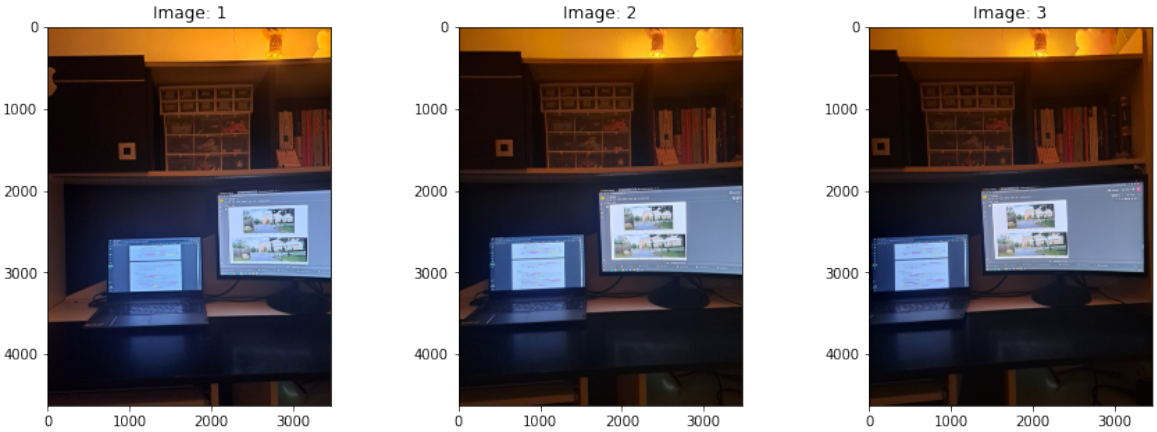
حال نتایج تصویر پانوراما را مشاهده می کنیم.



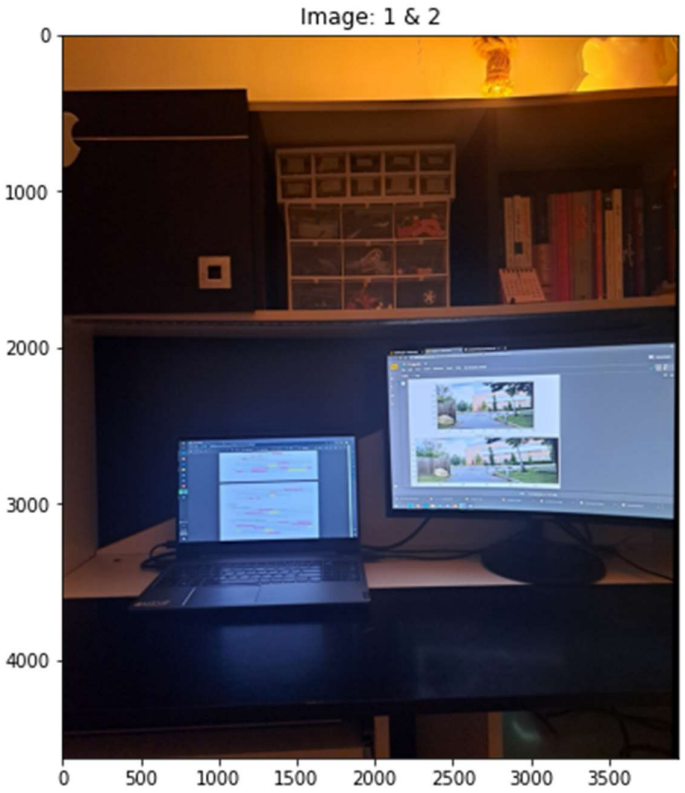
همانطور که مشاهده می شود الگوریتم به خوبی ویژگی های زیادی را به عنوان Match انتخاب کرده و تصویر پانوراما در مرحله سوم به خوبی ساخته شده است.

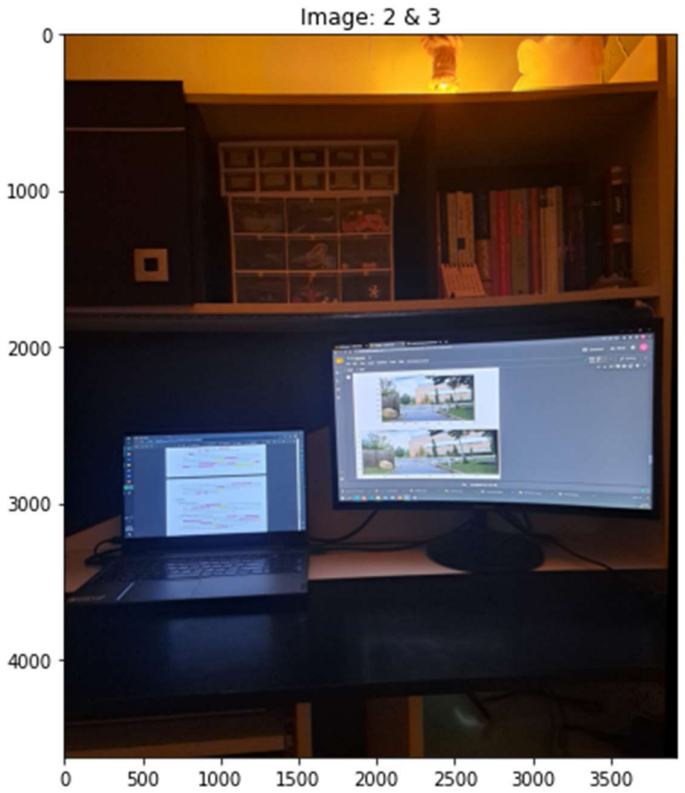
تصاویر گرفته شده توسط دوربین

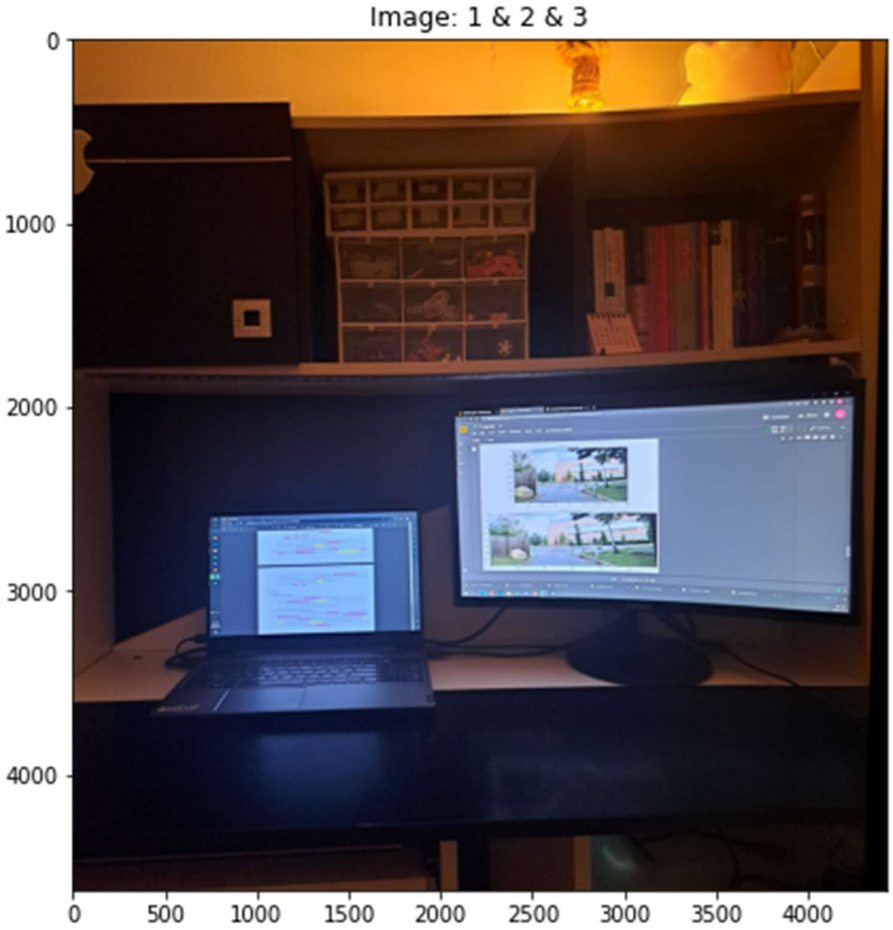
در ادامه تصاویر گرفته شده توسط دوربین خودم را مشاهده می کنیم و الگوریتم را بر روی آن نیز اعمال می کنیم.



نتیجه اعمال الگوریتم







همانگونه که می بینیم الگوریتم به خوبی در محیط کم نور عمل کرده و تصاویر را به هم چسبانده است.

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter
from google.colab import drive
```

In [ ]:
```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
ive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
img_bgr = cv2.imread("/content/drive/MyDrive/Images/7/harris.JPG")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
```

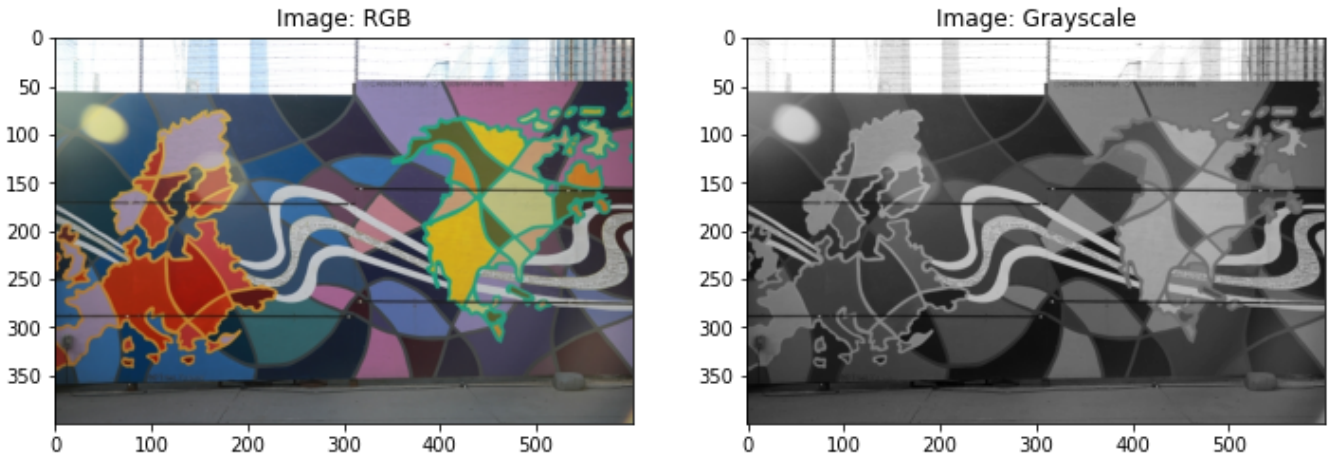
In [ ]:
```
def corner_response_calculator (image, alpha):
  blured_image = cv2.GaussianBlur(image, (3,3), 0)

  I_x = cv2.Sobel(blured_image, cv2.CV_64F, 1, 0, ksize=3)
  I_y = cv2.Sobel(blured_image, cv2.CV_64F, 0, 1, ksize=3)

  I_x2 = np.square(I_x)
  I_y2 = np.square(I_y)
  I_xy = I_x * I_y

  I_x2 = gaussian_filter(I_x2, 1)
  I_y2 = gaussian_filter(I_y2, 1)
  I_xy = gaussian_filter(I_xy, 1)

  window_size = 5
  offset = window_size // 2

  R, C = blured_image.shape
  harris_response = np.zeros((R, C))

  for r in range(offset, R - offset):
    for c in range(offset, C - offset):
      S_x2 = np.sum(I_x2[r - offset : r + 1 + offset, c - offset : c + 1 + offs
et])
      S_y2 = np.sum(I_y2[r - offset : r + 1 + offset, c - offset : c + 1 + offs
et])
      S_xy = np.sum(I_xy[r - offset : r + 1 + offset, c - offset : c + 1 + offs
et])

      second_moment_mastrix = np.array([[S_x2,S_xy], [S_xy,S_y2]])
      determinant = np.linalg.det(second_moment_mastrix)
      trace = np.matrix.trace(second_moment_mastrix)
      har = determinant - alpha * (trace ** 2)
      harris_response[r - offset, c - offset] = har

  cv2.normalize(harris_response, harris_response, 0, 1, cv2.NORM_MINMAX)
  return harris_response
```

In [ ]:
```
def corner_specification (image, harris_response, threshold):
  corner_specified_image = image.copy()

  R, C, ch = image.shape

  window_size = 5
  offset = window_size // 2
  feature_count = 0

  for r in range(offset, R - offset):
    for c in range(offset, C - offset):
      if harris_response[r][c] > threshold:
        corner_specified_image[r][c] = [255, 0, 0]
        feature_count += 1

  return corner_specified_image, feature_count
```

In [ ]:
```
def harris_corner_detector (image_rgb, image_gray, alpha, threshold):
  harris_response = corner_response_calculator(image_gray, alpha)
  corner_specified_image, feature_count = corner_specification(image_rgb, harri
s_response, threshold)
  return corner_specified_image, feature_count
```

In [ ]:
```
fig, plot = plt.subplots(1, 2, figsize = (12, 6))

plot[0].imshow(img_rgb)
plot[0].set_title("Image: RGB")

plot[1].imshow(img_gray, cmap='gray')
plot[1].set_title("Image: Grayscale")
```

Out[ ]:
```
Text(0.5, 1.0, 'Image: Grayscale')
```



In [ ]:
```
image_scale_1_gray = img_gray
image_scale_1_rgb = img_rgb

image_scale_2_gray = cv2.pyrDown(image_scale_1_gray)
image_scale_2_rgb = cv2.pyrDown(image_scale_1_rgb)

image_scale_3_gray = cv2.pyrDown(image_scale_2_gray)
image_scale_3_rgb = cv2.pyrDown(image_scale_2_rgb)

image_scale_4_gray = cv2.pyrDown(image_scale_3_gray)
image_scale_4_rgb = cv2.pyrDown(image_scale_3_rgb)
```

In [ ]:

```
thresholds = [0.25, 0.5, 0.75]
alphas = [0.04, 0.05, 0.06]

print("Feature Counts:")
for alpha_index in range (len(alphas)):
  alpha = alphas[alpha_index]
  print("With Alpha = " + str(alpha) + "\n")
  for threshold_index in range (len(thresholds)):
    threshold = thresholds[threshold_index]
    print("With Threshold = " + str(threshold))

    fig, plot = plt.subplots(2, 2, figsize = (12, 8))

    corner_specified_image_scale_1, feature_count_scale_1 = harris_corner_detec
tor(image_scale_1_rgb, image_scale_1_gray, alpha, threshold)
    plot[0][0].imshow(corner_specified_image_scale_1)
    plot[0][0].set_title("Image:Corners(scale=1, alpha=" + str(alpha) + ", thre
shold=" + str(threshold) + ")")

    corner_specified_image_scale_2, feature_count_scale_2 = harris_corner_detec
tor(image_scale_2_rgb, image_scale_2_gray, alpha, threshold)
    plot[0][1].imshow(corner_specified_image_scale_2)
    plot[0][1].set_title("Image:Corners(scale=2, alpha=" + str(alpha) + ", thre
shold=" + str(threshold) + ")")

    corner_specified_image_scale_3, feature_count_scale_3 = harris_corner_detec
tor(image_scale_3_rgb, image_scale_3_gray, alpha, threshold)
    plot[1][0].imshow(corner_specified_image_scale_3)
    plot[1][0].set_title("Image:Corners(scale=3, alpha=" + str(alpha) + ", thre
shold=" + str(threshold) + ")")

    corner_specified_image_scale_4, feature_count_scale_4 = harris_corner_detec
tor(image_scale_4_rgb, image_scale_4_gray, alpha, threshold)
    plot[1][1].imshow(corner_specified_image_scale_4)
    plot[1][1].set_title("Image:Corners(scale=4, alpha=" + str(alpha) + ", thre
shold=" + str(threshold) + ")")

    print("Scale = 1 -> Feature Count = " + str(feature_count_scale_1))
    print("Scale = 2 -> Feature Count = " + str(feature_count_scale_2))
    print("Scale = 3 -> Feature Count = " + str(feature_count_scale_3))
    print("Scale = 4 -> Feature Count = " + str(feature_count_scale_4) + "\n")
```

Feature Counts:
With Alpha = 0.04

With Threshold = 0.25
Scale = 1 -> Feature Count = 1141
Scale = 2 -> Feature Count = 10396
Scale = 3 -> Feature Count = 13957
Scale = 4 -> Feature Count = 3243

With Threshold = 0.5
Scale = 1 -> Feature Count = 17
Scale = 2 -> Feature Count = 18
Scale = 3 -> Feature Count = 1062
Scale = 4 -> Feature Count = 2987

With Threshold = 0.75
Scale = 1 -> Feature Count = 6
Scale = 2 -> Feature Count = 6
Scale = 3 -> Feature Count = 14
Scale = 4 -> Feature Count = 28

With Alpha = 0.05

With Threshold = 0.25
Scale = 1 -> Feature Count = 220825
Scale = 2 -> Feature Count = 57090
Scale = 3 -> Feature Count = 13965
Scale = 4 -> Feature Count = 3245

With Threshold = 0.5
Scale = 1 -> Feature Count = 20
Scale = 2 -> Feature Count = 19
Scale = 3 -> Feature Count = 13668
Scale = 4 -> Feature Count = 3189

With Threshold = 0.75
Scale = 1 -> Feature Count = 6
Scale = 2 -> Feature Count = 6
Scale = 3 -> Feature Count = 18
Scale = 4 -> Feature Count = 40

With Alpha = 0.06

With Threshold = 0.25
Scale = 1 -> Feature Count = 233085
Scale = 2 -> Feature Count = 57440
Scale = 3 -> Feature Count = 13974
Scale = 4 -> Feature Count = 3246

With Threshold = 0.5
Scale = 1 -> Feature Count = 26
Scale = 2 -> Feature Count = 28
Scale = 3 -> Feature Count = 13857
Scale = 4 -> Feature Count = 3210

With Threshold = 0.75
Scale = 1 -> Feature Count = 6
Scale = 2 -> Feature Count = 8
Scale = 3 -> Feature Count = 41
Scale = 4 -> Feature Count = 54

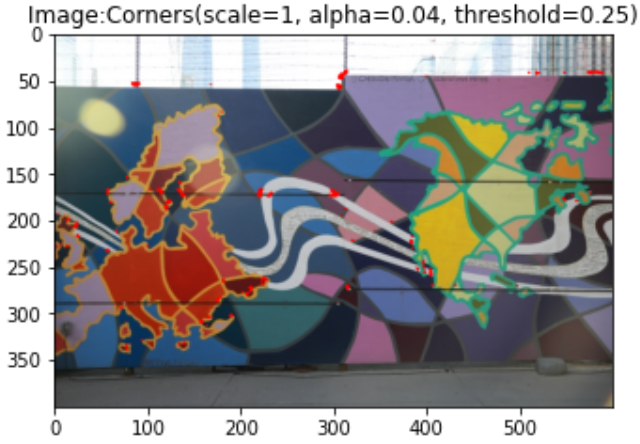


Image:Corners(scale=1, alpha=0.04, threshold=0.25)

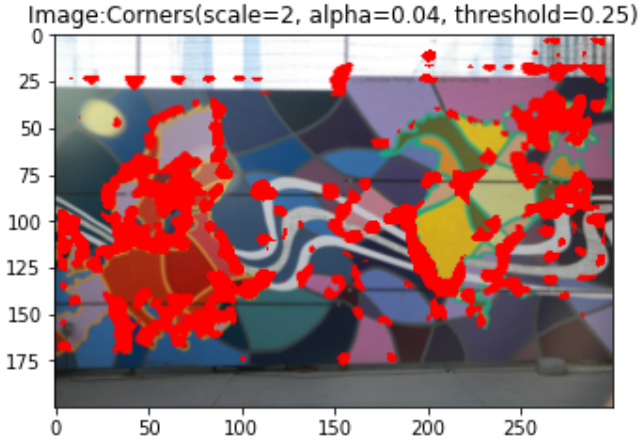


Image:Corners(scale=2, alpha=0.04, threshold=0.25)

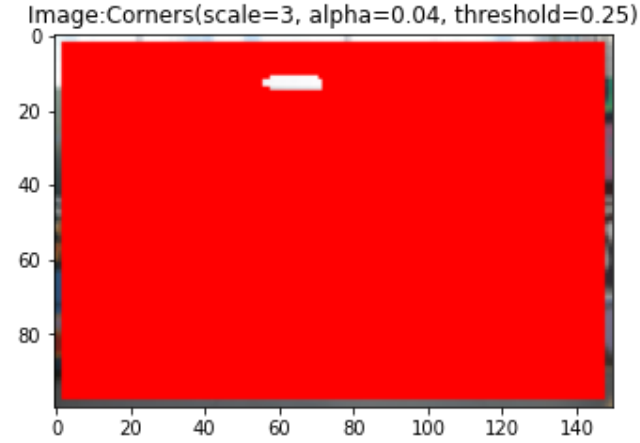


Image:Corners(scale=3, alpha=0.04, threshold=0.25)

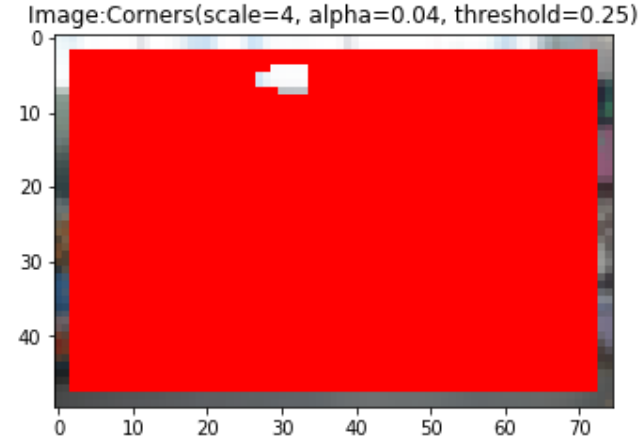


Image:Corners(scale=4, alpha=0.04, threshold=0.25)

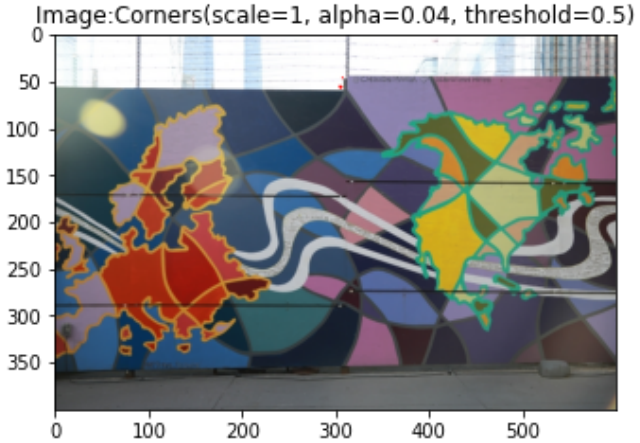


Image:Corners(scale=1, alpha=0.04, threshold=0.5)

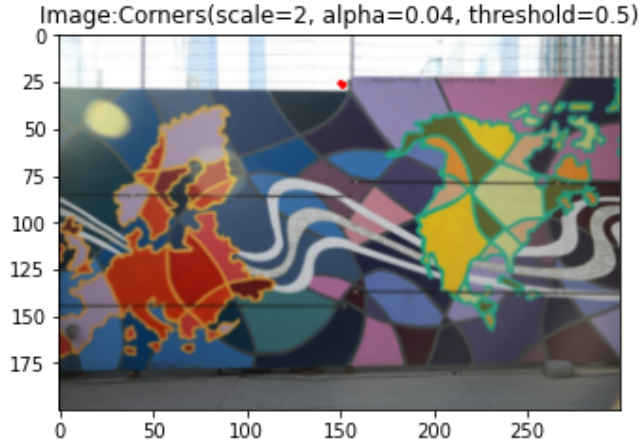


Image:Corners(scale=2, alpha=0.04, threshold=0.5)

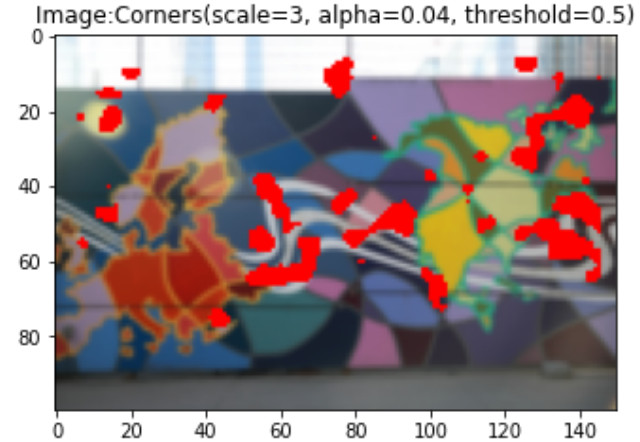


Image:Corners(scale=3, alpha=0.04, threshold=0.5)

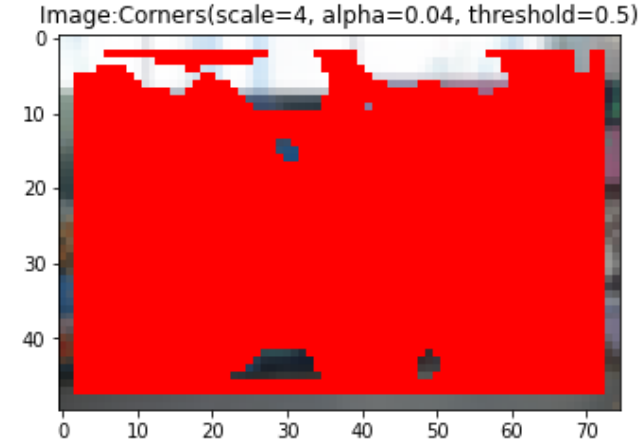


Image:Corners(scale=4, alpha=0.04, threshold=0.5)

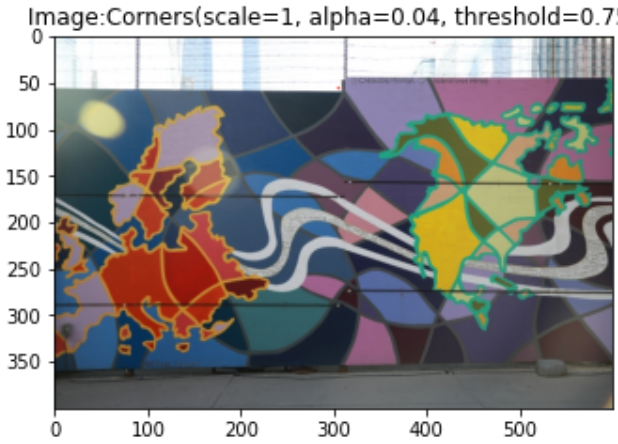


Image:Corners(scale=1, alpha=0.04, threshold=0.75)

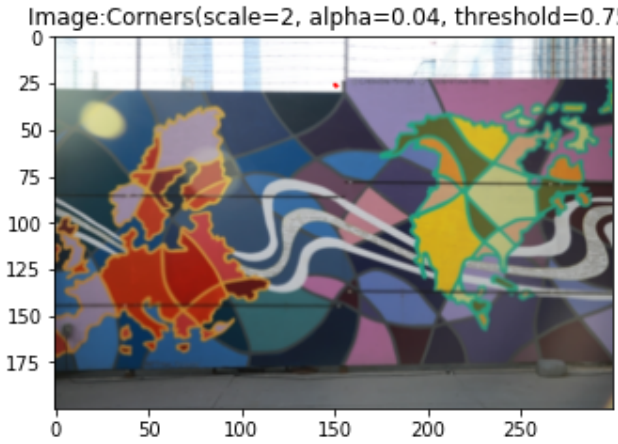


Image:Corners(scale=2, alpha=0.04, threshold=0.75)

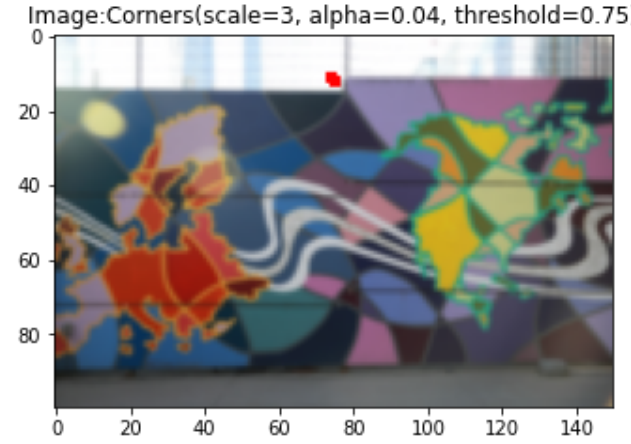


Image:Corners(scale=3, alpha=0.04, threshold=0.75)

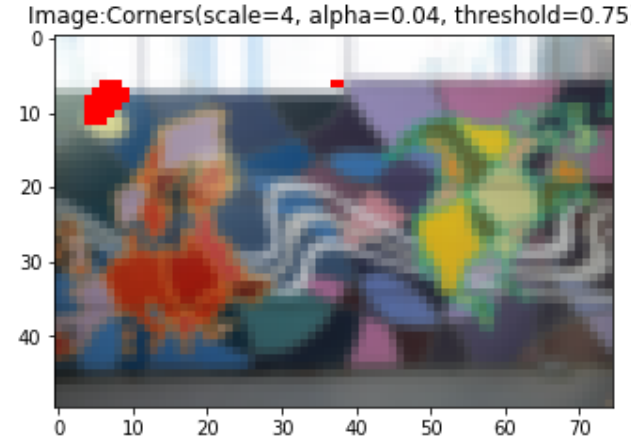


Image:Corners(scale=4, alpha=0.04, threshold=0.75)

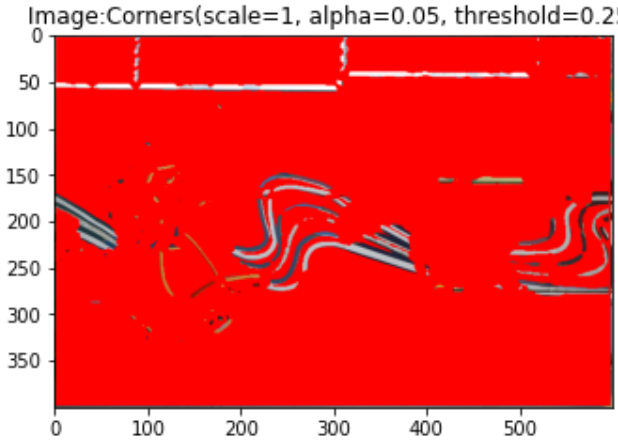


Image:Corners(scale=1, alpha=0.05, threshold=0.25)

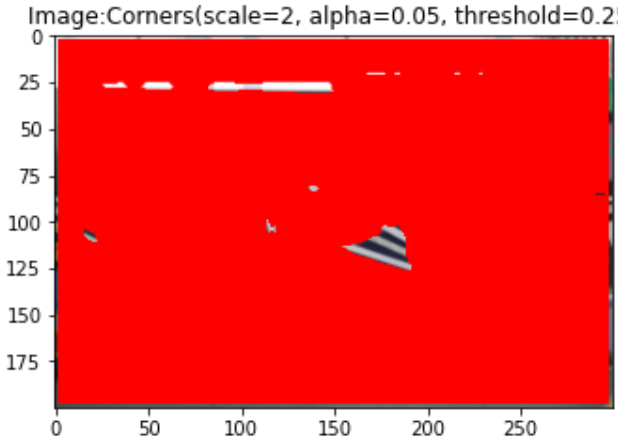


Image:Corners(scale=2, alpha=0.05, threshold=0.25)

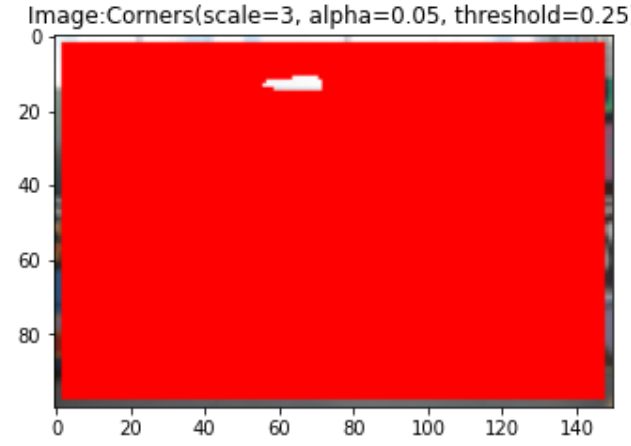


Image:Corners(scale=3, alpha=0.05, threshold=0.25)

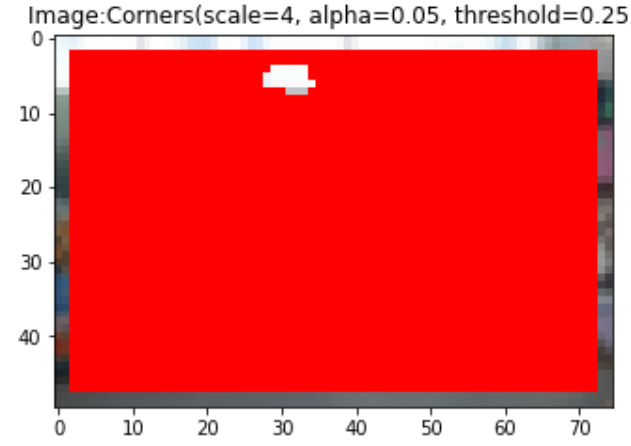


Image:Corners(scale=4, alpha=0.05, threshold=0.25)

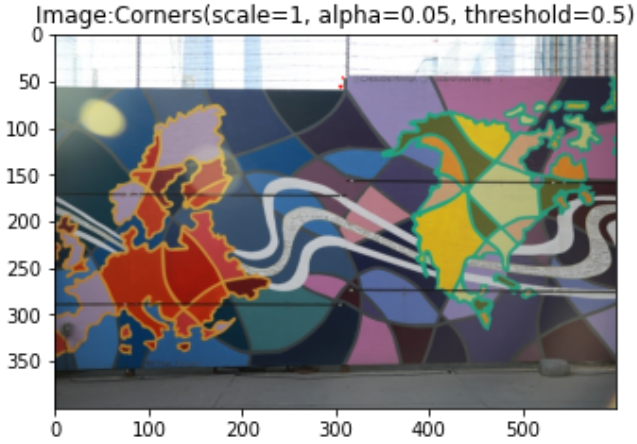


Image:Corners(scale=1, alpha=0.05, threshold=0.5)

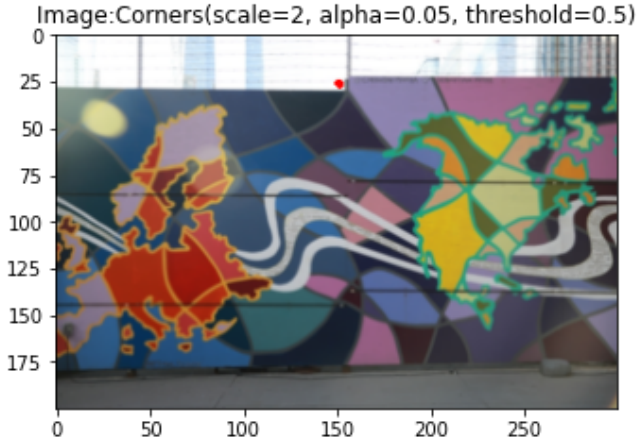


Image:Corners(scale=2, alpha=0.05, threshold=0.5)

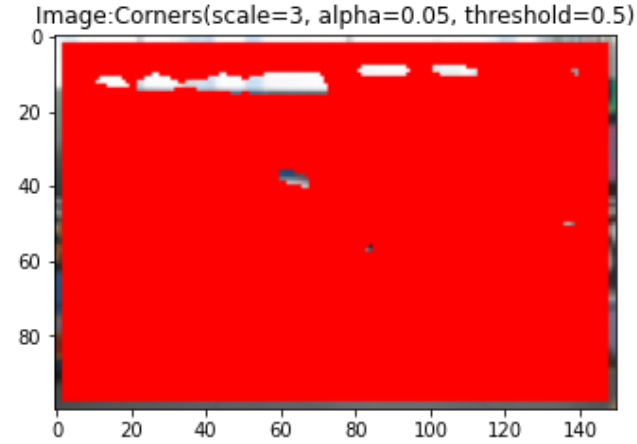


Image:Corners(scale=3, alpha=0.05, threshold=0.5)

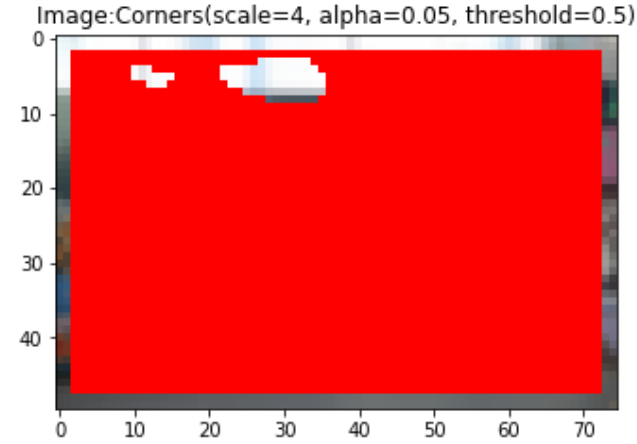


Image:Corners(scale=4, alpha=0.05, threshold=0.5)

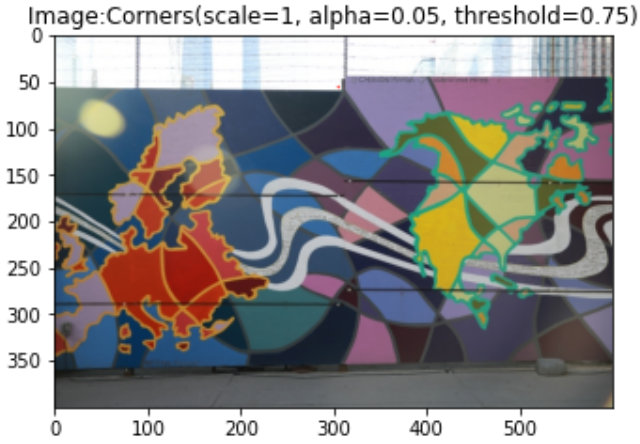


Image:Corners(scale=1, alpha=0.05, threshold=0.75)

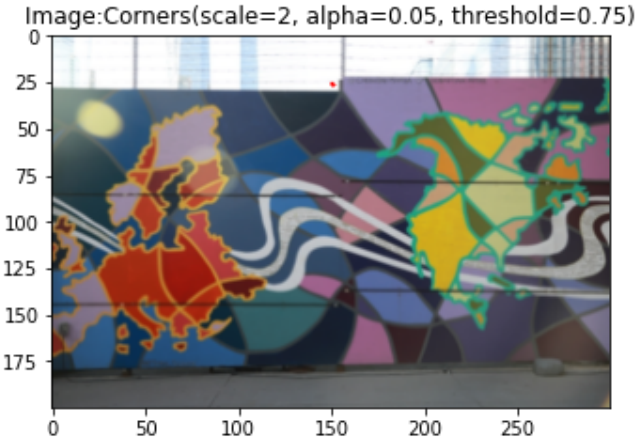


Image:Corners(scale=2, alpha=0.05, threshold=0.75)

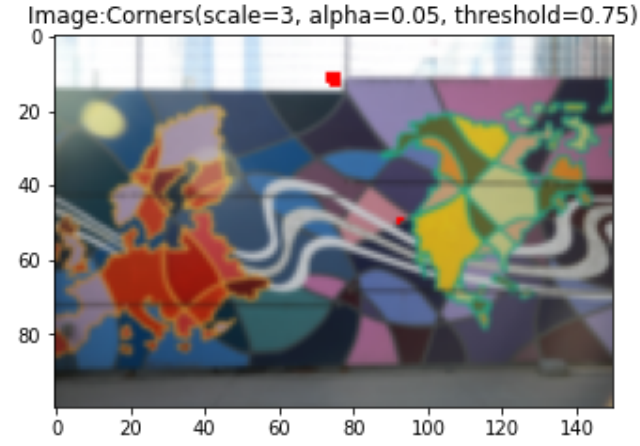


Image:Corners(scale=3, alpha=0.05, threshold=0.75)

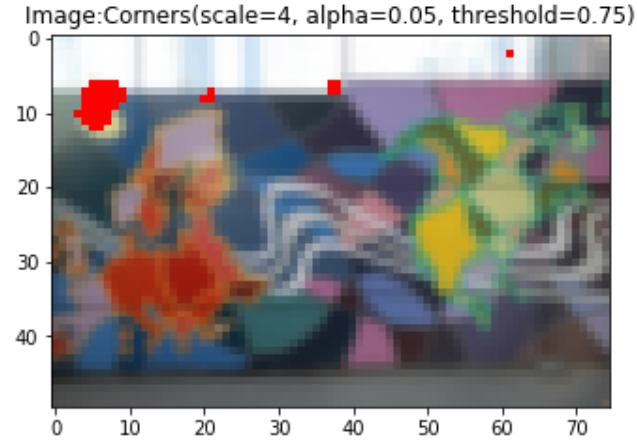


Image:Corners(scale=4, alpha=0.05, threshold=0.75)

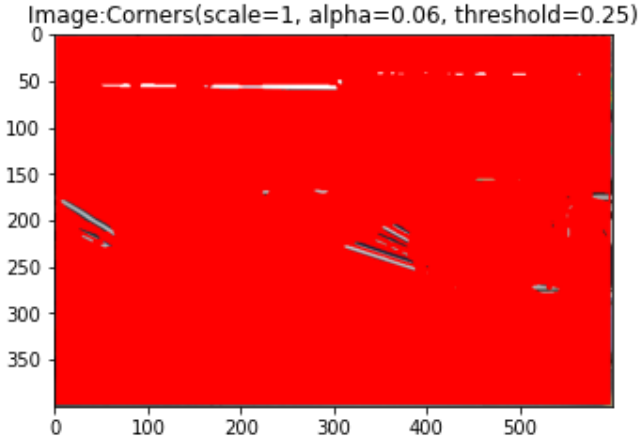


Image:Corners(scale=1, alpha=0.06, threshold=0.25)

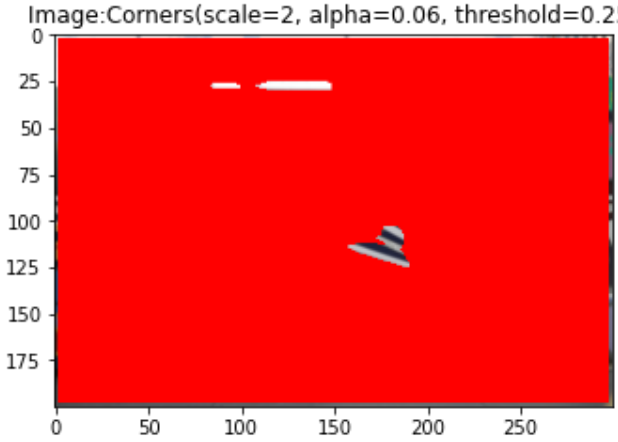


Image:Corners(scale=2, alpha=0.06, threshold=0.25)

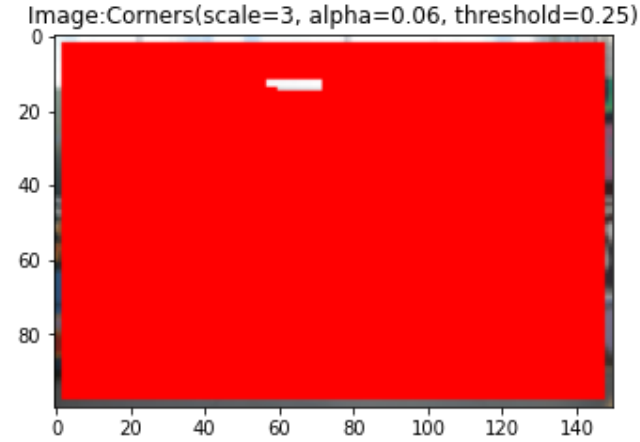


Image:Corners(scale=3, alpha=0.06, threshold=0.25)

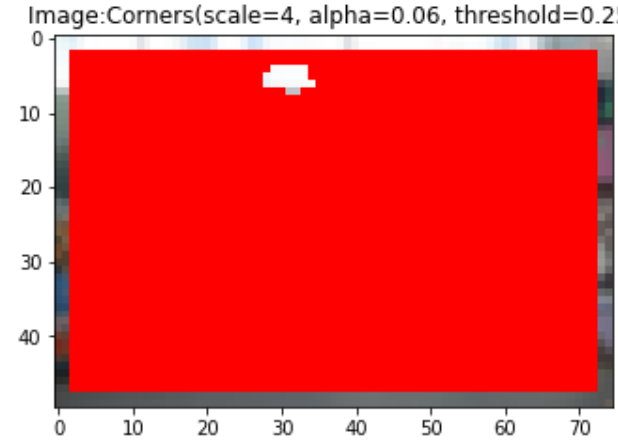


Image:Corners(scale=4, alpha=0.06, threshold=0.25)

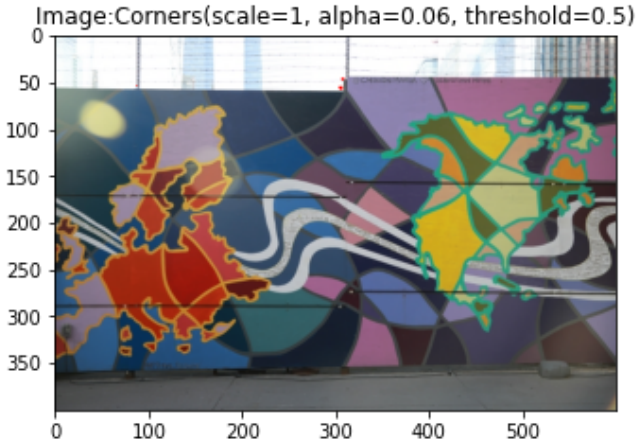


Image:Corners(scale=1, alpha=0.06, threshold=0.5)

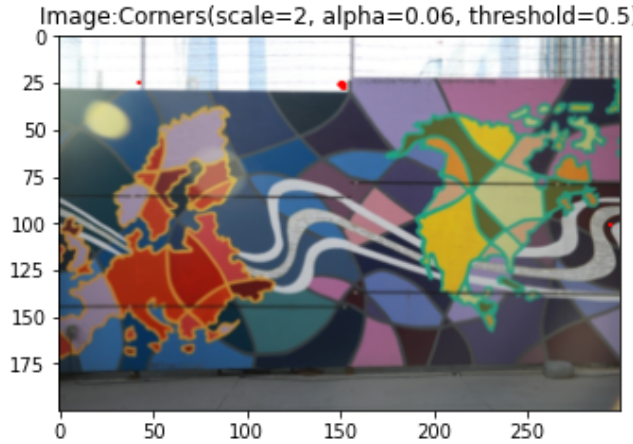


Image:Corners(scale=2, alpha=0.06, threshold=0.5)

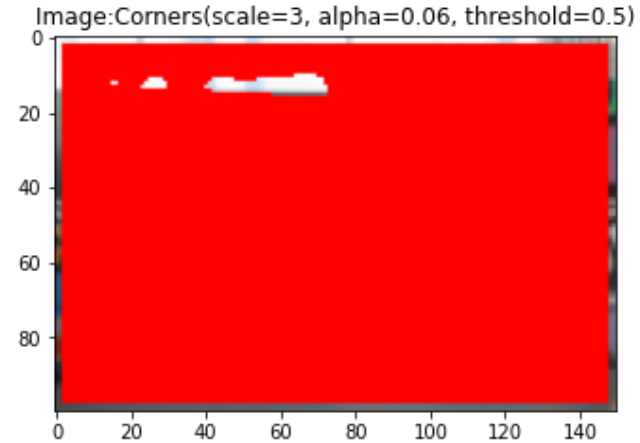


Image:Corners(scale=3, alpha=0.06, threshold=0.5)

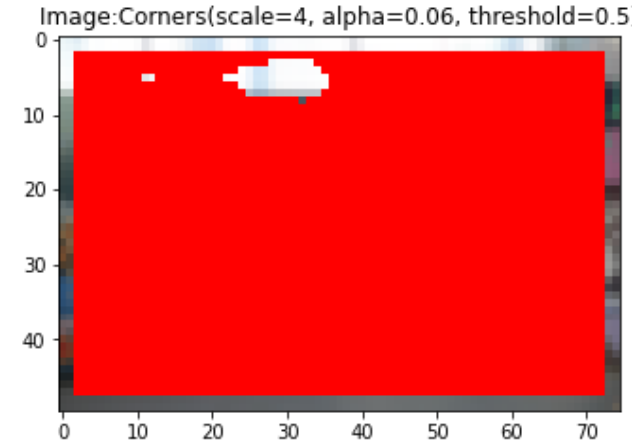


Image:Corners(scale=4, alpha=0.06, threshold=0.5)

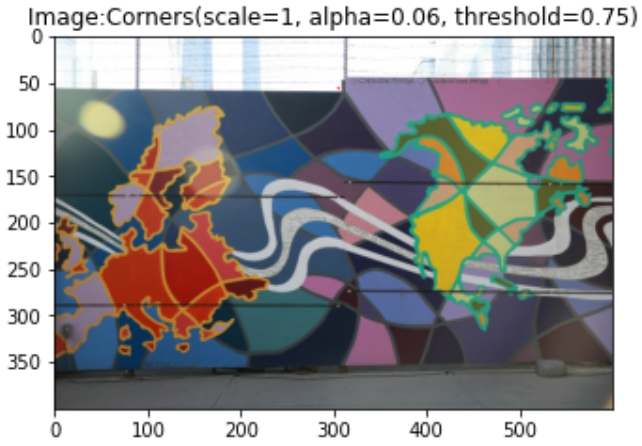


Image:Corners(scale=1, alpha=0.06, threshold=0.75)
0
50
100
150
200
250
300
350
0
100
200
300
400
500

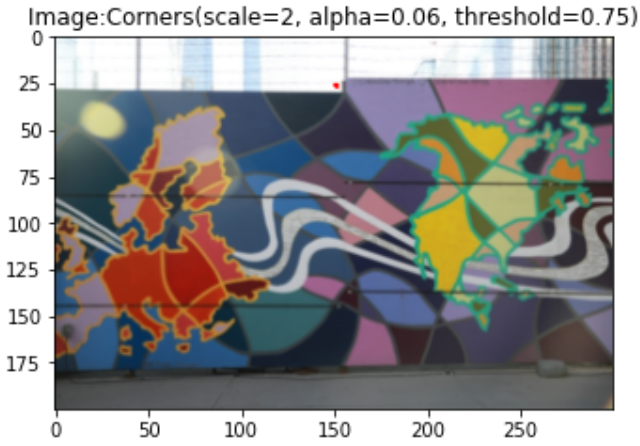


Image:Corners(scale=2, alpha=0.06, threshold=0.75)
0
25
50
75
100
125
150
175
0
50
100
150
200
250

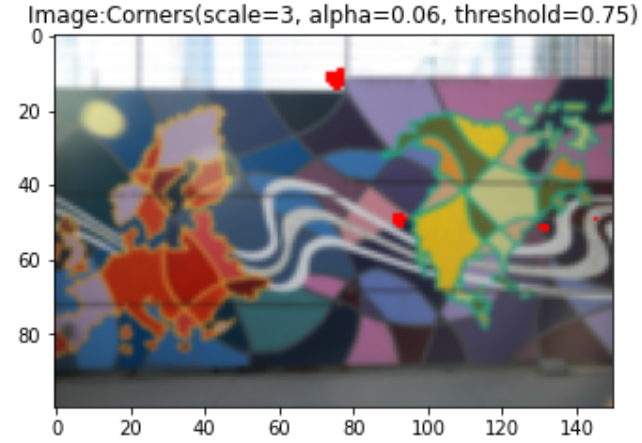


Image:Corners(scale=3, alpha=0.06, threshold=0.75)
0
20
40
60
80
0
20
40
60
80
100
120
140

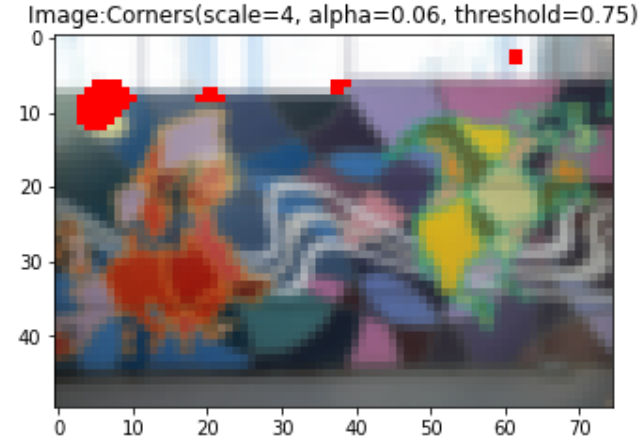


Image:Corners(scale=4, alpha=0.06, threshold=0.75)
0
10
20
30
40
0
10
20
30
40
50
60
70

In [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from google.colab import drive
```

In [2]:
```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
ive.mount("/content/drive", force_remount=True).
```

In [20]:
```
img_sl = cv2.imread("/content/drive/MyDrive/Images/7/sl.jpg")
img_sl = cv2.cvtColor(img_sl, cv2.COLOR_BGR2RGB)
img_sm = cv2.imread("/content/drive/MyDrive/Images/7/sm.jpg")
img_sm = cv2.cvtColor(img_sm, cv2.COLOR_BGR2RGB)
img_sr = cv2.imread("/content/drive/MyDrive/Images/7/sr.jpg")
img_sr = cv2.cvtColor(img_sr, cv2.COLOR_BGR2RGB)

img_1 = cv2.imread("/content/drive/MyDrive/Images/7/11.jpg")
img_1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB)
img_2 = cv2.imread("/content/drive/MyDrive/Images/7/22.jpg")
img_2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB)
img_3 = cv2.imread("/content/drive/MyDrive/Images/7/33.jpg")
img_3 = cv2.cvtColor(img_3, cv2.COLOR_BGR2RGB)
```

In [4]:
```
def implement_feature_selection_algorithm(image_left, image_right):
  sift = cv2.xfeatures2d.SIFT_create()

  key_points_left, descriptor_left = sift.detectAndCompute(image_left, None)
  key_points_right, descriptor_right = sift.detectAndCompute(image_right, None)

  return key_points_left, descriptor_left, key_points_right, descriptor_right
```

In [5]:
```
def create_drawn_key_points_image(image_left, image_right):
  key_points_left, descriptor_left, key_points_right, descriptor_right = implem
ent_feature_selection_algorithm(image_left, image_right)

  drawn_key_points_image_left = cv2.drawKeypoints(image_left, key_points_left,
None)
  drawn_key_points_image_right = cv2.drawKeypoints(image_right, key_points_righ
t, None)

  return drawn_key_points_image_left, drawn_key_points_image_right
```

In [6]:
```
def find_matches_of_two_image(descriptor_left, descriptor_right, image_left, im
age_right):
  # Re-use Created Function By tranleanh

  ratio = 0.85
  min_match = 10

  matcher = cv2.BFMatcher()
  raw_matches = matcher.knnMatch(descriptor_left, descriptor_right, k=2)

  good_points = []
  good_matches = []

  for m1, m2 in raw_matches:
    if m1.distance < ratio * m2.distance:
      good_points.append((m1.trainIdx, m1.queryIdx))
      good_matches.append([m1])

  return good_points, good_matches
```

In [7]:
```
def create_drawn_matched_key_points_image(image_left, image_right):
  key_points_left, descriptor_left, key_points_right, descriptor_right = implem
ent_feature_selection_algorithm(image_left, image_right)
  good_points, good_matches = find_matches_of_two_image(descriptor_left, descri
ptor_right, image_left, image_right)

  drawn_matched_key_points_image = cv2.drawMatchesKnn(image_left, key_points_le
ft, image_right, key_points_right, good_matches, None, flags=2)

  return drawn_matched_key_points_image
```

In [8]:
```
def find_homography_matrix(good_points, key_points_left, key_points_right):
  # Re-use Created Function By tranleanh
  min_match = 10

  image_left_key_points = np.float32([key_points_left[i].pt for (_, i) in good_
points])
  image_right_key_points = np.float32([key_points_right[i].pt for (i, _) in goo
d_points])

  # Using RANSAC
  homography_matrix, status = cv2.findHomography(image_right_key_points, image_
left_key_points, cv2.RANSAC, 5.0)

  return homography_matrix
```

In [9]:
```
def panorama_shape(image_left, image_right):
  R_left, C_left, ch = image_left.shape
  R_right, C_right, ch = image_right.shape

  R_panaroma = R_left
  C_panorama = C_left + C_right

  return R_panaroma, C_panorama
```

In [10]:
```
def get_mask(image_left, image_right, left_flag):
  # Re-use Created Function By tranleanh

  smoothing_window_size = 600

  R_panaroma, C_panorama = panorama_shape(image_left, image_right)

  offset = smoothing_window_size // 2
  barrier = image_left.shape[1] - smoothing_window_size // 2

  mask = np.zeros((R_panaroma, C_panorama))

  if left_flag:
    mask[:, barrier - offset:barrier + offset ] = np.tile(np.linspace(1, 0, 2 *
offset ).T, (R_panaroma, 1))
    mask[:, :barrier - offset] = 1
  else:
    mask[:, barrier - offset :barrier + offset ] = np.tile(np.linspace(0, 1, 2
* offset ).T, (R_panaroma, 1))
    mask[:, barrier + offset:] = 1

  mask = cv2.merge([mask, mask, mask])

  return mask
```

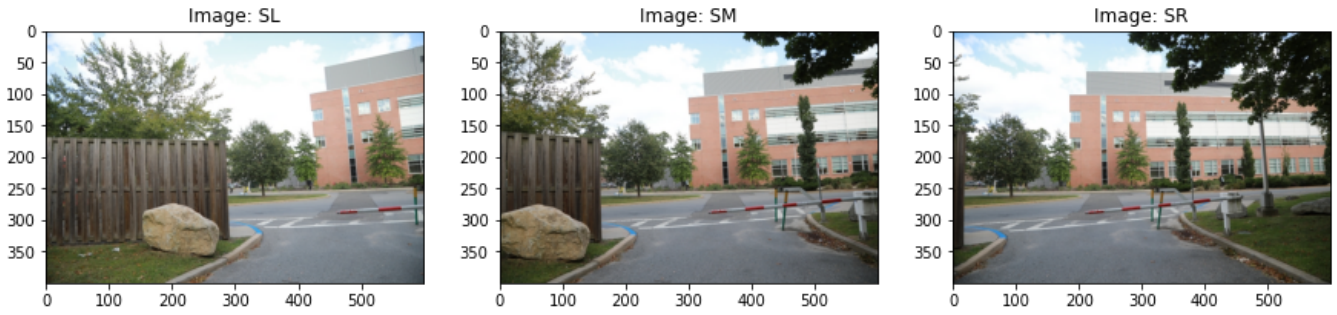
In [11]:
```
def stitch_images(image_left, image_right, homography_matrix):
  # Re-use Created Function By tranleanh

  R_panaroma, C_panorama = panorama_shape(image_left, image_right)

  panorama_left = np.zeros((R_panaroma, C_panorama, 3))

  mask_left = get_mask(image_left, image_right, True)
  panorama_left[0:image_left.shape[0], 0:image_left.shape[1], :] = image_left
  panorama_left *= mask_left

  mask_right = get_mask(image_left, image_right, False)
  panorama_right = cv2.warpPerspective(image_right, homography_matrix, (C_panor
ama, R_panaroma)) * mask_right
  result = panorama_left + panorama_right

  rows, cols = np.where(result[:, :, 0] != 0)
  min_row, max_row = min(rows), max(rows) + 1
  min_col, max_col = min(cols), max(cols) + 1
  final_result = result[min_row:max_row, min_col:max_col, :]

  return final_result.astype(np.uint8)
```

In [12]:
```
def get_panorama_result(image_left,image_right):
  key_points_left, descriptor_left, key_points_right, descriptor_right = implem
ent_feature_selection_algorithm(image_left, image_right)
  good_points, good_matches = find_matches_of_two_image(descriptor_left, descri
ptor_right, image_left, image_right)
  homography_matrix = find_homography_matrix(good_points, key_points_left, key_
points_right)
  panorama_result = stitch_images(image_left, image_right, homography_matrix)

  return panorama_result
```

In [13]:
```
image_sl_sm = get_panorama_result(img_sl, img_sm)
image_sm_sr = get_panorama_result(img_sm, img_sr)
image_panorama = get_panorama_result(image_sl_sm, image_sm_sr)
```

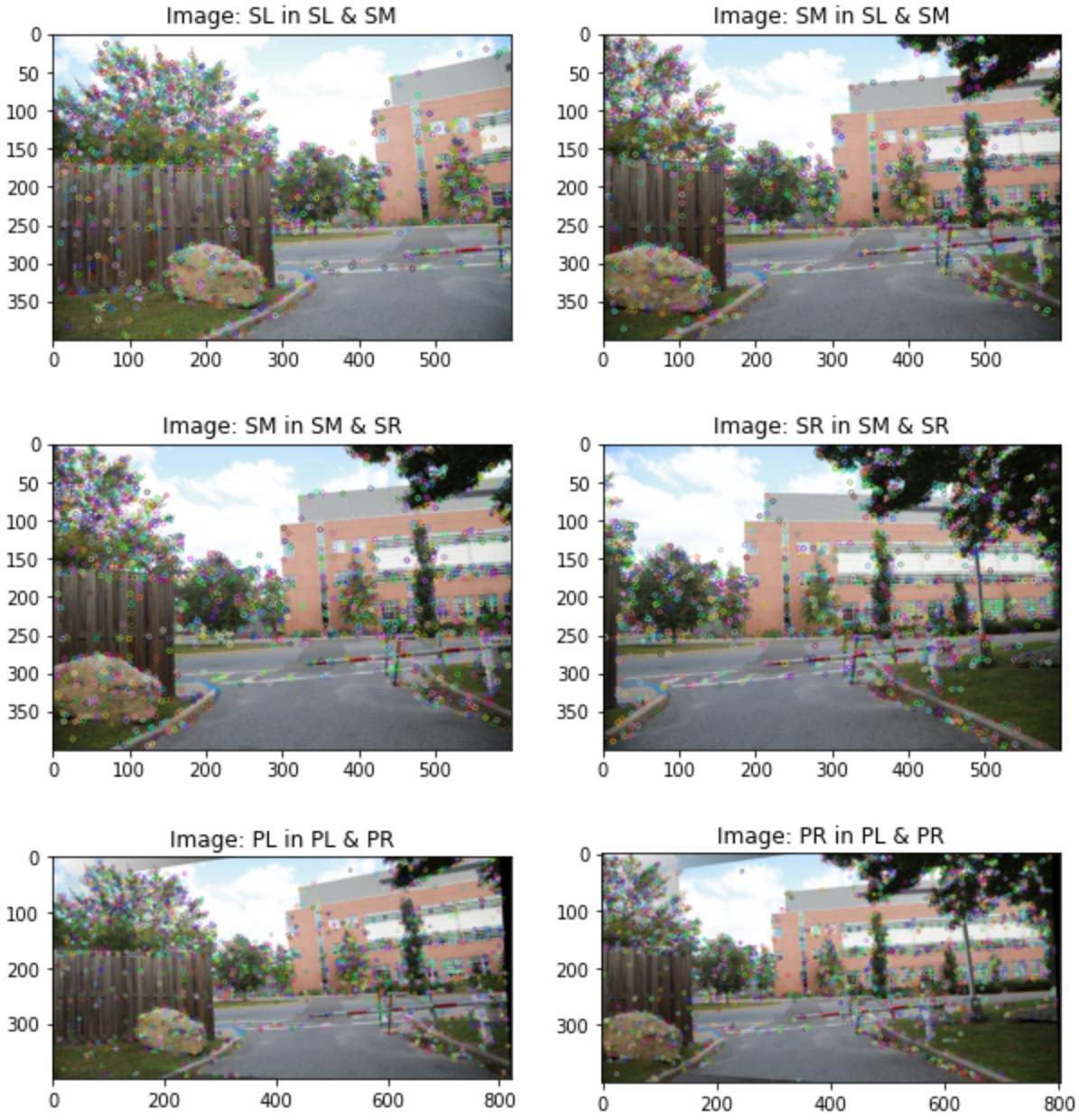
In [14]:
```
fig, plot = plt.subplots(1, 3, figsize = (15, 5))

plot[0].imshow(img_sl)
plot[0].set_title("Image: SL")

plot[1].imshow(img_sm)
plot[1].set_title("Image: SM")

plot[2].imshow(img_sr)
plot[2].set_title("Image: SR")
```

Out[14]:
```
Text(0.5, 1.0, 'Image: SR')
```



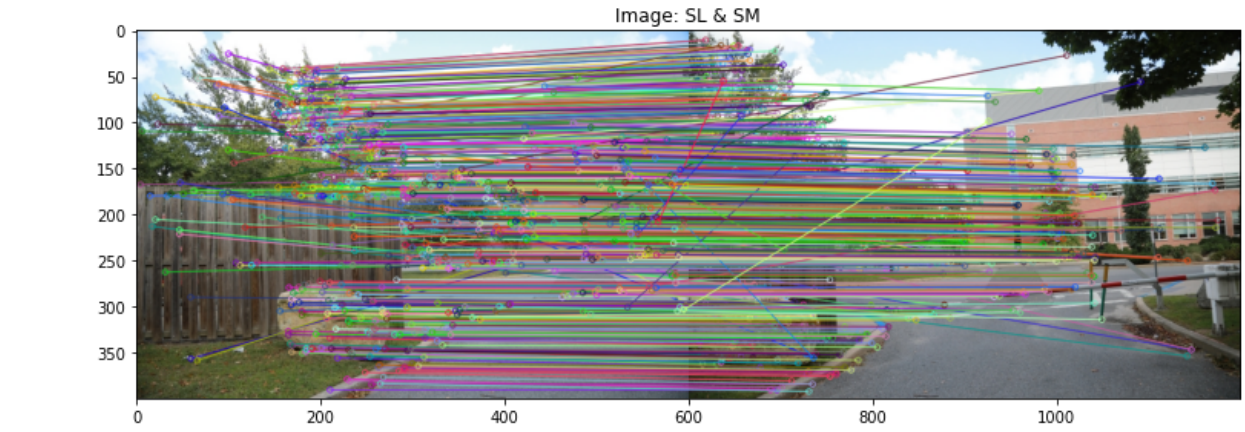
In [15]:
```
# Selected Features

sl_in_sl_and_sm, sm_in_sl_and_sm = create_drawn_key_points_image(img_sl, img_s
m)
sm_in_sm_and_sr, sr_in_sm_and_sr = create_drawn_key_points_image(img_sm, img_s
r)
pl_in_pl_and_pr, pr_in_pl_and_pr = create_drawn_key_points_image(image_sl_sm, i
mage_sm_sr)

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(sl_in_sl_and_sm)
plot[0].set_title("Image: SL in SL & SM")

plot[1].imshow(sm_in_sl_and_sm)
plot[1].set_title("Image: SM in SL & SM")

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(sm_in_sm_and_sr)
plot[0].set_title("Image: SM in SM & SR")

plot[1].imshow(sr_in_sm_and_sr)
plot[1].set_title("Image: SR in SM & SR")

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(pl_in_pl_and_pr)
plot[0].set_title("Image: PL in PL & PR")

plot[1].imshow(pr_in_pl_and_pr)
plot[1].set_title("Image: PR in PL & PR")
```

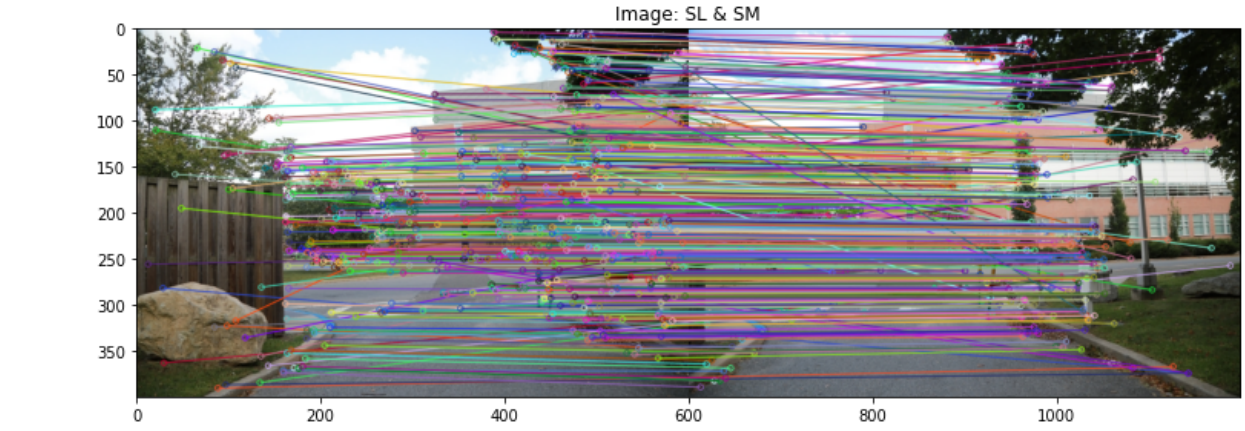
Out[15]: Text(0.5, 1.0, 'Image: PR in PL & PR')

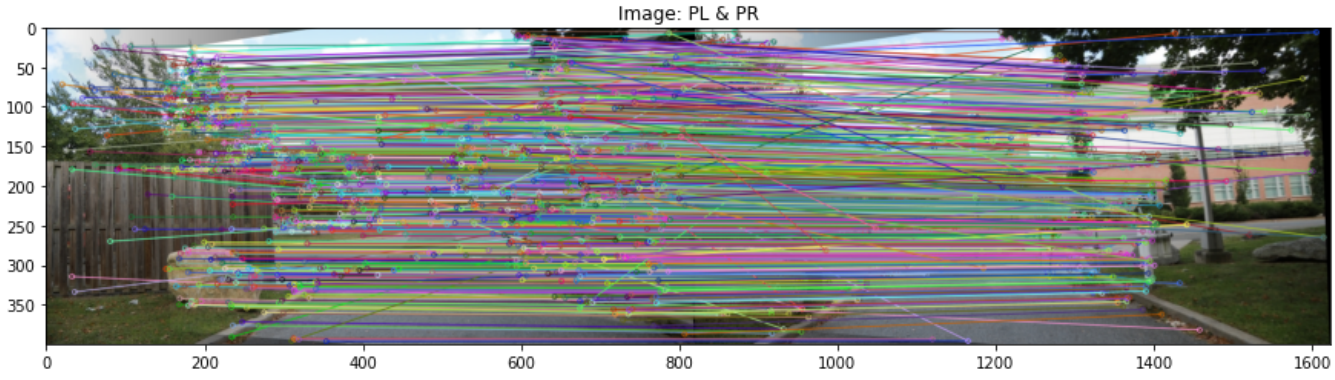


In [16]:
```
# Matched Features

sl_and_sm = create_drawn_matched_key_points_image(img_sl, img_sm)
sm_and_sr = create_drawn_matched_key_points_image(img_sm, img_sr)
pl_and_pr = create_drawn_matched_key_points_image(image_sl_sm, image_sm_sr)

fig, plot = plt.subplots(3, 1, figsize = (15, 15))

plot[0].imshow(sl_and_sm)
plot[0].set_title("Image: SL & SM")

plot[1].imshow(sm_and_sr)
plot[1].set_title("Image: SL & SM")

plot[2].imshow(pl_and_pr)
plot[2].set_title("Image: PL & PR")
```

Out[16]:
```
Text(0.5, 1.0, 'Image: PL & PR')
```

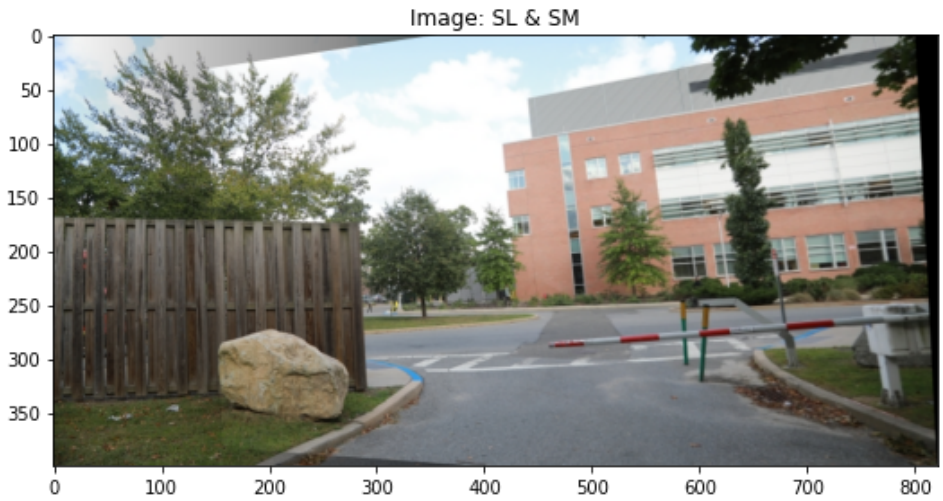


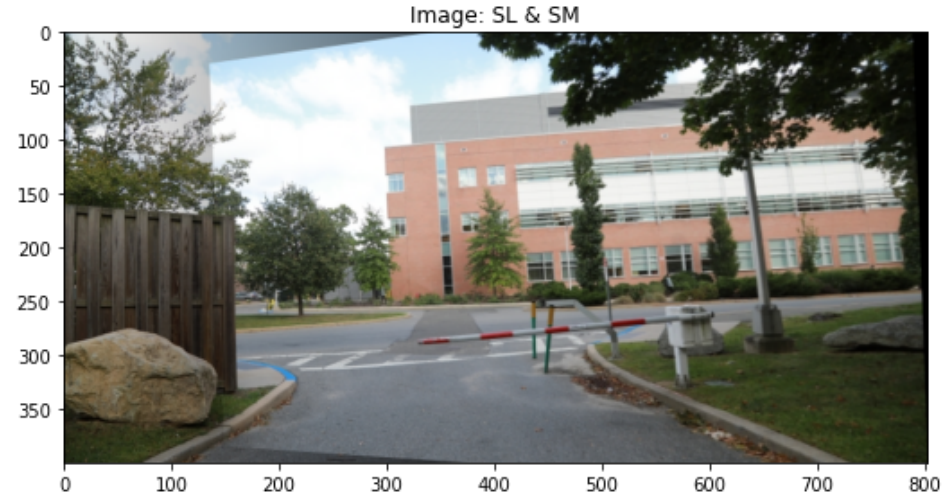


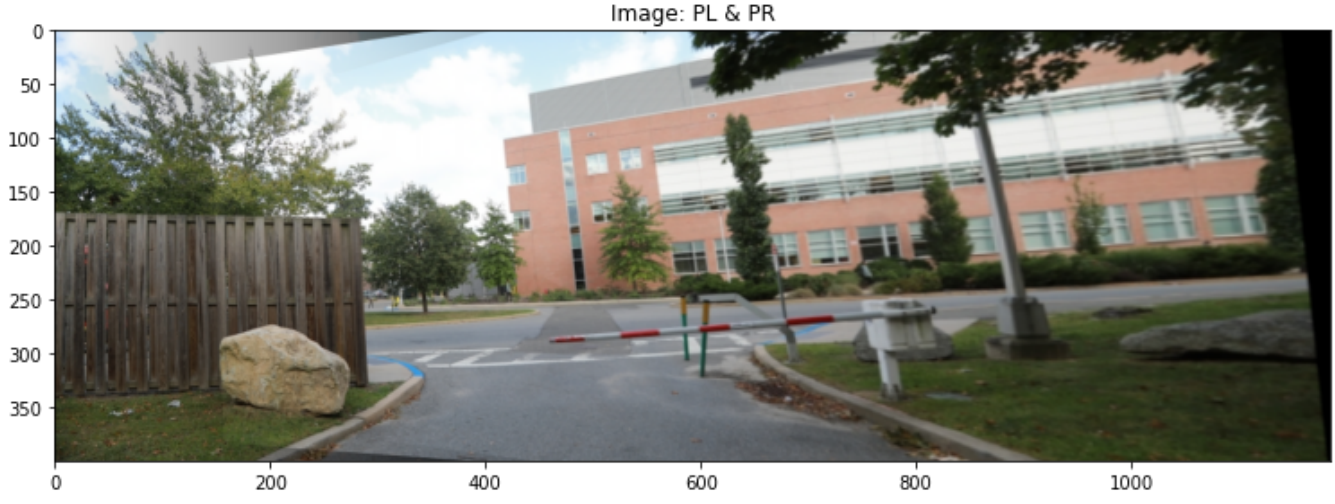


In [17]:
```
# Stitched Image

fig, plot = plt.subplots(3, 1, figsize = (15, 15))

plot[0].imshow(image_sl_sm / 255.0)
plot[0].set_title("Image: SL & SM")

plot[1].imshow(image_sm_sr / 255.0)
plot[1].set_title("Image: SL & SM")

plot[2].imshow(image_panorama / 255.0)
plot[2].set_title("Image: PL & PR")
```

Out[17]:
```
Text(0.5, 1.0, 'Image: PL & PR')
```



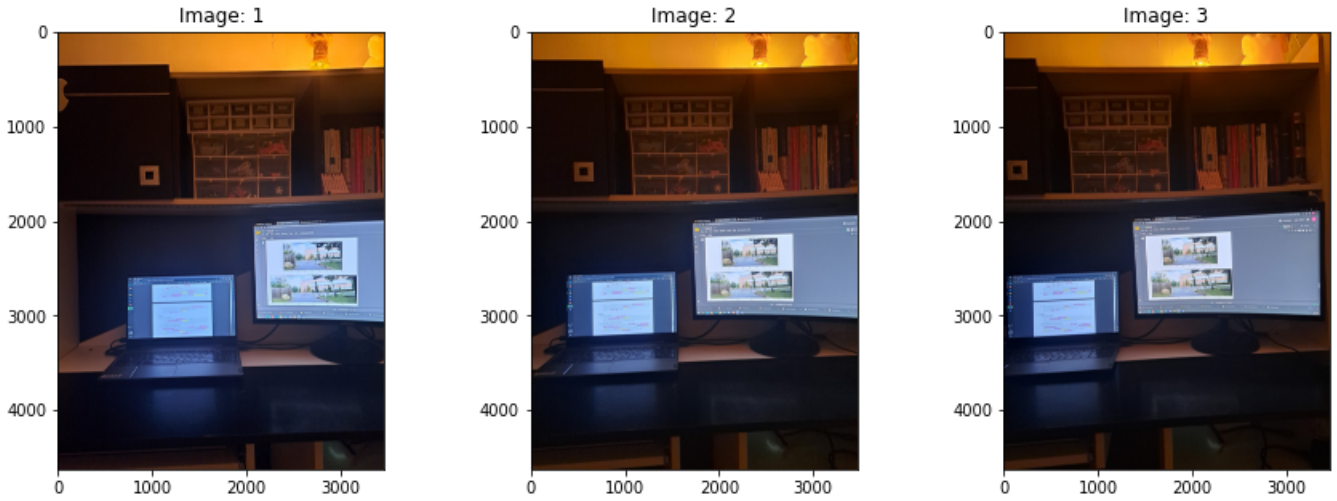




In [21]:
```
image_1_2 = get_panorama_result(img_1, img_2)
image_2_3 = get_panorama_result(img_2, img_3)
image_1_2_3 = get_panorama_result(image_1_2, image_2_3)
```

In [26]:
```
# Original Images

fig, plot = plt.subplots(1, 3, figsize = (15, 5))

plot[0].imshow(img_1)
plot[0].set_title("Image: 1")

plot[1].imshow(img_2)
plot[1].set_title("Image: 2")

plot[2].imshow(img_3)
plot[2].set_title("Image: 3")
```

Out[26]:
```
Text(0.5, 1.0, 'Image: 3')
```



In [25]:

```
# Stitched Images

fig, plot = plt.subplots(3, 1, figsize = (10, 25))

plot[0].imshow(image_1_2 / 255.0)
plot[0].set_title("Image: 1 & 2")

plot[1].imshow(image_2_3 / 255.0)
plot[1].set_title("Image: 2 & 3")

plot[2].imshow(image_1_2_3 / 255.0)
plot[2].set_title("Image: 1 & 2 & 3")
```

Out[25]: Text(0.5, 1.0, 'Image: 1 & 2 & 3')

Image: 1 & 2

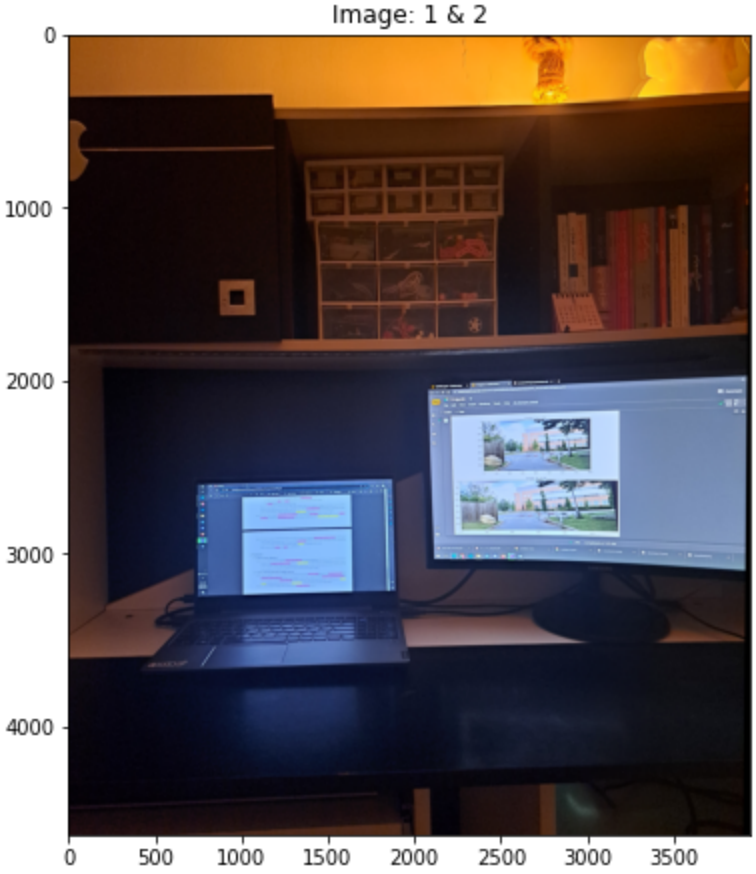


Image: 2 & 3

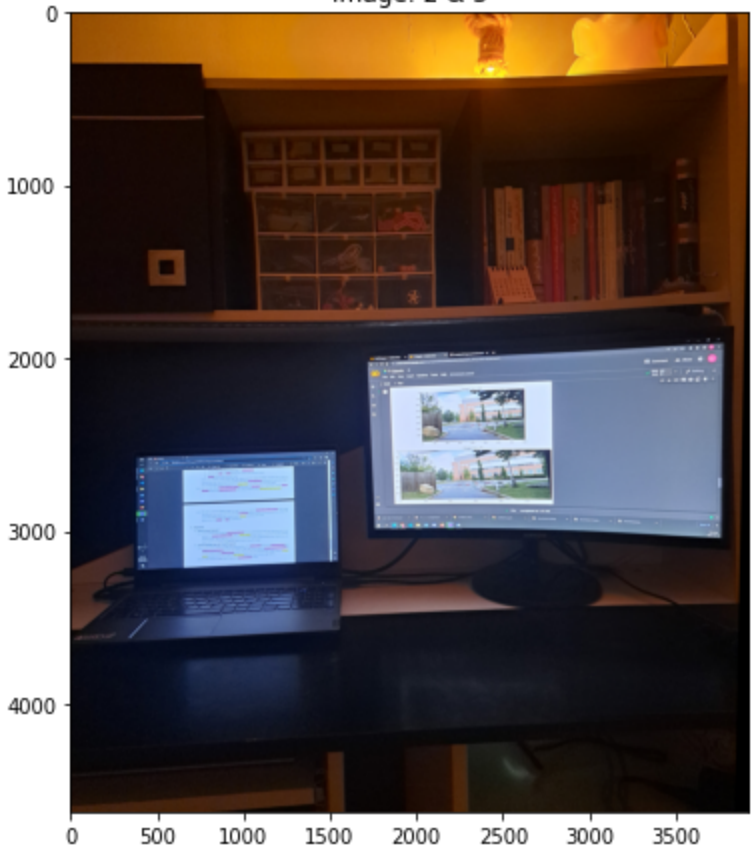
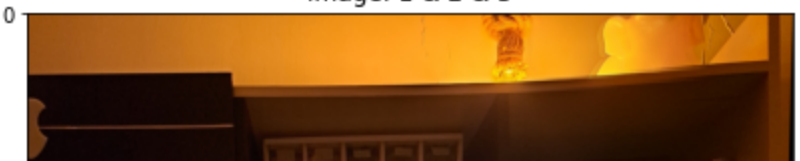
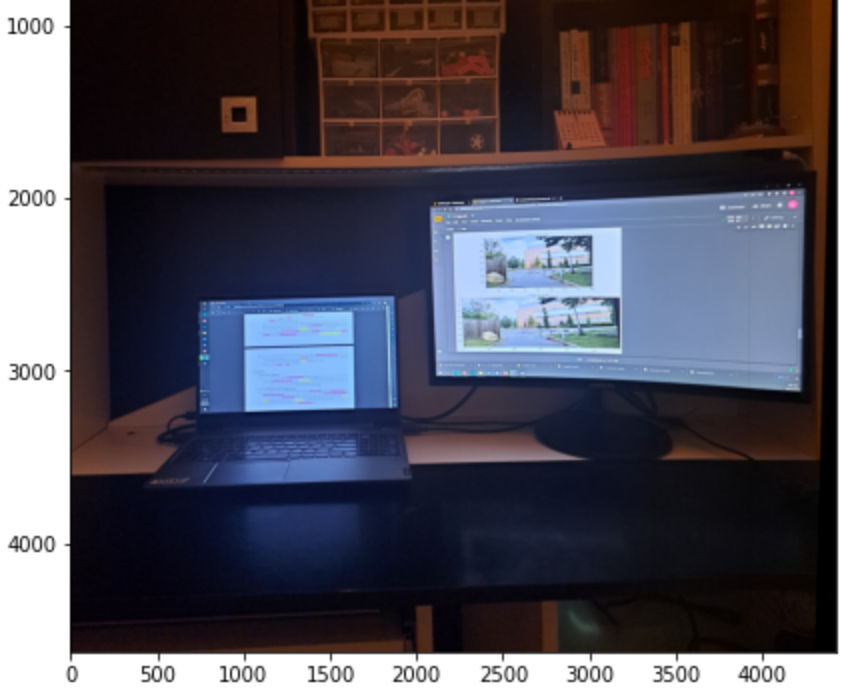


Image: 1 & 2 & 3




1000
2000
3000
4000
0
500
1000
1500
2000
2500
3000
3500
4000