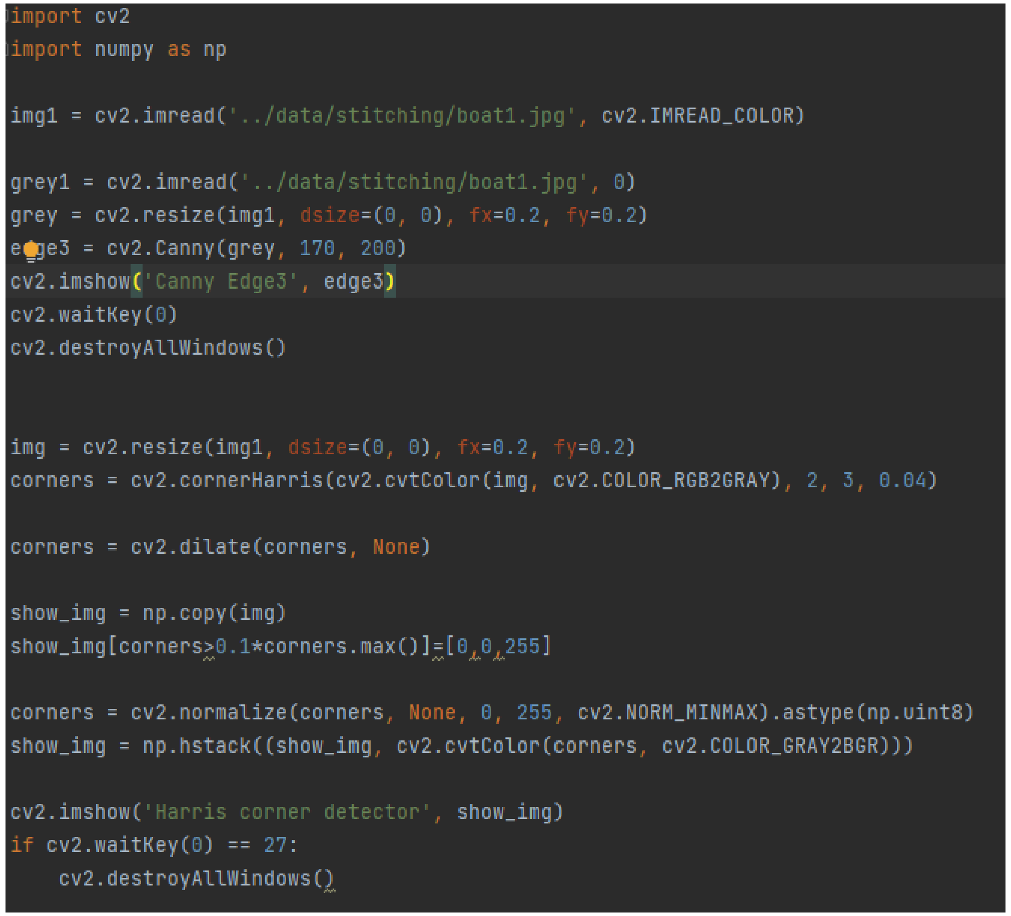
산업 컴퓨터비젼 실제 프로그래밍 과제 #2

산업인공지능학과

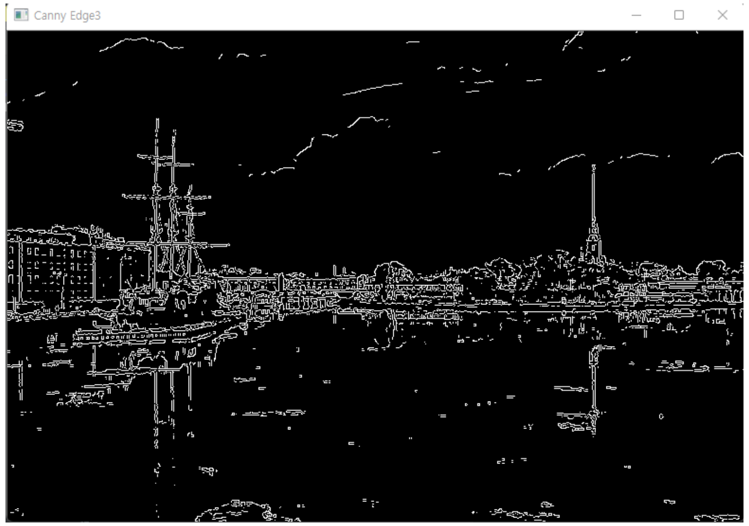
2020254018 강윤구

1) Feature Detection

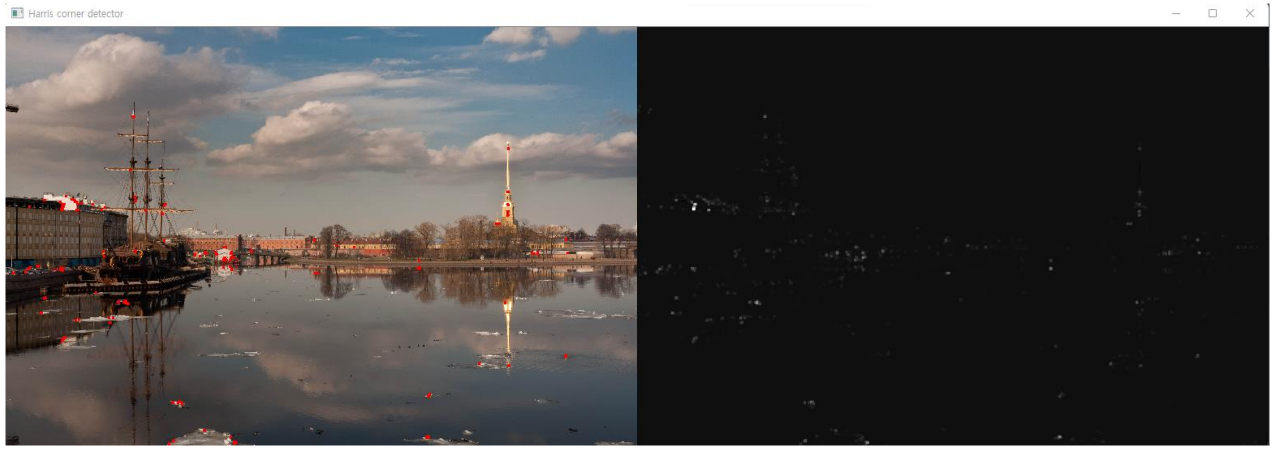


```
import cv2
import numpy as np

img1 = cv2.imread('../data/stitching/boat1.jpg', cv2.IMREAD_COLOR)

grey1 = cv2.imread('../data/stitching/boat1.jpg', 0)
grey = cv2.resize(img1, dsize=(0, 0), fx=0.2, fy=0.2)
edge3 = cv2.Canny(grey, 170, 200)
cv2.imshow('Canny Edge3', edge3)
cv2.waitKey(0)
cv2.destroyAllWindows()


img = cv2.resize(img1, dsize=(0, 0), fx=0.2, fy=0.2)
corners = cv2.cornerHarris(cv2.cvtColor(img, cv2.COLOR_RGB2GRAY), 2, 3, 0.04)

corners = cv2.dilate(corners, None)

show_img = np.copy(img)
show_img[corners>0.1*corners.max()]=[0,0,255]

corners = cv2.normalize(corners, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
show_img = np.hstack((show_img, cv2.cvtColor(corners, cv2.COLOR_GRAY2BGR)))

cv2.imshow('Harris corner detector', show_img)
if cv2.waitKey(0) == 27:
    cv2.destroyAllWindows()
```

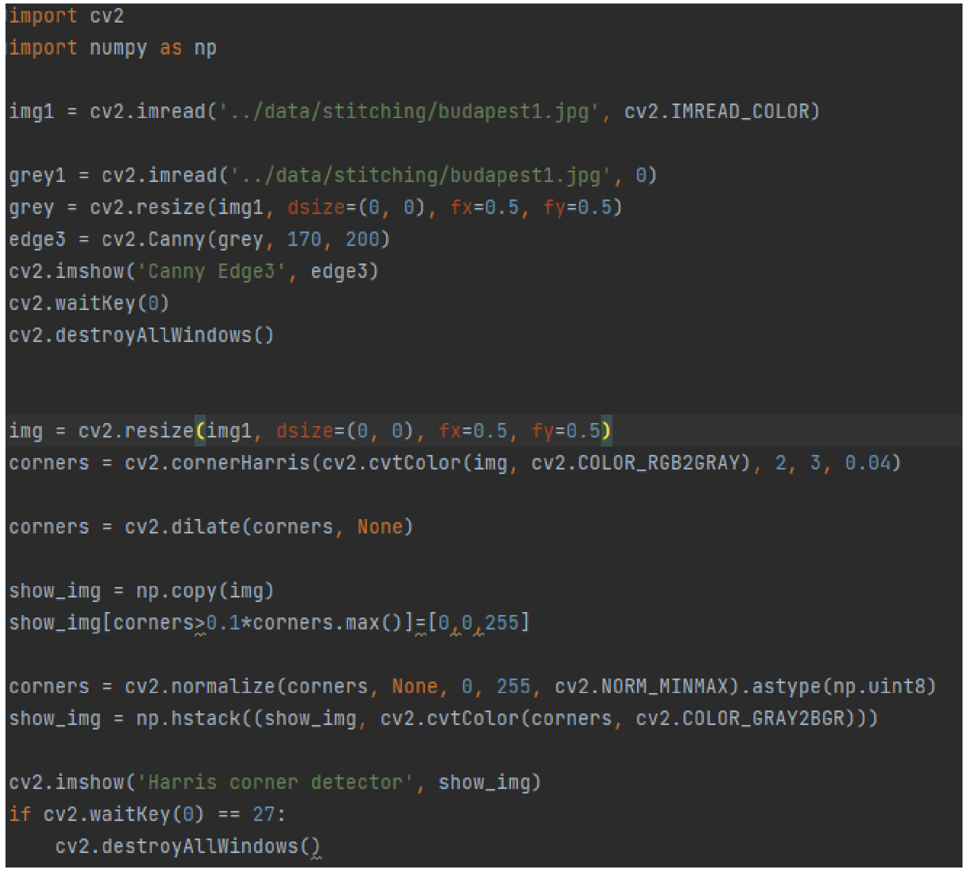
Canny Edge



Harris Corner

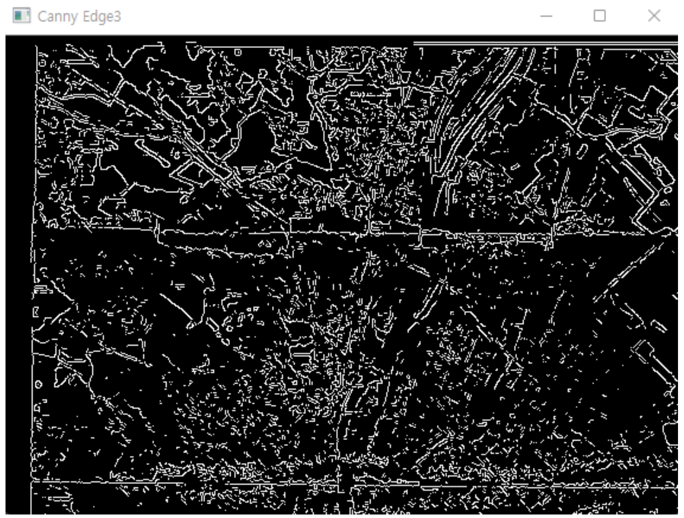


1) Feature Detection

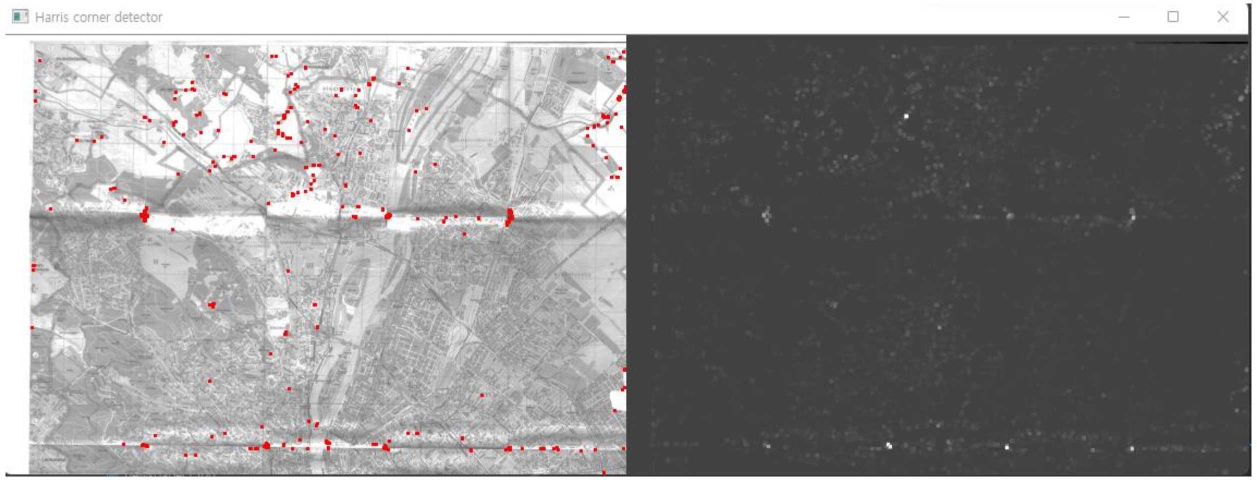


```
import cv2
import numpy as np

img1 = cv2.imread('../data/stitching/budapest1.jpg', cv2.IMREAD_COLOR)

grey1 = cv2.imread('../data/stitching/budapest1.jpg', 0)
grey = cv2.resize(img1, dsize=(0, 0), fx=0.5, fy=0.5)
edge3 = cv2.Canny(grey, 170, 200)
cv2.imshow('Canny Edge3', edge3)
cv2.waitKey(0)
cv2.destroyAllWindows()


img = cv2.resize(img1, dsize=(0, 0), fx=0.5, fy=0.5)
corners = cv2.cornerHarris(cv2.cvtColor(img, cv2.COLOR_RGB2GRAY), 2, 3, 0.04)

corners = cv2.dilate(corners, None)

show_img = np.copy(img)
show_img[corners>0.1*corners.max()]=[0,0,255]

corners = cv2.normalize(corners, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
show_img = np.hstack((show_img, cv2.cvtColor(corners, cv2.COLOR_GRAY2BGR)))

cv2.imshow('Harris corner detector', show_img)
if cv2.waitKey(0) == 27:
    cv2.destroyAllWindows()
```

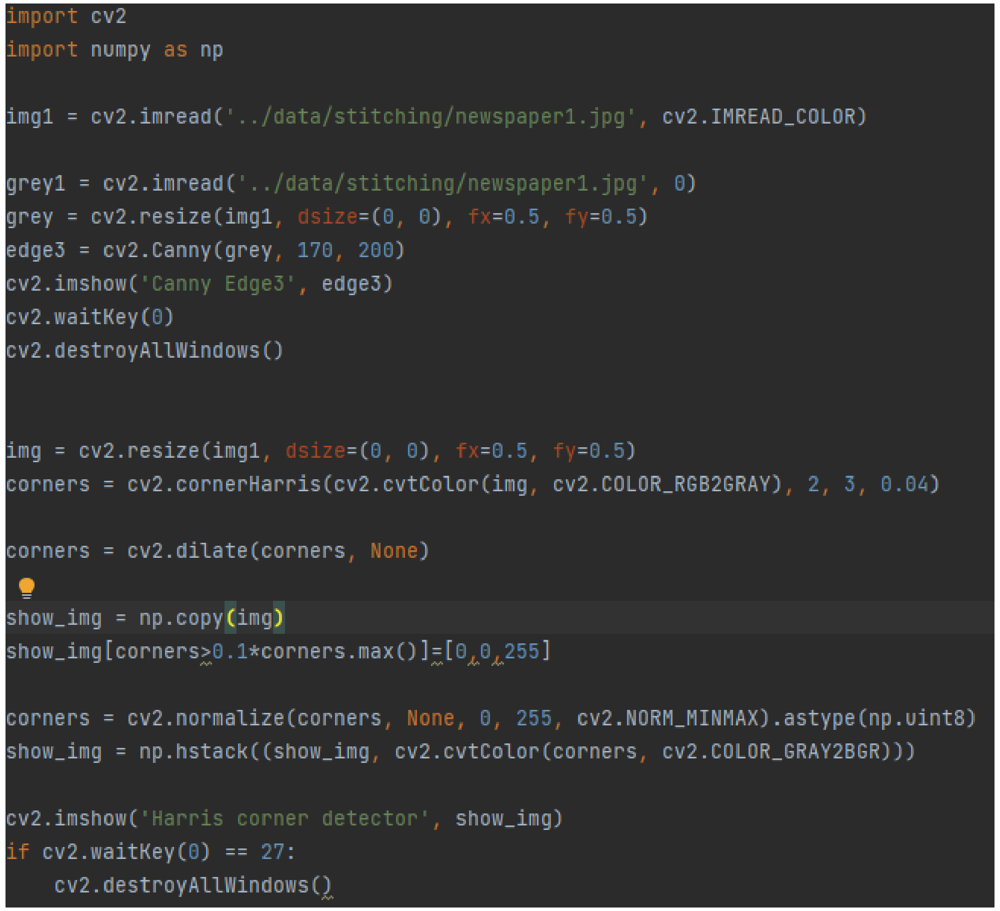
Canny Edge



Harris Corner

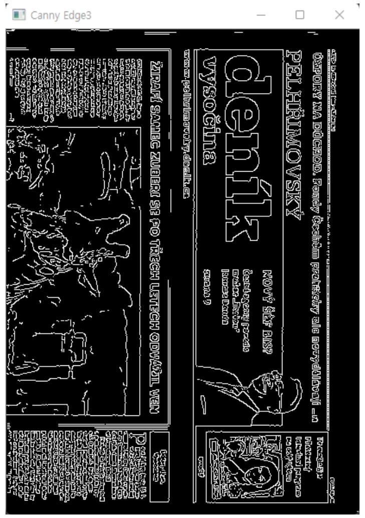


1) Feature Detection

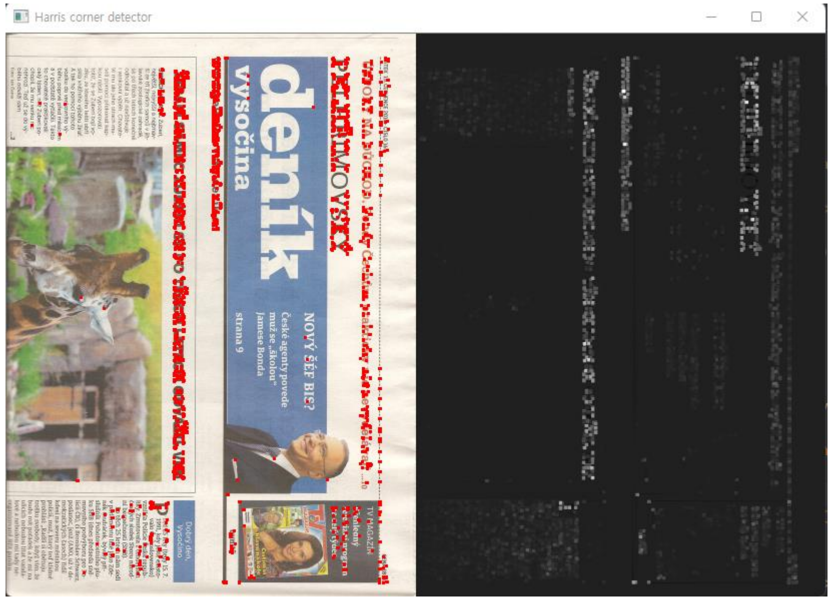


```python
import cv2
import numpy as np

img1 = cv2.imread('../data/stitching/newspaper1.jpg', cv2.IMREAD_COLOR)

grey1 = cv2.imread('../data/stitching/newspaper1.jpg', 0)
grey = cv2.resize(img1, dsize=(0, 0), fx=0.5, fy=0.5)
edge3 = cv2.Canny(grey, 170, 200)
cv2.imshow('Canny Edge3', edge3)
cv2.waitKey(0)
cv2.destroyAllWindows()


img = cv2.resize(img1, dsize=(0, 0), fx=0.5, fy=0.5)
corners = cv2.cornerHarris(cv2.cvtColor(img, cv2.COLOR_RGB2GRAY), 2, 3, 0.04)

corners = cv2.dilate(corners, None)

show_img = np.copy(img)
show_img[corners>0.1*corners.max()]=[0,0,255]

corners = cv2.normalize(corners, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
show_img = np.hstack((show_img, cv2.cvtColor(corners, cv2.COLOR_GRAY2BGR)))

cv2.imshow('Harris corner detector', show_img)
if cv2.waitKey(0) == 27:
    cv2.destroyAllWindows()
```

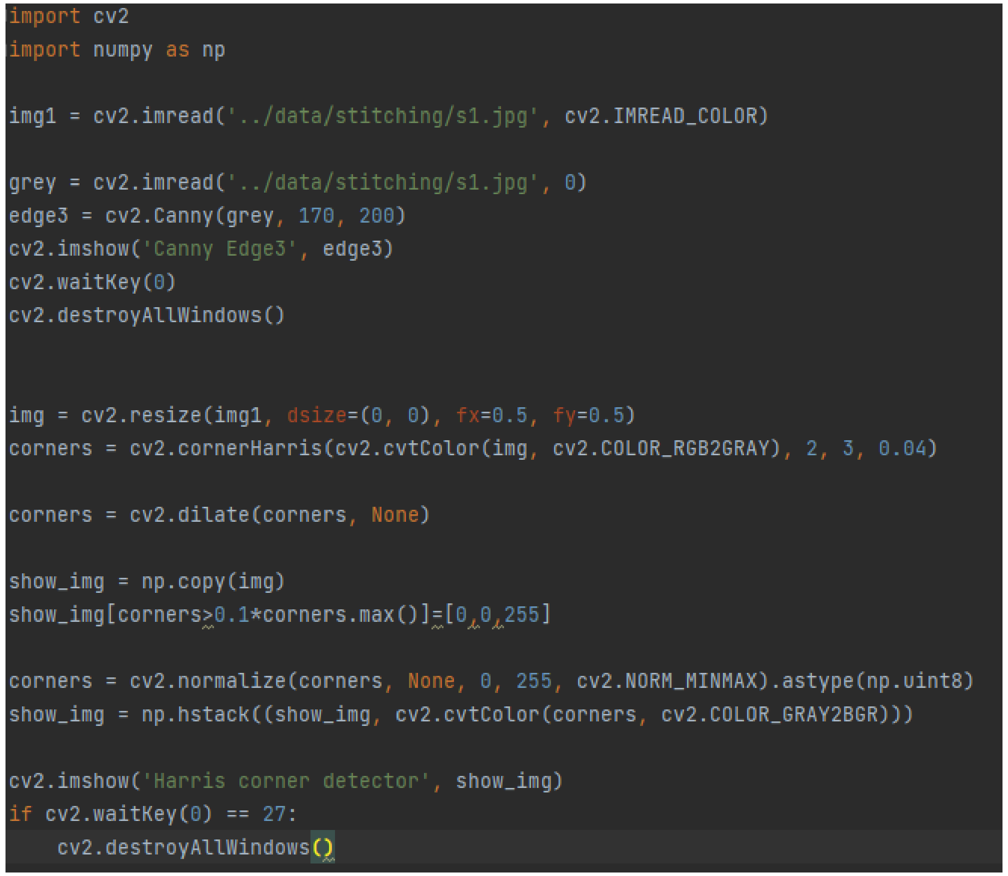
Canny Edge



Harris Corner

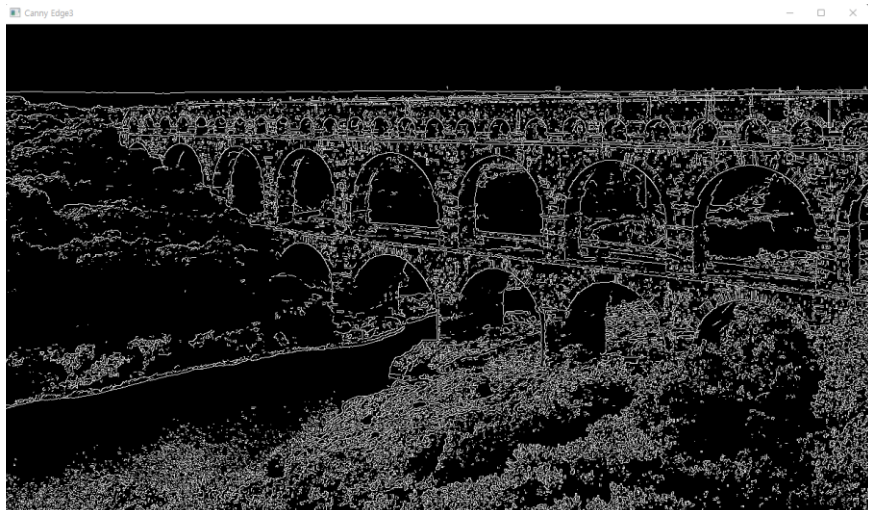


1) Feature Detection

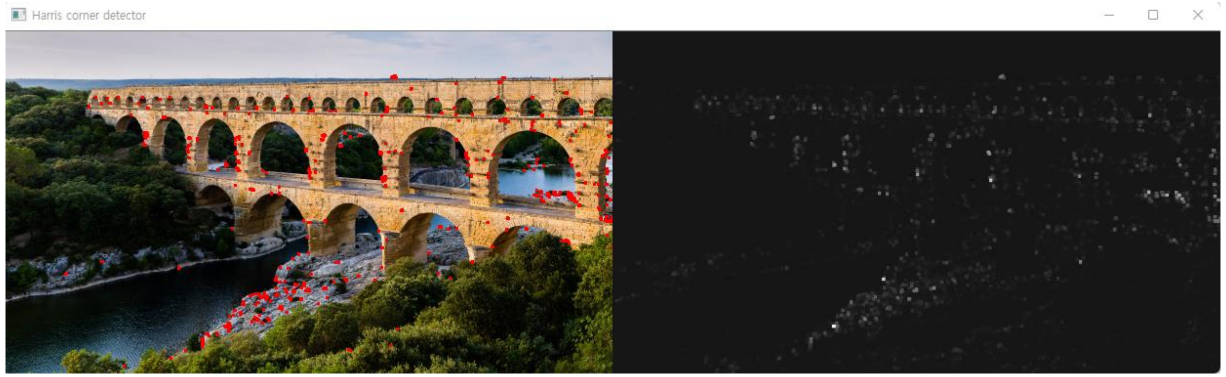


```
import cv2
import numpy as np

img1 = cv2.imread('../data/stitching/s1.jpg', cv2.IMREAD_COLOR)

grey = cv2.imread('../data/stitching/s1.jpg', 0)
edge3 = cv2.Canny(grey, 170, 200)
cv2.imshow('Canny Edge3', edge3)
cv2.waitKey(0)
cv2.destroyAllWindows()


img = cv2.resize(img1, dsize=(0, 0), fx=0.5, fy=0.5)
corners = cv2.cornerHarris(cv2.cvtColor(img, cv2.COLOR_RGB2GRAY), 2, 3, 0.04)

corners = cv2.dilate(corners, None)

show_img = np.copy(img)
show_img[corners>0.1*corners.max()]=[0,0,255]

corners = cv2.normalize(corners, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
show_img = np.hstack((show_img, cv2.cvtColor(corners, cv2.COLOR_GRAY2BGR)))

cv2.imshow('Harris corner detector', show_img)
if cv2.waitKey(0) == 27:
    cv2.destroyAllWindows()
```

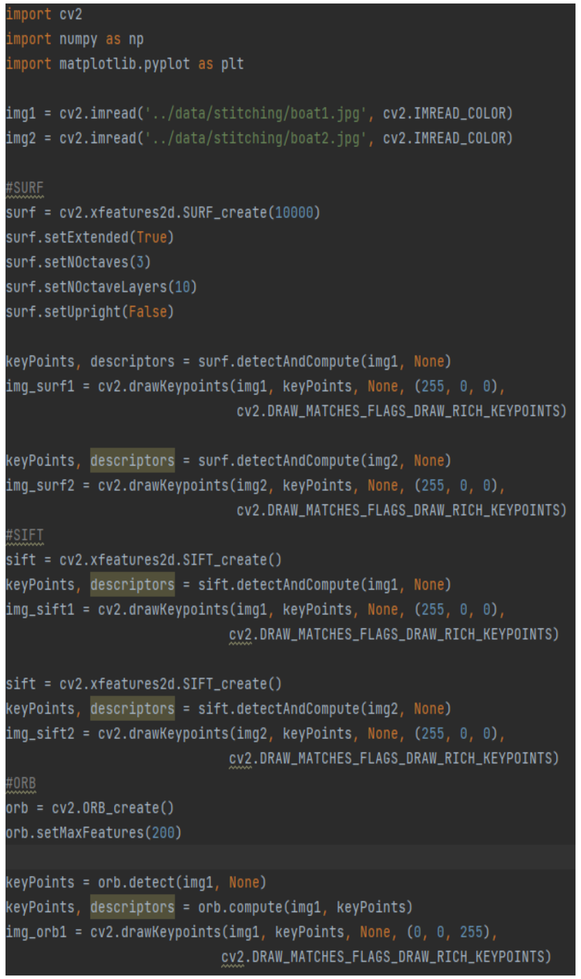
Canny Edge



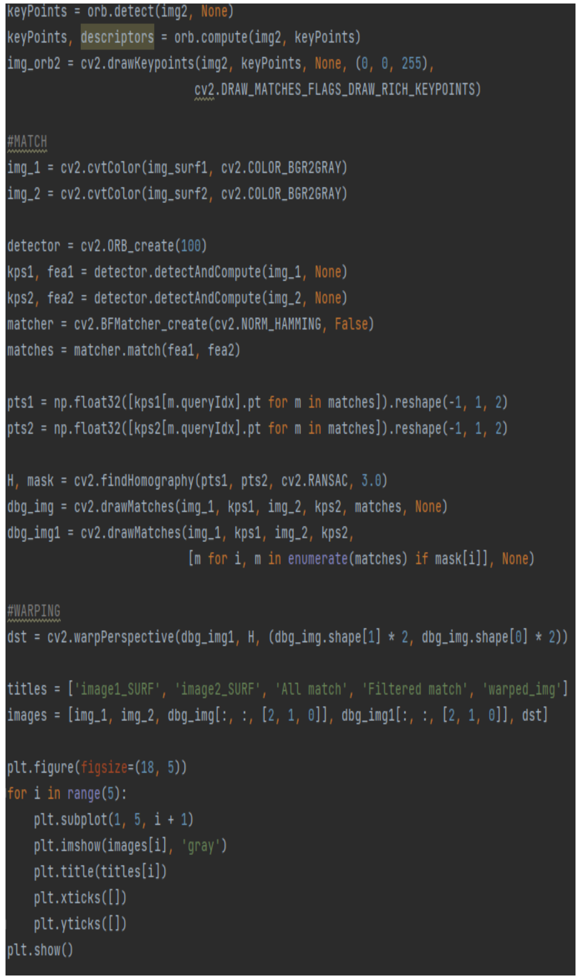
Harris Corner



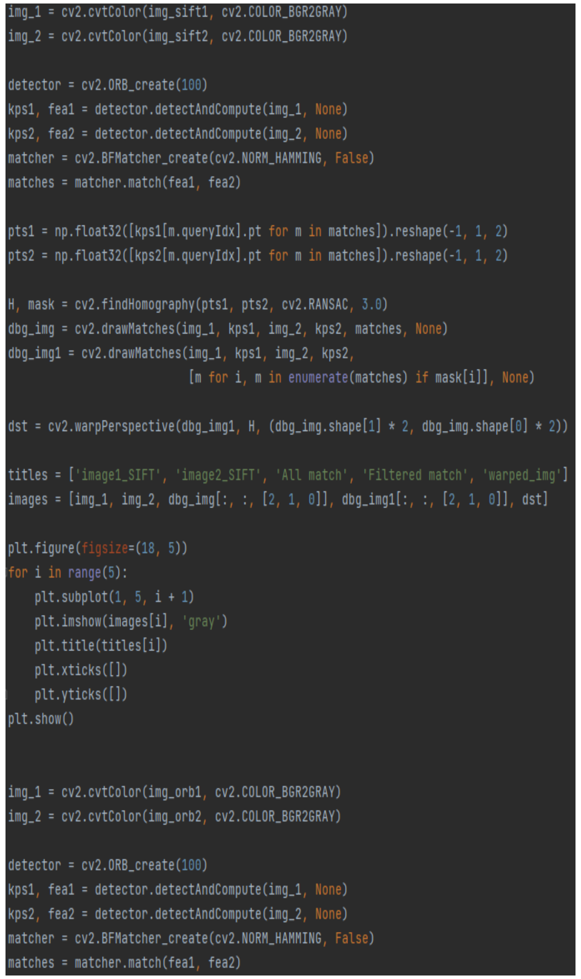
2) Matching



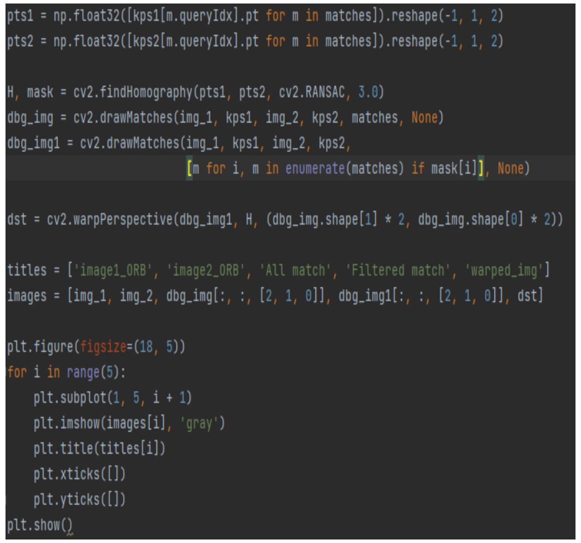
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img1 = cv2.imread('../data/stitching/boat1.jpg', cv2.IMREAD_COLOR)
img2 = cv2.imread('../data/stitching/boat2.jpg', cv2.IMREAD_COLOR)

#SURF
surf = cv2.xfeatures2d.SURF_create(10000)
surf.setExtended(True)
surf.setNOctaves(3)
surf.setNOctaveLayers(10)
surf.setUpright(False)

keyPoints, descriptors = surf.detectAndCompute(img1, None)
img_surf1 = cv2.drawKeypoints(img1, keyPoints, None, (255, 0, 0),
                              cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

keyPoints, descriptors = surf.detectAndCompute(img2, None)
img_surf2 = cv2.drawKeypoints(img2, keyPoints, None, (255, 0, 0),
                              cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
#SIFT
sift = cv2.xfeatures2d.SIFT_create()
keyPoints, descriptors = sift.detectAndCompute(img1, None)
img_sift1 = cv2.drawKeypoints(img1, keyPoints, None, (255, 0, 0),
                             cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

sift = cv2.xfeatures2d.SIFT_create()
keyPoints, descriptors = sift.detectAndCompute(img2, None)
img_sift2 = cv2.drawKeypoints(img2, keyPoints, None, (255, 0, 0),
                             cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
#ORB
orb = cv2.ORB_create()
orb.setMaxFeatures(200)

keyPoints = orb.detect(img1, None)
keyPoints, descriptors = orb.compute(img1, keyPoints)
img_orb1 = cv2.drawKeypoints(img1, keyPoints, None, (0, 0, 255),
                            cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```



```python
keyPoints = orb.detect(img2, None)
keyPoints, descriptors = orb.compute(img2, keyPoints)
img_orb2 = cv2.drawKeypoints(img2, keyPoints, None, (0, 0, 255),
                            cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

#MATCH
img_1 = cv2.cvtColor(img_surf1, cv2.COLOR_BGR2GRAY)
img_2 = cv2.cvtColor(img_surf2, cv2.COLOR_BGR2GRAY)

detector = cv2.ORB_create(100)
kps1, fea1 = detector.detectAndCompute(img_1, None)
kps2, fea2 = detector.detectAndCompute(img_2, None)
matcher = cv2.BFMatcher_create(cv2.NORM_HAMMING, False)
matches = matcher.match(fea1, fea2)

pts1 = np.float32([kps1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
pts2 = np.float32([kps2[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)

H, mask = cv2.findHomography(pts1, pts2, cv2.RANSAC, 3.0)
dbg_img = cv2.drawMatches(img_1, kps1, img_2, kps2, matches, None)
dbg_img1 = cv2.drawMatches(img_1, kps1, img_2, kps2,
                           [m for i, m in enumerate(matches) if mask[i]], None)

#WARPING
dst = cv2.warpPerspective(dbg_img1, H, (dbg_img.shape[1] * 2, dbg_img.shape[0] * 2))

titles = ['image1_SURF', 'image2_SURF', 'All match', 'Filtered match', 'warped_img']
images = [img_1, img_2, dbg_img[:, :, [2, 1, 0]], dbg_img1[:, :, [2, 1, 0]], dst]

plt.figure(figsize=(18, 5))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])
plt.show()
```



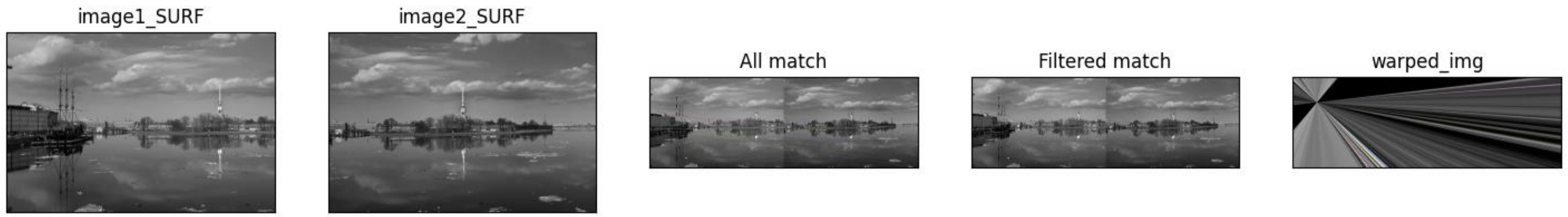
```python
img_1 = cv2.cvtColor(img_sift1, cv2.COLOR_BGR2GRAY)
img_2 = cv2.cvtColor(img_sift2, cv2.COLOR_BGR2GRAY)

detector = cv2.ORB_create(100)
kps1, fea1 = detector.detectAndCompute(img_1, None)
kps2, fea2 = detector.detectAndCompute(img_2, None)
matcher = cv2.BFMatcher_create(cv2.NORM_HAMMING, False)
matches = matcher.match(fea1, fea2)

pts1 = np.float32([kps1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
pts2 = np.float32([kps2[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)

H, mask = cv2.findHomography(pts1, pts2, cv2.RANSAC, 3.0)
dbg_img = cv2.drawMatches(img_1, kps1, img_2, kps2, matches, None)
dbg_img1 = cv2.drawMatches(img_1, kps1, img_2, kps2,
                           [m for i, m in enumerate(matches) if mask[i]], None)

dst = cv2.warpPerspective(dbg_img1, H, (dbg_img.shape[1] * 2, dbg_img.shape[0] * 2))

titles = ['image1_SIFT', 'image2_SIFT', 'All match', 'Filtered match', 'warped_img']
images = [img_1, img_2, dbg_img[:, :, [2, 1, 0]], dbg_img1[:, :, [2, 1, 0]], dst]

plt.figure(figsize=(18, 5))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])
plt.show()


img_1 = cv2.cvtColor(img_orb1, cv2.COLOR_BGR2GRAY)
img_2 = cv2.cvtColor(img_orb2, cv2.COLOR_BGR2GRAY)

detector = cv2.ORB_create(100)
kps1, fea1 = detector.detectAndCompute(img_1, None)
kps2, fea2 = detector.detectAndCompute(img_2, None)
matcher = cv2.BFMatcher_create(cv2.NORM_HAMMING, False)
matches = matcher.match(fea1, fea2)
```



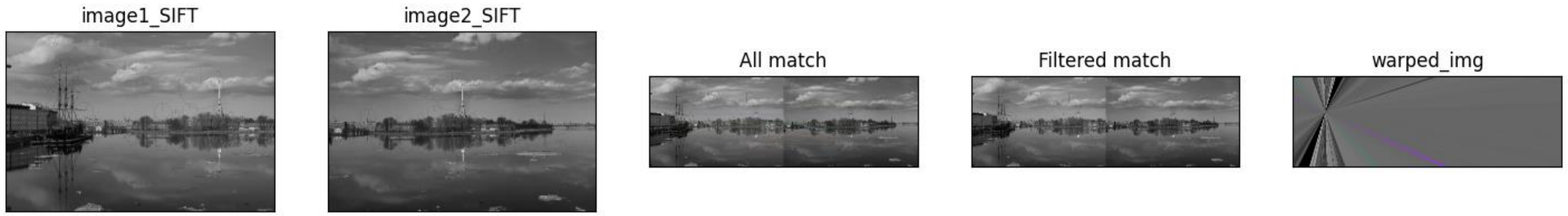
```python
pts1 = np.float32([kps1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
pts2 = np.float32([kps2[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)

H, mask = cv2.findHomography(pts1, pts2, cv2.RANSAC, 3.0)
dbg_img = cv2.drawMatches(img_1, kps1, img_2, kps2, matches, None)
dbg_img1 = cv2.drawMatches(img_1, kps1, img_2, kps2,
                           [m for i, m in enumerate(matches) if mask[i]], None)

dst = cv2.warpPerspective(dbg_img1, H, (dbg_img.shape[1] * 2, dbg_img.shape[0] * 2))

titles = ['image1_ORB', 'image2_ORB', 'All match', 'Filtered match', 'warped_img']
images = [img_1, img_2, dbg_img[:, :, [2, 1, 0]], dbg_img1[:, :, [2, 1, 0]], dst]

plt.figure(figsize=(18, 5))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])
plt.show()
```

2) Matching

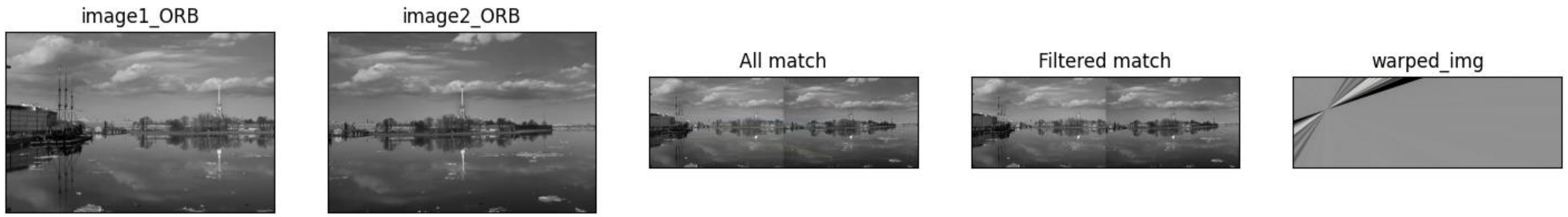
boat : SURF, MATCH, WARPED



boat : SIFT, MATCH, WARPED

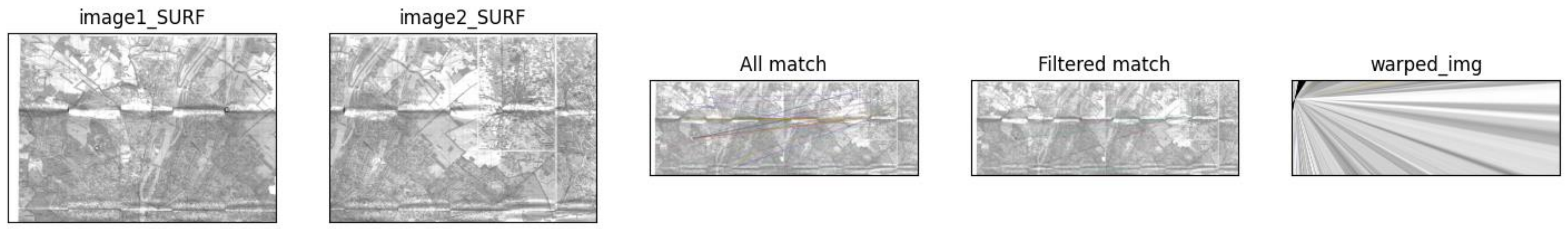


boat : ORB, MATCH, WARPED

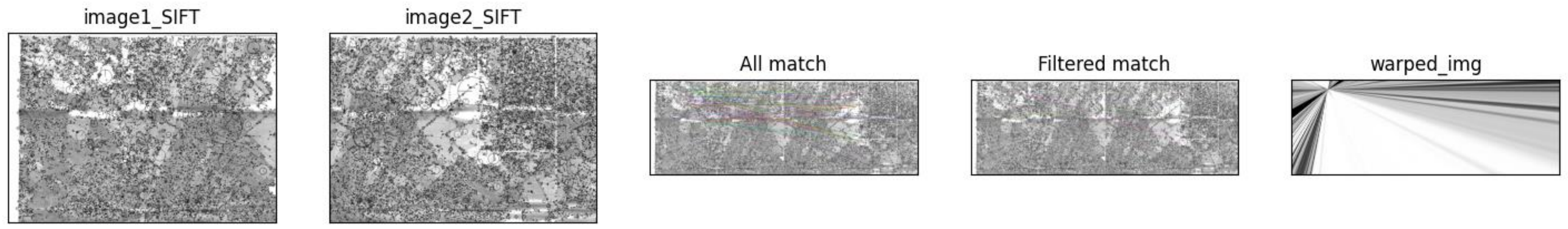


2) Matching

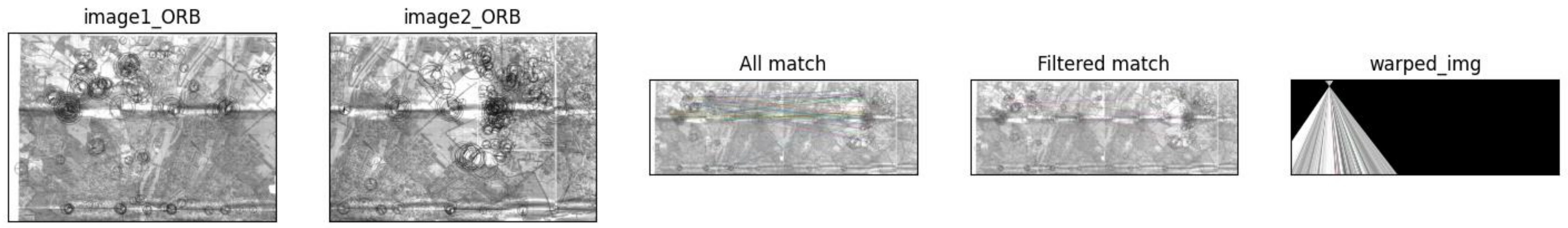
budapest : SURF, MATCH, WARPED



budapest : SIFT, MATCH, WARPED



budapest : ORB, MATCH, WARPED



2) Matching

newspaper : SURF, MATCH, WARPED

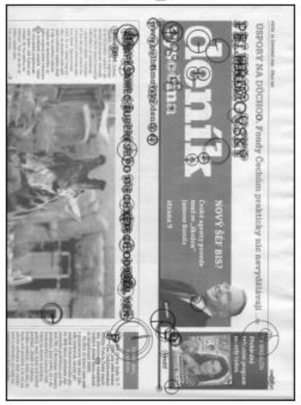


image1_SURF

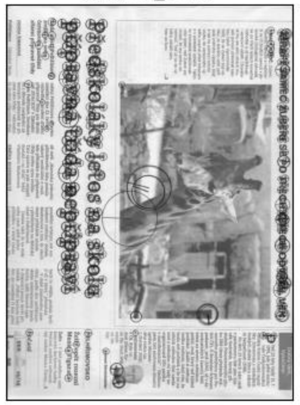
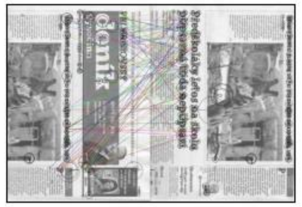
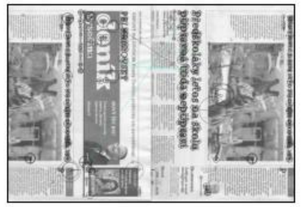
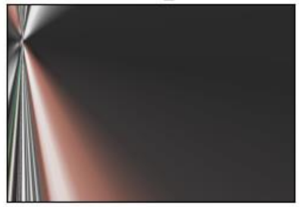


image2_SURF


All match


Filtered match


warped_img

newspaper : SIFT, MATCH, WARPED

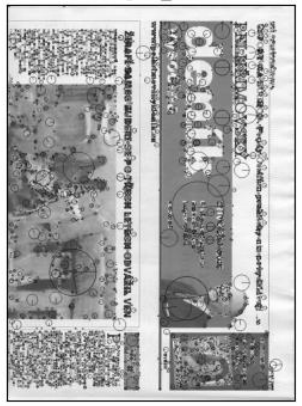


image1_SIFT

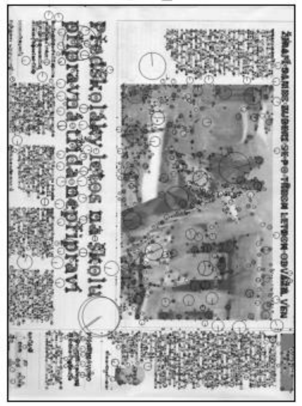
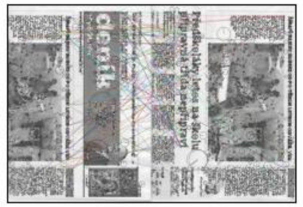
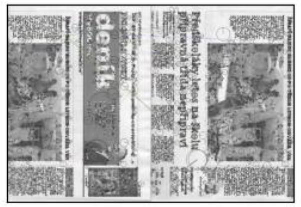
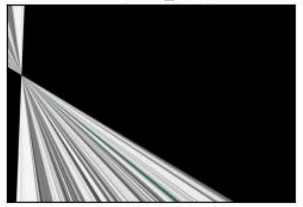


image2_SIFT


All match


Filtered match


warped_img

newspaper : ORB, MATCH, WARPED

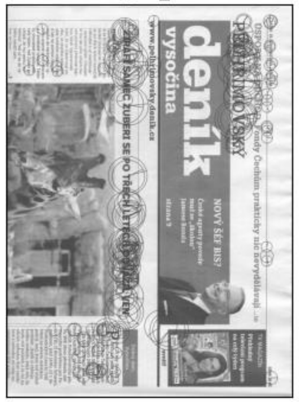


image1_ORB

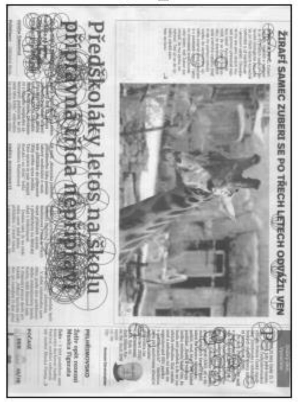
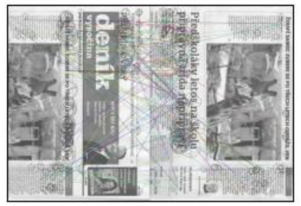
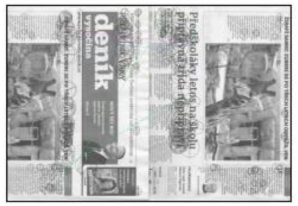
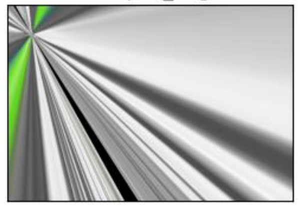


image2_ORB

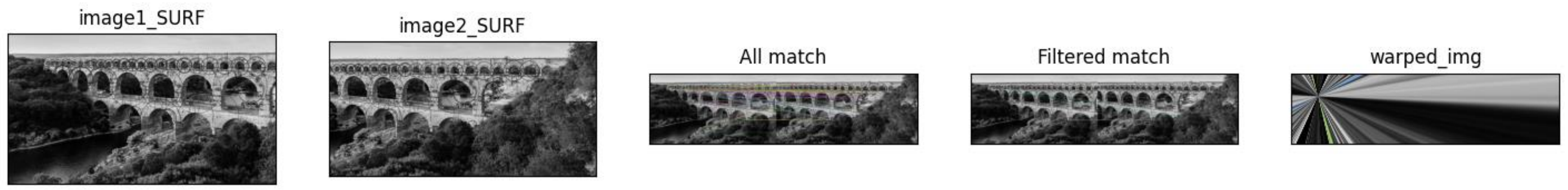

All match


Filtered match

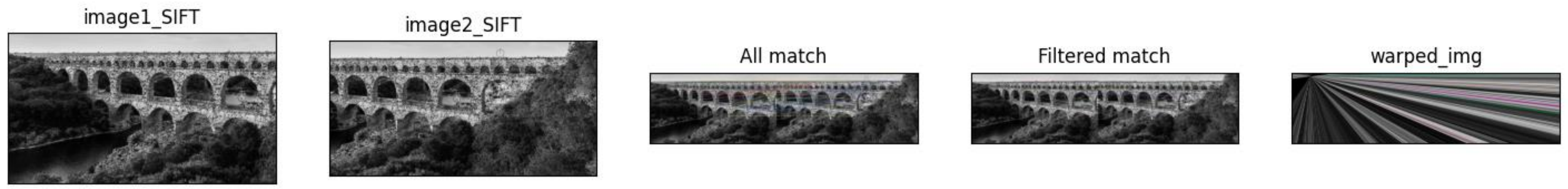

warped_img

2) Matching

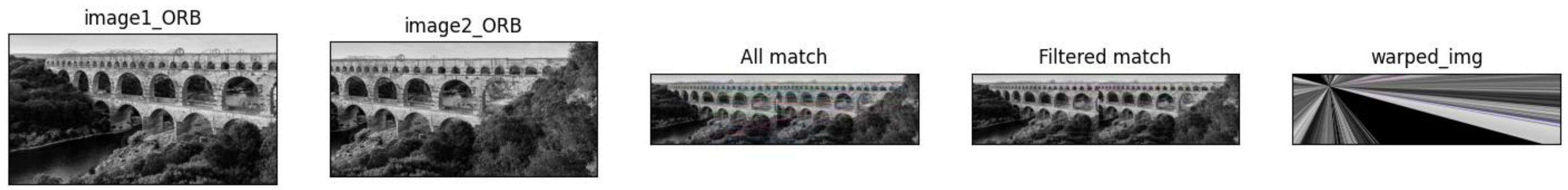
s : SURF, MATCH, WARPED



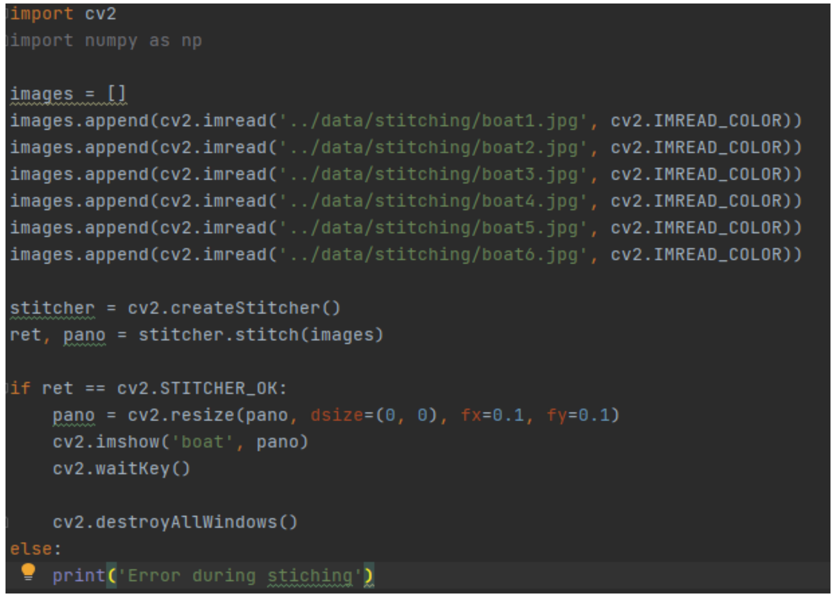
s : SIFT, MATCH, WARPED



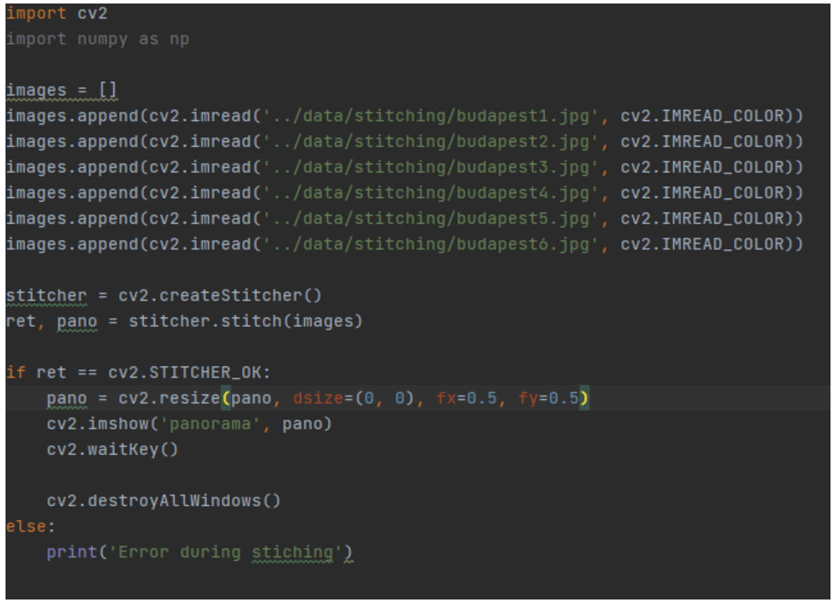
s : ORB, MATCH, WARPED



3) Panorama

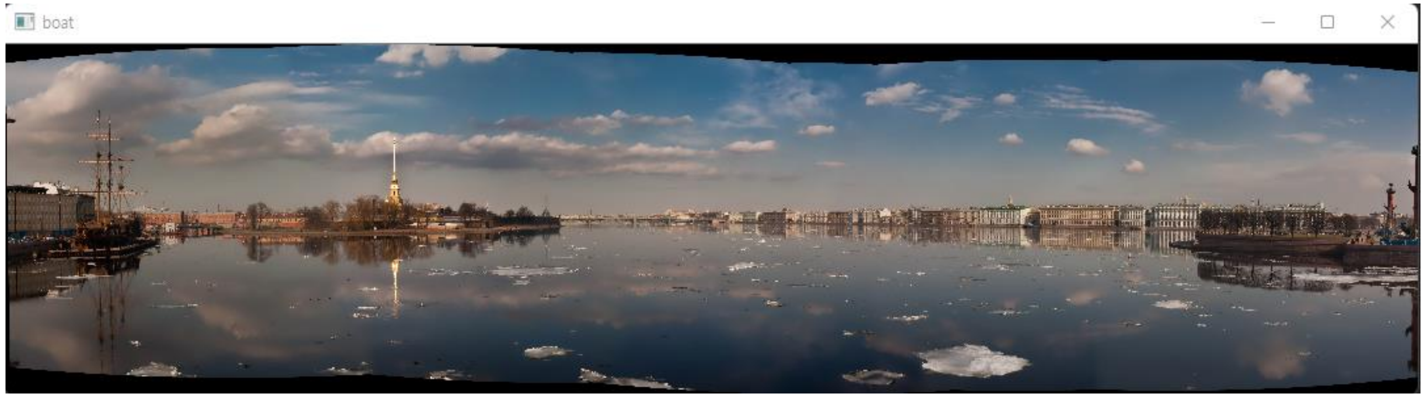


```
import cv2
import numpy as np

images = []
images.append(cv2.imread('../data/stitching/boat1.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/boat2.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/boat3.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/boat4.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/boat5.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/boat6.jpg', cv2.IMREAD_COLOR))

stitcher = cv2.createStitcher()
ret, pano = stitcher.stitch(images)

if ret == cv2.STITCHER_OK:
    pano = cv2.resize(pano, dsize=(0, 0), fx=0.1, fy=0.1)
    cv2.imshow('boat', pano)
    cv2.waitKey()

    cv2.destroyAllWindows()
else:
    print('Error during stiching')
```

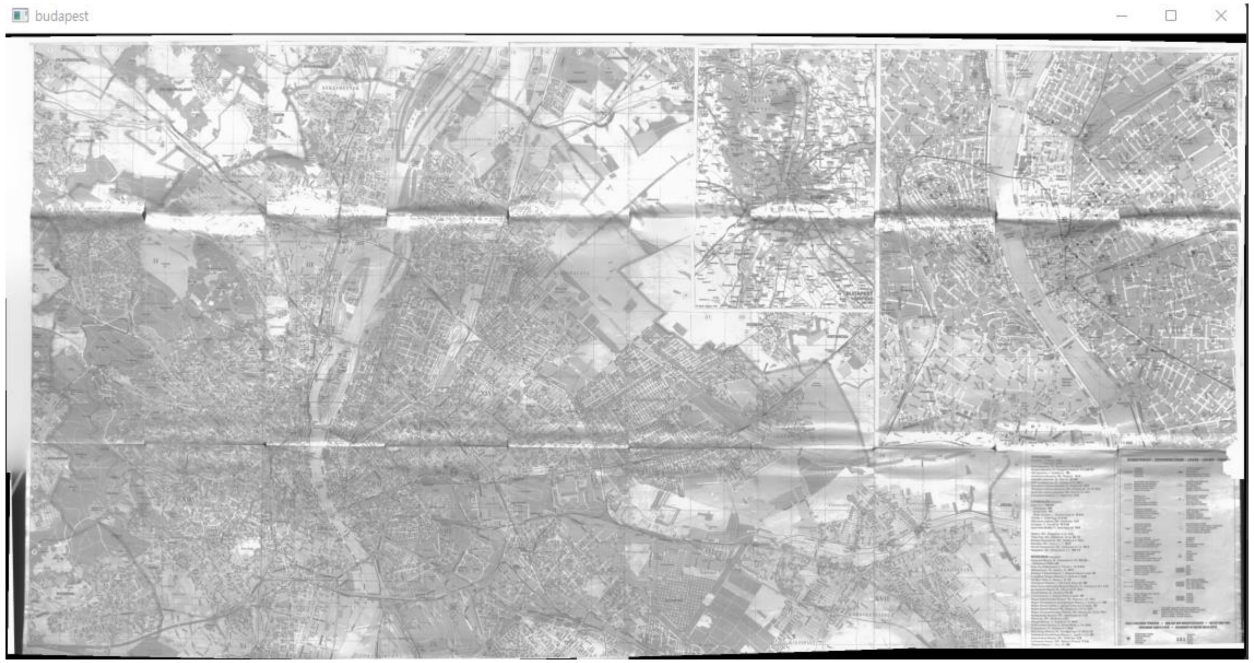


```
import cv2
import numpy as np

images = []
images.append(cv2.imread('../data/stitching/budapest1.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/budapest2.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/budapest3.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/budapest4.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/budapest5.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/budapest6.jpg', cv2.IMREAD_COLOR))

stitcher = cv2.createStitcher()
ret, pano = stitcher.stitch(images)

if ret == cv2.STITCHER_OK:
    pano = cv2.resize(pano, dsize=(0, 0), fx=0.5, fy=0.5)
    cv2.imshow('panorama', pano)
    cv2.waitKey()

    cv2.destroyAllWindows()
else:
    print('Error during stiching')
```

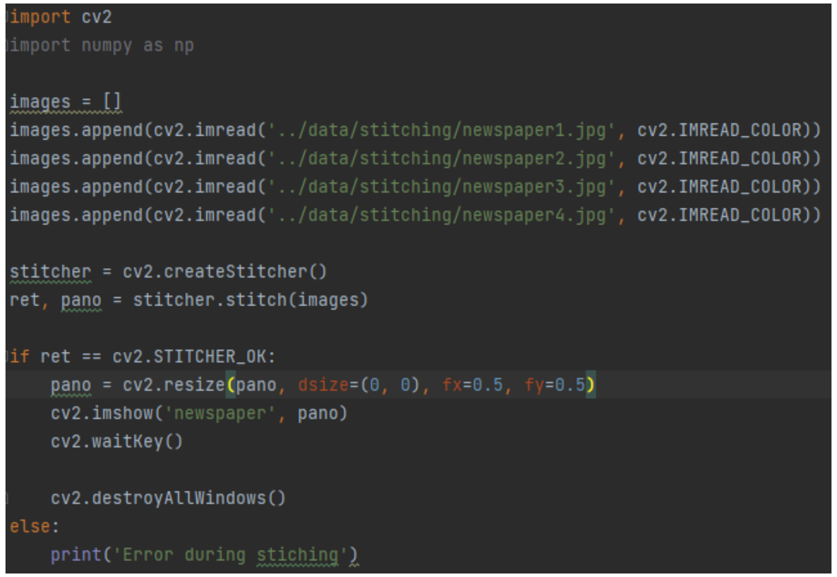
boat : fx=0.1, fy=0.1



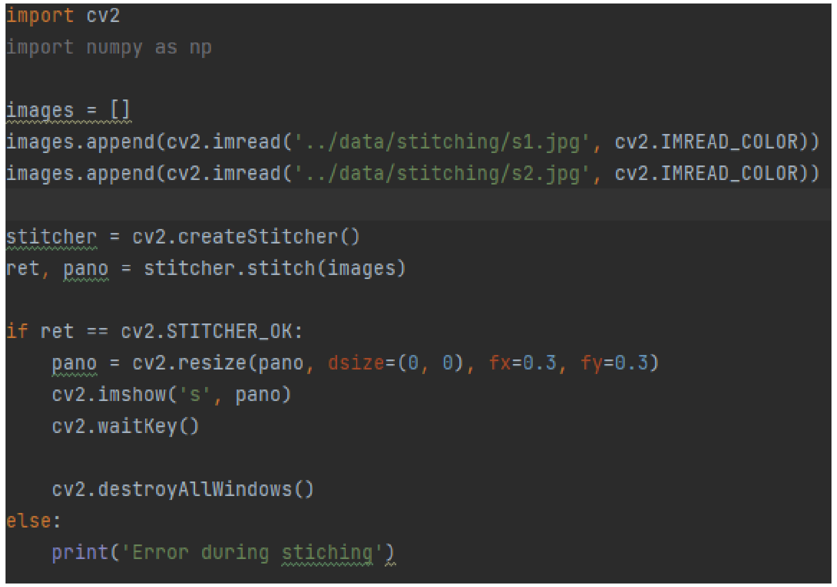
budapest : fx=0.5, fy=0.5



3) Panorama

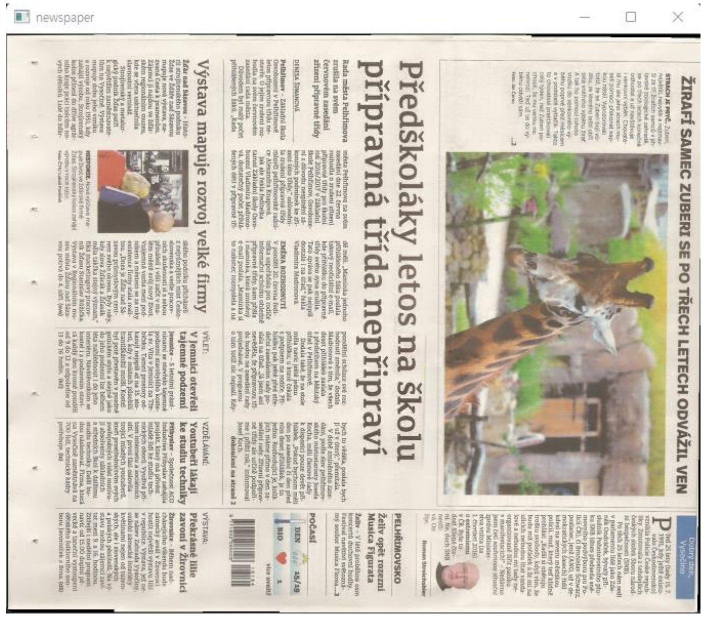


```
import cv2
import numpy as np

images = []
images.append(cv2.imread('../data/stitching/newspaper1.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/newspaper2.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/newspaper3.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/newspaper4.jpg', cv2.IMREAD_COLOR))

stitcher = cv2.createStitcher()
ret, pano = stitcher.stitch(images)

if ret == cv2.STITCHER_OK:
    pano = cv2.resize(pano, dsize=(0, 0), fx=0.5, fy=0.5)
    cv2.imshow('newspaper', pano)
    cv2.waitKey()

    cv2.destroyAllWindows()
else:
    print('Error during stiching')
```

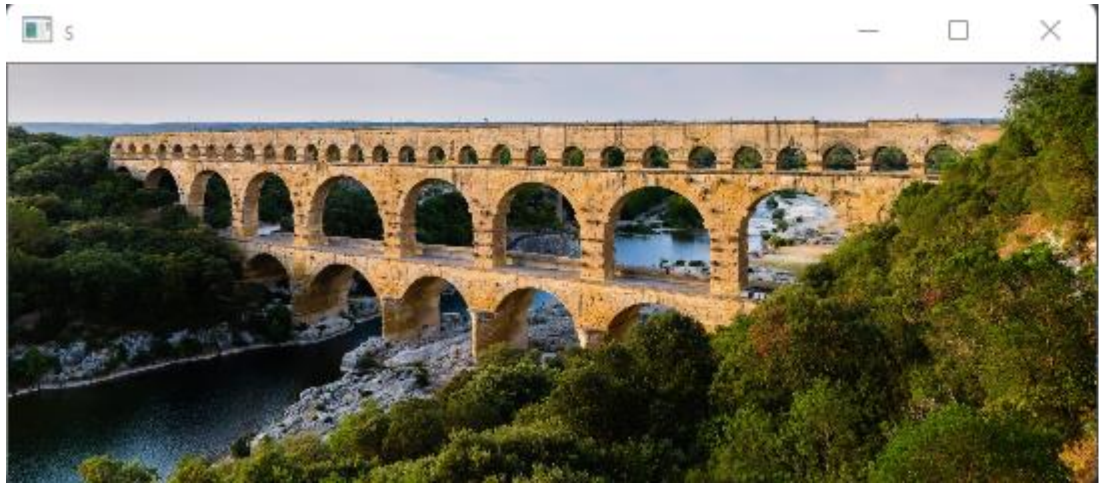


```
import cv2
import numpy as np

images = []
images.append(cv2.imread('../data/stitching/s1.jpg', cv2.IMREAD_COLOR))
images.append(cv2.imread('../data/stitching/s2.jpg', cv2.IMREAD_COLOR))

stitcher = cv2.createStitcher()
ret, pano = stitcher.stitch(images)

if ret == cv2.STITCHER_OK:
    pano = cv2.resize(pano, dsize=(0, 0), fx=0.3, fy=0.3)
    cv2.imshow('s', pano)
    cv2.waitKey()

    cv2.destroyAllWindows()
else:
    print('Error during stiching')
```

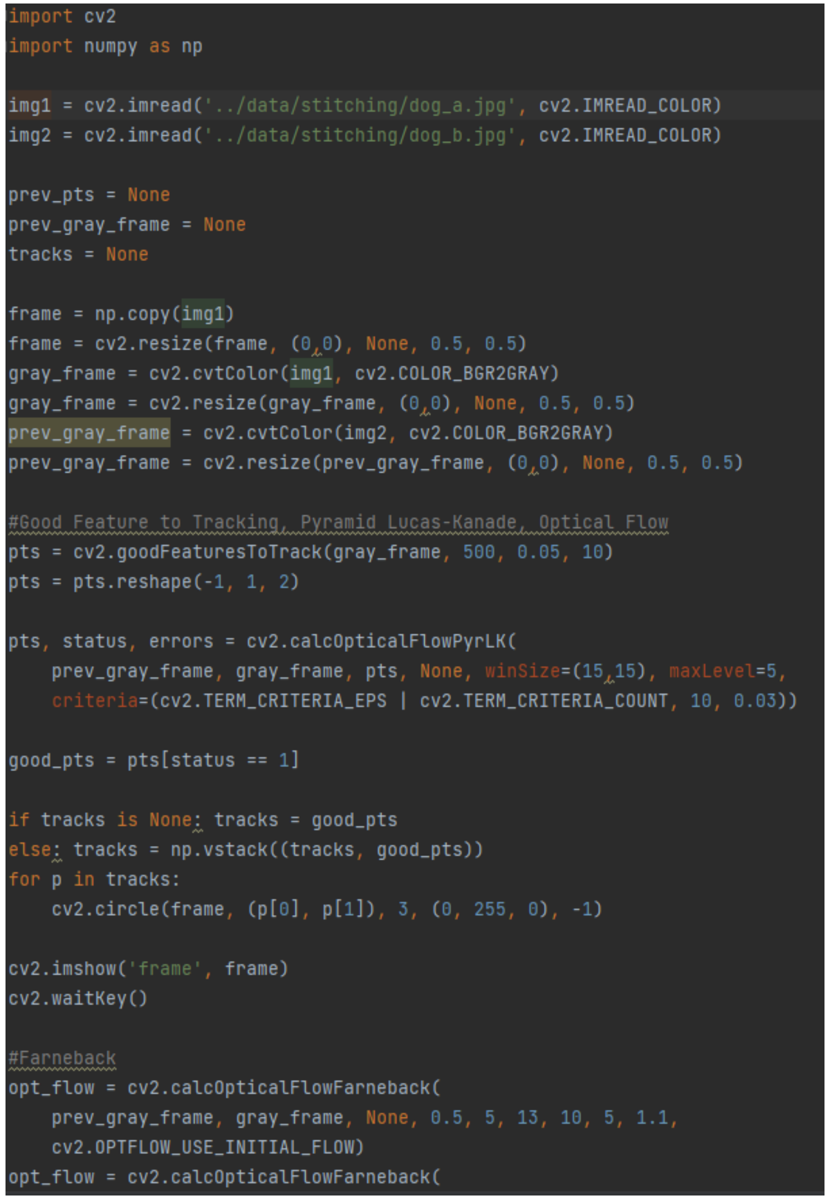
newspaper : fx=0.5, fy=0.5


newspaper
ŽIRAFÍ SAMEC ZUBERI SE PO TŘECH LETECH ODVÁŽIL VEN
Předškoláky letos na školu připravná třída nepřipraví
Výstava mapuje rozvoj velké firmy

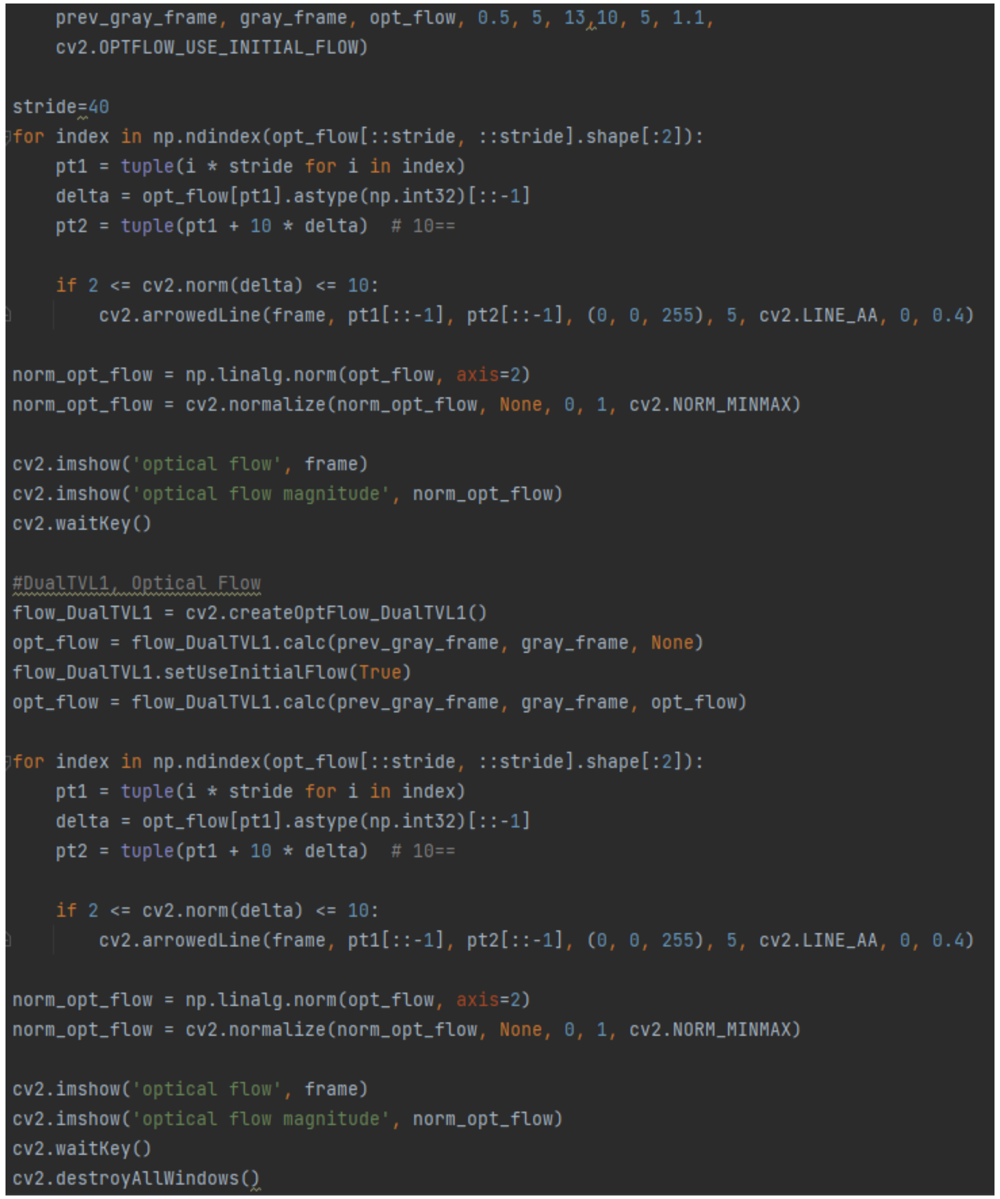
s : fx=0.3, fy=0.3



4) Optical Flow



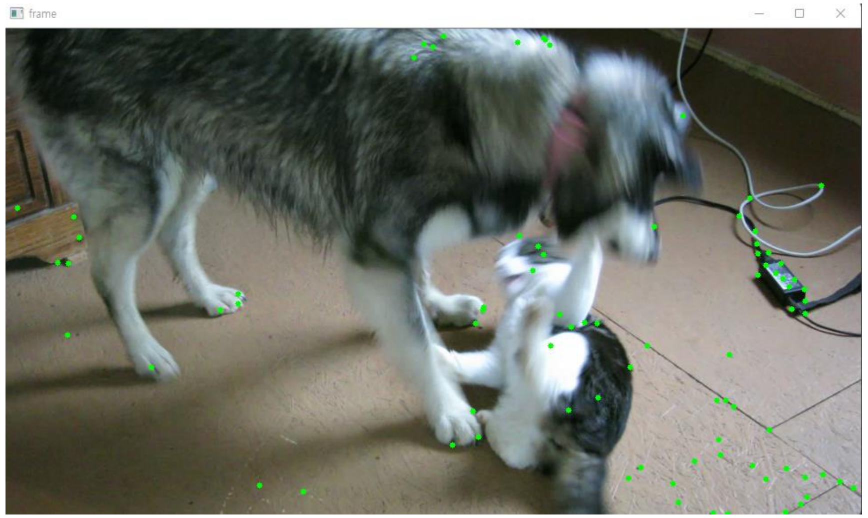
```
import cv2
import numpy as np

img1 = cv2.imread('../data/stitching/dog_a.jpg', cv2.IMREAD_COLOR)
img2 = cv2.imread('../data/stitching/dog_b.jpg', cv2.IMREAD_COLOR)

prev_pts = None
prev_gray_frame = None
tracks = None

frame = np.copy(img1)
frame = cv2.resize(frame, (0,0), None, 0.5, 0.5)
gray_frame = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray_frame = cv2.resize(gray_frame, (0,0), None, 0.5, 0.5)
prev_gray_frame = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
prev_gray_frame = cv2.resize(prev_gray_frame, (0,0), None, 0.5, 0.5)

#Good Feature to Tracking, Pyramid Lucas-Kanade, Optical Flow
pts = cv2.goodFeaturesToTrack(gray_frame, 500, 0.05, 10)
pts = pts.reshape(-1, 1, 2)

pts, status, errors = cv2.calcOpticalFlowPyrLK(
    prev_gray_frame, gray_frame, pts, None, winSize=(15,15), maxLevel=5,
    criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

good_pts = pts[status == 1]

if tracks is None: tracks = good_pts
else: tracks = np.vstack((tracks, good_pts))
for p in tracks:
    cv2.circle(frame, (p[0], p[1]), 3, (0, 255, 0), -1)

cv2.imshow('frame', frame)
cv2.waitKey()

#Farneback
opt_flow = cv2.calcOpticalFlowFarneback(
    prev_gray_frame, gray_frame, None, 0.5, 5, 13, 10, 5, 1.1,
    cv2.OPTFLOW_USE_INITIAL_FLOW)
opt_flow = cv2.calcOpticalFlowFarneback(
```



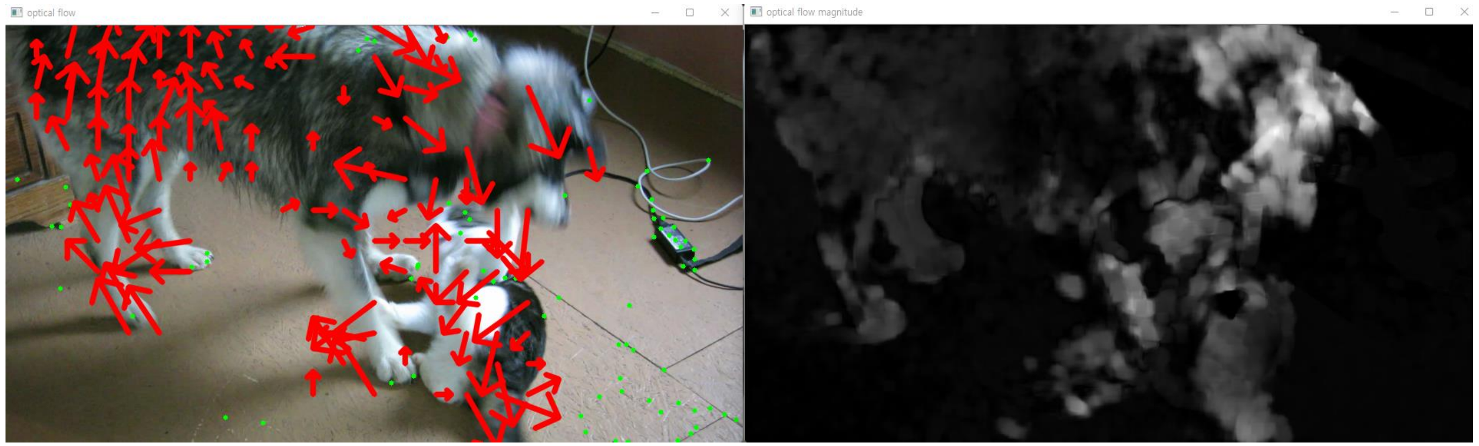
```
    prev_gray_frame, gray_frame, opt_flow, 0.5, 5, 13,10, 5, 1.1,
    cv2.OPTFLOW_USE_INITIAL_FLOW)

stride=40
for index in np.ndindex(opt_flow[::stride, ::stride].shape[:2]):
    pt1 = tuple(i * stride for i in index)
    delta = opt_flow[pt1].astype(np.int32)[::-1]
    pt2 = tuple(pt1 + 10 * delta)  # 10==

    if 2 <= cv2.norm(delta) <= 10:
        cv2.arrowedLine(frame, pt1[::-1], pt2[::-1], (0, 0, 255), 5, cv2.LINE_AA, 0, 0.4)

norm_opt_flow = np.linalg.norm(opt_flow, axis=2)
norm_opt_flow = cv2.normalize(norm_opt_flow, None, 0, 1, cv2.NORM_MINMAX)

cv2.imshow('optical flow', frame)
cv2.imshow('optical flow magnitude', norm_opt_flow)
cv2.waitKey()

#DualTVL1, Optical Flow
flow_DualTVL1 = cv2.createOptFlow_DualTVL1()
opt_flow = flow_DualTVL1.calc(prev_gray_frame, gray_frame, None)
flow_DualTVL1.setUseInitialFlow(True)
opt_flow = flow_DualTVL1.calc(prev_gray_frame, gray_frame, opt_flow)

for index in np.ndindex(opt_flow[::stride, ::stride].shape[:2]):
    pt1 = tuple(i * stride for i in index)
    delta = opt_flow[pt1].astype(np.int32)[::-1]
    pt2 = tuple(pt1 + 10 * delta)  # 10==

    if 2 <= cv2.norm(delta) <= 10:
        cv2.arrowedLine(frame, pt1[::-1], pt2[::-1], (0, 0, 255), 5, cv2.LINE_AA, 0, 0.4)

norm_opt_flow = np.linalg.norm(opt_flow, axis=2)
norm_opt_flow = cv2.normalize(norm_opt_flow, None, 0, 1, cv2.NORM_MINMAX)

cv2.imshow('optical flow', frame)
cv2.imshow('optical flow magnitude', norm_opt_flow)
cv2.waitKey()
cv2.destroyAllWindows()
```

4) Optical Flow

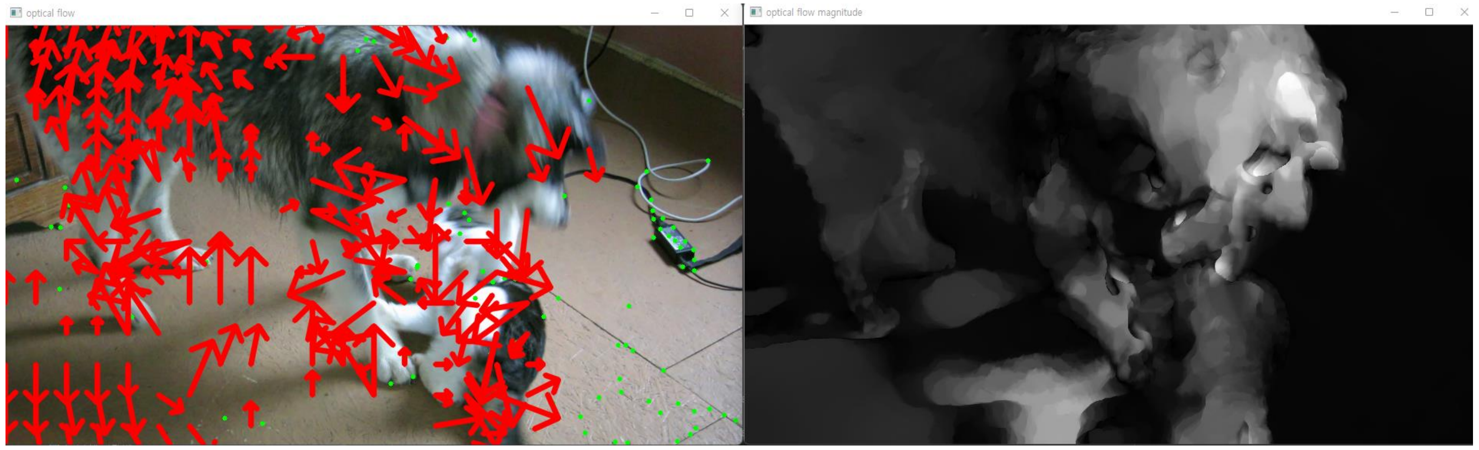
Good Feature to Tracking,
Pyramid Lucas-Kanade알고리즘 Optical Flow



Farneback



DualTVL1 Optical Flow


optical flow
optical flow magnitude