19M041DOS2 2023/2024 – drugi domaći zadatak

sudent: Veljko Milojević
broj indeksa: 2023/3066

Ucitavanje biblioteka

In [ ]:
```
import cv2 as cv
import numpy as np
import os
import matplotlib.pyplot as plt
```

Definisanje konstanti

In [ ]:
```
FIGSIZE = (16, 9)
DPI = 80
S = 20
```

ZADATAK 1 – Realizacija Lucas-Kanade metode optičkog toka

U okviru ovog zadatka potrebno je implementirati funkciju kojom se realizuje Lucas-Kanade estimacija optičkog toka. Funkciju nazvati **lucasKanade**. Argumenti funkcije s:

Ulazi:
img1 - ulazna siva slika trenutnog frejma.
img2 - ulazna siva slika narednog frejm.
keypoints – niz dimenzija (N,2) sa pozicijama ključnih tačaka.
winSize - veličina prozora oko ključne tačke koji se koristi za estimaciju.
errThr – prag za odbacivanje loše detektovanih tačaka.

Izlaz:
flowVectors – niz dimenzija (N,2) sa vrednostima komponenti pomeraja u svakoj ključnoj tački.
validPoints – niz dimenzija N boolean vrednosti koji određuje da li je odgovarajuća ključna tačk pronađena u narednom frejmu. Ključna tačka se smatra izgubljenom ukoliko je njena nova pozicija izvan okvira frejma ili ukoliko je srednja kvadratna razlika normalizovanog prozora 3x3 oko stare i nove pozicije tačke veća od praga *errThr*.

Prilikom testiranja funkcije, iskoristiti neku od gotovih funkcija *corner_peaks*, *corner_harris* iz skimage biblioteke ili *cornerHarris*, *goodFeaturesToTrack* iz OpenCV biblioteke.

Testirati implementiranu funkciju na sekvenci slika iz direktorijuma **Can**. Prilikom testiranja, detektovati ključne tačke samo na prvoj slici sekvence. U izveštaju prikazati ovu sliku sa ključnim tačkama. Odrediti vektore optičkog toka, iscrtati strelice koje predstavljaju ove vektore na svakoj ključnoj tački i prikazati ovaj rezultat u izveštaju. Nakon toga, izvršiti praćenje ključnih tačaka kroz sve slike sekvence. Praćenje se obavlja tako što se na osnovu pozicije ključnih tačaka trenutnog frejma određuje pozicija ključnih tačaka narednog frejma. **Prate se samo validne ključne tačke**. Dakle, broj ključnih tačaka će se smanjivati iz frejma u frejm. U izveštaju prikazati pozicije ključnih tačaka na frejmovima 1, 5, 9 i 13. Prilikom estimacije optičkog toka koristiti prozor 5x5. Podesiti vrednost praga greške tako da se odbaci najveći broj loše detektovanih pomeraja. U okviru direktorijuma CanLK sačuvati slike svih frejmova sa obeleženim validnim ključnim tačkama.

Opis funkcije i implementacija

1. Proracun gradijenta tekuceg frejma po vremenu (I_t), po x-osi (I_x) i y-osi(I_y), za detekciju gradijenta po x i y osi koristi se funkcija i biblioteke OpenCV Sobel koja filtrira sliku Sobel-ovim filtrom koji sluzi kao aproksimacija gradijenata. Gradijent tekuceg frejma po vremenu se dobija kao razlika susednih frejmova.

2. Za svaku kljucnu tacku se posmatra okolina i pronalazi vektor pomeraja kao resenje sistema linearnih jednacina u srednje kvadratnom smislu. Podrazumevane pretpostavke algoritma su **konstantnost osvetljaja** data jednacinom $I_t + I_x u + I_y v = 0$ i **konstantnost pomeraja** koja podrazumeva da sve tacke iz neke okoline imaju isti pomeraj, odnosno fiksan vektor x.

Sistem jednacina:

$$Ax = b$$

$$A = [I_x\ I_y]$$

$$b = -[I_t]$$

$$x = [u, v],$$

gde su u i v pomeraju po x, y osi respektivno

In [ ]:
```
def lucasKanade(img1, img2, keyPoints, winSize, errThr):
    """
    Ulazi:
        img1 - ulazna siva slika trenutnog frejma.
        img2 - ulazna siva slika narednog frejma.
        keypoints - niz dimenzija (N,2) sa pozicijama ključnih tačaka.
        winSize - veličina prozora oko ključne tačke koji se koristi za estimaciju.
        errThr - prag za odbacivanje loše detektovanih tačaka.

    Izlaz:
        flowVectors - niz dimenzija (N,2) sa vrednostima komponenti pomeraja u svakoj ključnoj tački.
        validPoints - niz dimenzija N boolean vrednosti koji određuje da li je odgovarajuća ključna tačka
        pronađena u narednom frejmu. Ključna tačka se smatra izgubljenom ukoliko je njena nova pozicija
        izvan okvira frejma ili ukoliko je srednja kvadratna razlika normalizovanog prozora 3x3 oko stare i
        nove pozicije tačke veća od praga errThr.

    """
    flowVectors = np.zeros(keyPoints.shape, dtype=np.float32)
    validPoints = np.zeros(keyPoints.shape[0], dtype=bool)

    It = img2 - img1
    Ix = cv.Sobel(img1, cv.CV_64F, 1, 0, ksize=1)
    Iy = cv.Sobel(img1, cv.CV_64F, 0, 1, ksize=1)

    w = winSize//2

    height, width = img1.shape

    cnt = 0
    for point in keyPoints:
        j, i = np.int32(point)

        if i-w >= 0 and i+w+1<height and j-w>=0 and j+w+1<width:
            I_x = Ix[i-w:i+w+1, j-w:j+w+1].flatten()
            I_y = Iy[i-w:i+w+1, j-w:j+w+1].flatten()
            I_t = It[i-w:i+w+1, j-w:j+w+1].flatten()
```

```
            b = -np.reshape(I_t, (I_t.shape[0],1))
            A = np.vstack((I_x, I_y)).T

            # x[0, 0] = u -> dx
            # x[1, 0] = v -> dy
            x = np.linalg.inv(A.T@A)@A.T@b

            flowVectors[cnt, :] = x[:, 0]

            newPoint = np.round(keyPoints[cnt] + x[:, 0])
            newPoint = np.int32(newPoint)
            if newPoint[0]-1>=0 and newPoint[0]+2<width and newPoint[1]-1>=0 and newPoint[1]+2<height:
                temp1 = img1[i-1:i+2, j-1:j+2].flatten()
                temp2 = img2[newPoint[1]-1:newPoint[1]+2, newPoint[0]-1:newPoint[0]+2].flatten()

                if np.sum((temp1-temp2)**2) < errThr:
                    validPoints[cnt] = True

        cnt += 1

    return flowVectors, validPoints
```

Rezulatati

In [ ]:
```
# ucitavanje svih frejmova iz Can direktorijuma
dir_path = "imgs/Can"
file_name_list = os.listdir(dir_path)
file_name_list.sort()
```

In [ ]:
```
# ucitavanje prvog frejma
file_name = file_name_list[0]
file_path = os.path.join(dir_path, file_name)
img1 = cv.imread(file_path)
```

In [ ]:
```
# parametri Lucas-Kanade algoritma
errThr = 0.1
winSize = 5
```

In [ ]:
```
# detektovanje kljucnih tacaka
# goodFeaturesToTrack from OpenCV
"""
OpenCV documentation:
Maximum number of corners to return. If there are more corners than are found,
the strongest of them is returned. maxCorners <= 0 implies that no limit on the maximum is
set and all detected corners are returned.
"""
maxCorners = 0

"""
OpenCV documentation:
Parameter characterizing the minimal accepted quality of image corners.
The parameter value is multiplied by the best corner quality measure,
which is the minimal eigenvalue (see cornerMinEigenVal)
or the Harris function response (see cornerHarris ).
The corners with the quality measure less than the product are rejected.
For example, if the best corner has the quality measure = 1500,
and the qualityLevel=0.01 , then all the corners with the quality measure less than 15 are rejected.
"""
qualityLevel = 0.3

"""
OpenCV documentation:
Minimum possible Euclidean distance between the returned corners.
"""
minDistance = 10
img1_gray = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
img1_gray = np.float32(img1_gray/255.)
keyPoints = cv.goodFeaturesToTrack(img1_gray, maxCorners, qualityLevel, minDistance)[:, 0, :]
```

In [ ]:
```
def mark_points_on_image(img, points, color_BGR=[52, 235, 73]):
    """
    OpenCV documentation: cv.circle()
    img - Image where the circle is drawn.
    center - Center of the circle.
    radius - Radius of the circle.
    color - Circle color.
    thickness - Thickness of the circle outline, if positive.
                Negative values, like FILLED, mean that a filled circle is to be drawn.
```

```
    """
    radius = 2
    thickness = -1
    img_out = img
    for p in points:
        p = (np.int32(p[0]), np.int32(p[1]))
        img_out = cv.circle(img_out, p, radius, color_BGR, thickness)

    return img_out
```

In [ ]:
```
result_dir_path = "results/CanLK"
frames_with_keyPoints = []
keyPoints_list = []
flowVectors_list = []

frames_with_keyPoints.append(mark_points_on_image(img1, keyPoints))
keyPoints_list.append(keyPoints)

for i in range(1, len(file_name_list)):
    file_name = file_name_list[i]
    file_path = os.path.join(dir_path, file_name)
    img2 = cv.imread(file_path)
    img2_gray = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
    img2_gray = np.float32(img2_gray/255.)

    flowVectors, validPoints = lucasKanade(img1_gray, img2_gray, keyPoints, winSize, errThr)

    # updating key points
    keyPoints = keyPoints[validPoints, :]
    keyPoints += flowVectors[validPoints, :]

    frames_with_keyPoints.append(mark_points_on_image(img2, keyPoints))
    keyPoints_list.append(keyPoints)
    flowVectors_list.append(flowVectors[validPoints, :])

    img1_gray = img2_gray
```

In [ ]:
```
# cuvanje obradjenih frejmova u direktorijumu CanLK
for i in range(len(frames_with_keyPoints)):
```

```
    cv.imwrite(os.path.join(result_dir_path, file_name_list[i]), frames_with_keyPoints[i])
```

In [ ]:
```
# prikaz odgovarajucih frejmova
frame_id_to_show = [1, 5, 9, 13]

fig, ax = plt.subplots(figsize=FIGSIZE, dpi=DPI, nrows=2, ncols=len(frame_id_to_show)//2)
ax = ax.ravel()
for i in range(len(frame_id_to_show)):
    ax[i].imshow(cv.cvtColor(frames_with_keyPoints[frame_id_to_show[i]-1], cv.COLOR_BGR2RGB))
    ax[i].set_title(file_name_list[frame_id_to_show[i]-1] + "\nKey Points Number: {}".format(keyPoints_list[frame_i
    ax[i].set_xticks([]), ax[i].set_yticks([])
    plt.tight_layout()
```

frame01.jpg
Key Points Number: 43

frame05.jpg
Key Points Number: 6

frame09.jpg
Key Points Number: 4

frame13.jpg
Key Points Number: 2

Komentar:

Lucas-Kanade algoritam za procenu optickog toka je izveden pod pretpostavkom da su pomeraju manji od jednog piskela, sto u sekvenci frejmova nije slucaj, tako da algoritam ima jako lose performanse.

ZADATAK 2 – Realizacija iterativne Lucas-Kanade metode optičkog toka

Prethodno realizovana funkcija podrazumeva male pomeraje ključnih tačaka između frejmova. U slučaju većih pomeraja javlja se značajna greška u proceni optičkog toka. Zbog toga je potrebno realizovati poboljšanu verziju prethodne funkcije u kjoj se procena vektora pomeraja obavlja iterativno. Na početku se svi vektori pomeraja inicijalizuju na osnovu ulaza initFlow a u svakoj narednoj iteraciji se koristi prethodno procenjena vrednost. Prilikom računanja temporalnog gradijenta posmatra se razlika piksela na poziciji ključne tačke trenutnog frejma i piksela na poziciji određenoj vektorom pomeraja (iz prethodne iteracije) narednog frejma. Funkciju nazvati **iterLucasKanade**. Argumenti funkcije su:

Ulazi:
img1 - ulazna siva slika trenutnog frejma.
img2 - ulazna siva slika narednog frejm.
keypoints – niz dimenzija (N,2) sa pozicijama ključnih tačaka.
winSize - veličina prozora oko ključne tačke koji se koristi za estimaciju.
numIter - broj iteracija.
initFlow - niz dimenzija (N,2) sa inicijalnim vrednostima vektora pomeraja u svakoj ključnoj tački
errThr - prag za odbacivanje loše detektovanih tačaka.

Izlaz:
flowVectors - niz dimenzija (N,2) sa vrednostima komponenti pomeraja u svakoj ključnoj tački.
validPoints - niz dimenzija N boolean vrednosti koji određuje da li je odgovarajuća ključna tačk pronađena u narednom frejmu. Ključna tačka se smatra izgubljenom ukoliko je njena nova pozicija izvan okvira frejma ili ukoliko je srednja kvadratna razlika normalizovanog prozora 3x3 oko stare i nove pozicije tačke veća od praga *errThr*.

Testirati implementiranu funkciju na sekvenci slika iz direktorijuma **Can**. Testiranje obaviti na način opisan u zadatku 1. Prilikom testiranja koristiti veličinu prozora 9x9 i broj iteracija 7. Smatrati da je inicijalna vrednost pomeraja jednaka 0. U okviru direktorijuma **CanLKiter**

sačuvati slike svih frejmova sa obeleženim validnim ključnim tačkama.

Opis funckije i implementacija

Ideja iterativnog Lucas-Kanade algoritma je da se prethodno implementirani algoritam ponavlja vise puta.

1. Proracun gradijenata po x i y osi se obavlja na pocetku koristeci Sobel-ov filtar kao aproksimaciju gradijenta.
2. Iterativna procedura koja pocinje proracunom gradijenta po vremenu koristeci konacnu razliku kao aproksimaciju, resavanje sistema linearnih jednacina u srednje kvadratnom simislu za dobijanje vektora pomeraja. Azuriranje pozicije validnih kljucnih tacaka.

In [ ]:
```
def iterLucasKanade(img1, img2, keyPoints, winSize, numIter, initFlow, errThr):
    """
    Ulazi:
        img1 - ulazna siva slika trenutnog frejma.
        img2 - ulazna siva slika narednog frejma.
        keypoints - niz dimenzija (N,2) sa pozicijama ključnih tačaka.
        winSize - veličina prozora oko ključne tačke koji se koristi za estimaciju.numIter - broj iteracija.
        initFlow - niz dimenzija (N,2) sa inicijalnim vrednostima vektora pomeraja u svakoj ključnoj tački.
        errThr - prag za odbacivanje loše detektovanih tačaka.
    Izlaz:
        flowVectors - niz dimenzija (N,2) sa vrednostima komponenti pomeraja u svakoj ključnoj tački.
        validPoints - niz dimenzija N boolean vrednosti koji određuje da li je odgovarajuća ključna tačka
                      pronađena u narednom frejmu. Ključna tačka se smatra izgubljenom ukoliko je njena nova pozici
                      izvan okvira frejma ili ukoliko je srednja kvadratna razlika normalizovanog prozora 3x3 oko s
                      nove pozicije tačke veća od praga errThr.
    """

    flowVectors = np.zeros(keyPoints.shape, dtype=np.float32)
    validPoints = np.zeros(keyPoints.shape[0], dtype=bool)

    Ix = cv.Sobel(img1, cv.CV_64F, 1, 0, ksize=1)
    Iy = cv.Sobel(img1, cv.CV_64F, 0, 1, ksize=1)
    w = winSize//2

    height, width = img1.shape

    new_keyPoints = keyPoints + initFlow
```

```
    while numIter>0:
        for k in range(keyPoints.shape[0]):
            j, i = np.int32(keyPoints[k, :])
            j_new, i_new = np.int32(new_keyPoints[k, :])

            if i-w >= 0 and i+w+1<height and j-w>=0 and j+w+1<width and \
                i_new-w >= 0 and i_new+w+1<height and j_new-w>=0 and j_new+w+1<width:

                    I_x = Ix[i-w:i+w+1, j-w:j+w+1].flatten()
                    I_y = Iy[i-w:i+w+1, j-w:j+w+1].flatten()
                    It = img2[i_new-w:i_new+w+1, j_new-w:j_new+w+1] - img1[i-w:i+w+1, j-w:j+w+1]
                    I_t = It.flatten()
                    b = -np.reshape(I_t, (I_t.shape[0],1))
                    A = np.vstack((I_x, I_y)).T

                    # x[0, 0] = u -> dx
                    # x[1, 0] = v -> dy
                    x = np.linalg.inv(A.T@A)@A.T@b

                    flowVectors[k, :] += x[:, 0]

                    new_keyPoints[k, :] += x[:, 0]

        numIter -= 1


    for cnt in range(keyPoints.shape[0]):
        j, i = np.int32(keyPoints[cnt, :])
        j_new, i_new = np.int32(new_keyPoints[cnt, :])
        if j_new-1>=0 and j_new+2<width and i_new-1>=0 and i_new+2<height:
            temp1 = img1[i-1:i+2, j-1:j+2].flatten()
            temp2 = img2[i_new-1:i_new+2, j_new-1:j_new+2].flatten()

            if np.sum((temp1-temp2)**2) < errThr:
                validPoints[cnt] = True

    return flowVectors, validPoints
```

In [ ]:
```
# ucitavanje prvog frejma
file_name = file_name_list[0]
```

```
file_path = os.path.join(dir_path, file_name)
img1 = cv.imread(file_path)
```

In [ ]:
```
# parametri Lucas-Kanade algoritma
errThr = 0.2
winSize = 9
numIter = 7
```

In [ ]:
```
# detektovanje kljucnih tacaka
# goodFeaturesToTrack from OpenCV
maxCorners = 0
qualityLevel = 0.3
minDistance = 10
img1_gray = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
img1_gray = np.float32(img1_gray/255.)
keyPoints = cv.goodFeaturesToTrack(img1_gray, maxCorners, qualityLevel, minDistance)[:, 0, :]
initFlow = np.zeros(keyPoints.shape) # poctni pomeraj
```

In [ ]:
```
result_dir_path = "results/CanLKiter"
frames_with_keyPoints = []
keyPoints_list = []
flowVectors_list = []

frames_with_keyPoints.append(mark_points_on_image(img1, keyPoints))
keyPoints_list.append(keyPoints)

for i in range(1, len(file_name_list)):
    file_name = file_name_list[i]
    file_path = os.path.join(dir_path, file_name)
    img2 = cv.imread(file_path)
    img2_gray = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
    img2_gray = np.float32(img2_gray/255.)
    initFlow = np.zeros(keyPoints.shape) # poctni pomeraj
    flowVectors, validPoints = iterLucasKanade(img1_gray, img2_gray, keyPoints,\
                                               winSize, numIter, initFlow, errThr)
    # updating key points
    keyPoints = keyPoints[validPoints, :]
    keyPoints += flowVectors[validPoints, :]

    frames_with_keyPoints.append(mark_points_on_image(img2, keyPoints))
```

```
    keyPoints_list.append(keyPoints)
    flowVectors_list.append(flowVectors[validPoints, :])

    img1_gray = img2_gray
```

In [ ]:
```
# cuvanje obradjenih frejmova u direktorijumu CanLK
for i in range(len(frames_with_keyPoints)):
    cv.imwrite(os.path.join(result_dir_path, file_name_list[i]), frames_with_keyPoints[i])
```

In [ ]:
```
# prikaz odgovarajucih frejmova
frame_id_to_show = [1, 5, 9, 13]

fig, ax = plt.subplots(figsize=FIGSIZE, dpi=DPI, nrows=2, ncols=len(frame_id_to_show)//2)
ax = ax.ravel()
for i in range(len(frame_id_to_show)):
    ax[i].imshow(cv.cvtColor(frames_with_keyPoints[frame_id_to_show[i]-1], cv.COLOR_BGR2RGB))
    ax[i].set_title(file_name_list[frame_id_to_show[i]-1] + "\nKey Points Number: {}".format(keyPoints_list[frame_i
    ax[i].set_xticks([]), ax[i].set_yticks([])
    plt.tight_layout()
```

frame01.jpg
Key Points Number: 43

frame05.jpg
Key Points Number: 29

frame09.jpg
Key Points Number: 25

frame13.jpg
Key Points Number: 24

Komentar

Ovaj algoritam ima dosta bolje performase od prethodnog jer, moze da isprati pomeraje vece od jednog piksela ali je sklon zaglavljivanju u lokalne minimume zbog preklapanja ("aliasing").

ZADATAK 3 – Realizacija piramidalne Lucas-Kanade metode optičkog toka

U slučaju velikih pomeraja može doći do temporalnog alijasinga i iterativna procena optičkog toka može iskonvergirati u lokalni minimum. Kako bi se omogućila detekcija velikih pomeraja ideja je da se iskoristi Gausova piramida. Na taj način se veliki pomeraji na višim nivoima piramide svode na manje pomeraje i moguće ih je proceniti korišćenjem iterativnog metoda. Funkciju nazvati **pyrLucasKanade**. Argumenti funkcije su:

Ulazi:
img1 - ulazna siva slika trenutnog frejma.
img2 - ulazna siva slika narednog frejm.
keypoints – niz dimenzija (N,2) sa pozicijama ključnih tačaka.
winSize - veličina prozora oko ključne tačke koji se koristi za estimaciju.
numIter - broj iteracija.
levelNum - broj nivoa Gausove piramide.
levelScale - faktor skaliranja između susednih nivoa piramide (npr ima vednost 2 ukoliko se radi decimacija 2 puta po obe dimenzije)
errThr - prag za odbacivanje loše detektovanih tačaka.

Izlaz:
flowVectors - niz dimenzija (N,2) sa vrednostima komponenti pomeraja u svakoj ključnoj tački.
validPoints - niz dimenzija N boolean vrednosti koji određuje da li je odgovarajuća ključna tačk pronađena u narednom frejmu. Ključna tačka se smatra izgubljenom ukoliko je njena nova pozicija izvan okvira frejma ili ukoliko je srednja kvadratna razlika normalizovanog prozora 3x3 oko stare i nove pozicije tačke veća od praga *errThr*.

Prilikom implementacije funkcije za kreiranje Gausove piramide može se iskoristiti funkcija *pyramid_gaussian* iz skimage.transform biblioteke. Voditi računa da se pri prelasku na novi nivo piramide skaliraju i pozicije ključnih tačaka.

Testirati implementiranu funkciju na sekvenci slika iz direktorijuma **Can**. Testiranje obaviti na način opisan u zadatku 1. Prilikom testiranja

koristiti veličinu prozora 9x9, broj iteracija 7, ukupno 3 nivoa Gausove piramide, sa faktorom decimacije 2 između nivoa. U okviru direktorijuma **CanLKpyr** sačuvati slike svih frejmova sa obeleženim validnim ključnim tačkama.

Opis funckije i implementacija

Ideja kod ovog pristupa je da se kreiranjem Gausove piramide pomeraji posmatraju na vise skala, i tako ce veliki pomeraji na "malim" skalama postati mali pomeraji na "velikim" skalama, pa ce iterativni Lucas-Kanade algoritam moci da detektuje te pomeraje.

1. konstrukcije Gausove piramide
2. pozivanje iterativnog Lucas-Kanade algoritma pocevsi od najnizeg nivoa, uzimajuci u obzir pomeraje detektovane na vecoj skali

In [ ]:
```
from skimage.transform import pyramid_gaussian
def pyrLucasKanade(img1, img2, keyPoints, winSize, numIter,\
                   levelNum, levelScale, errThr):
    flowVectors = np.zeros(keyPoints.shape, dtype=np.float32)
    validPoints = np.ones(keyPoints.shape[0], dtype=bool)

    initFlow = np.zeros(keyPoints.shape)

    pyr1 = tuple(pyramid_gaussian(img1, max_layer=levelNum, downscale=levelScale))
    pyr2 = tuple(pyramid_gaussian(img2, max_layer=levelNum, downscale=levelScale))

    for level in range(levelNum, -1, -1):
        fv, vp = iterLucasKanade(pyr1[level], pyr2[level], keyPoints/levelScale**level,
                                 winSize, numIter, initFlow/levelScale**level, errThr)
        flowVectors[vp] += fv[vp]*levelScale**level
        initFlow = flowVectors
        validPoints *= vp


    return flowVectors, validPoints
```

In [ ]:
```
# ucitavanje prvog frejma
file_name = file_name_list[0]
file_path = os.path.join(dir_path, file_name)
img1 = cv.imread(file_path)
```

In [ ]:
```
# parametri piramidalnog Lucas-Kanade algoritma
winSize = 9
numIter = 7
levelNum = 3
levelScale = 2
errThr = 0.2
```

In [ ]:
```
# detektovanje kljucnih tacaka
# goodFeaturesToTrack from OpenCV
maxCorners = 0
qualityLevel = 0.3
minDistance = 10
img1_gray = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
img1_gray = np.float32(img1_gray/255.)
keyPoints = cv.goodFeaturesToTrack(img1_gray, maxCorners, qualityLevel, minDistance)[:, 0, :]
```

In [ ]:
```
result_dir_path = "results/CanLKpyr"
frames_with_keyPoints = []
keyPoints_list = []
flowVectors_list = []

frames_with_keyPoints.append(mark_points_on_image(img1, keyPoints))
keyPoints_list.append(keyPoints)

for i in range(1, len(file_name_list)):
    file_name = file_name_list[i]
    file_path = os.path.join(dir_path, file_name)
    img2 = cv.imread(file_path)
    img2_gray = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
    img2_gray = np.float32(img2_gray/255.)
    initFlow = np.zeros(keyPoints.shape) # poctni pomeraj
    flowVectors, validPoints = pyrLucasKanade(img1_gray, img2_gray, keyPoints, winSize, numIter,\
                                              levelNum, levelScale, errThr)
    # updating key points
    keyPoints = keyPoints[validPoints, :]
    keyPoints += flowVectors[validPoints, :]

    frames_with_keyPoints.append(mark_points_on_image(img2, keyPoints))
    keyPoints_list.append(keyPoints)
    flowVectors_list.append(flowVectors[validPoints, :])
```

```
    img1_gray = img2_gray
```

In [ ]:
```
# cuvanje obradjenih frejmova u direktorijumu CanLK
for i in range(len(frames_with_keyPoints)):
    cv.imwrite(os.path.join(result_dir_path, file_name_list[i]), frames_with_keyPoints[i])
```

In [ ]:
```
# prikaz odgovarajucih frejmova
frame_id_to_show = [1, 5, 9, 13]

fig, ax = plt.subplots(figsize=FIGSIZE, dpi=DPI, nrows=2, ncols=len(frame_id_to_show)//2)
ax = ax.ravel()
for i in range(len(frame_id_to_show)):
    ax[i].imshow(cv.cvtColor(frames_with_keyPoints[frame_id_to_show[i]-1], cv.COLOR_BGR2RGB))
    ax[i].set_title(file_name_list[frame_id_to_show[i]-1] + "\nKey Points Number: {}".format(keyPoints_list[frame_i
    ax[i].set_xticks([]), ax[i].set_yticks([])
    plt.tight_layout()
```

frame01.jpg
Key Points Number: 43

frame05.jpg
Key Points Number: 38

frame09.jpg
Key Points Number: 33

frame13.jpg
Key Points Number: 31

Komentar:

Piramidalna realizacija algoritma za detekciju pomeraja ima najbolje performanse jer na vecim skalama "veliki" poemraju postaju "mali", sto iterativni Lucas-Kanade algoritam moze da isprati.

ZADATAK 4 – Praćenje objekta

U okviru ovog zadatka potrebno je realizovati praćenje lica čoveka na sekvenci slika. Slike se nalaze u okviru direktorijuma **Man**. Potrebno je na prvoj slici iz sekvence ručno definisati pravougaonik koji obuhvata lice čoveka. Pravougaonik se zadaje preko koordinata gornjeg levog ugla, visine i širine. Tokom postupka praćenja menja se samo koordinata gornjeg levog ugla dok su dimenzije pravougaonika konstantne. Praćenje se obavlja tako što se na početku detektuju ključne tačke u okviru definisanog pravougaonika. Nakon toga se koriste algoritmi optičkog toka realizovani u prethodnim tačkama za određivanje pomeraja ključnih tačaka od frejma do frejma. Pomeraj pravougaonika oko lica predstavlja srednju vrednost pomeraja svih validnih ključnih tačaka. Nakon određivanja nove pozicije pravougaonika odbacuju se sve one ključne tačke koje se nalaze van njegovih granica. Kako se na ovaj način broj ključnih tačaka smanjuje s vremenom, potrebno je na svakih 20 frejmova ponovo detektovati ključne tačke u okviru pravougaonika od interesa.

Potrebno je sačuvati dve izlazne video sekvence. Jedna video sekvenca (**man_track.mp4**) predstavlja originalni video sa iscrtanim pravougaonikom oko lica na procenjenim pozicijama. Druga video sekvenca ima ucrtane i validne karakteristične tačke (**man_track_key.mp4**).

Opis i implementacija

Algoritam za pracenje lica:

1. Inicijalna detekcija lica definisanje pravougaonika oko lica na prvom frejmu
2. Detekcija kljuvnih tacaka unutar tog pravugaonika
3. Iteriranje kroz sve frejmove estimacija pomeraja na dva susedna frejma
4. Pomeranje pravougaonika za srednju vrednost pomeraja svih tacaka u pracenju
5. Na svakih 20 frejmova se vrsi reakvizicija kljucnih tacaka jer se neke kljucne tacke postaju nevalidne kljucne tacke

In [ ]:
```
def cut_rectangle(img_gray, upper_left_corner, height, width):
```

```
    img_cut = img_gray[upper_left_corner[1]:upper_left_corner[1]+height,\
                       upper_left_corner[0]:upper_left_corner[0]+width]
    return img_cut
```

In [ ]:
```
# ucitavanje svih frejmova iz Man direktorijuma
dir_path = "imgs/Man"
file_name_list = os.listdir(dir_path)
file_name_list.sort()
```

In [ ]:
```
# ucitavanje prvog frejma
file_name = file_name_list[0]
file_path = os.path.join(dir_path, file_name)
img1 = cv.imread(file_path)
# definisanje pravugaonika
upper_left_corner = np.array([67, 47])
height = 37
width = 33
# detektovanje kljucnih tacaka
# goodFeaturesToTrack from OpenCV
maxCorners = 0
qualityLevel = 0.1
minDistance = 1
img1_gray = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
img1_gray = np.float32(img1_gray/255.)
keyPoints = cv.goodFeaturesToTrack(cut_rectangle(img1_gray, upper_left_corner, height, width), maxCorners, qualityL
keyPoints += upper_left_corner

FPS = 30
result_file_name_1 = "results/man_track.mp4"
result1 = cv.VideoWriter(result_file_name_1, cv.VideoWriter_fourcc(*'MP4V'), FPS, (img1_gray.shape[1], img1_gray.sh
result_file_name_2 = "results/man_track_key.mp4"
result2 = cv.VideoWriter(result_file_name_2, cv.VideoWriter_fourcc(*'MP4V'), FPS, (img1_gray.shape[1], img1_gray.sh
winSize = 9
numIter = 7
levelNum = 3
levelScale = 2
errThr = 0.1

reacq = 20
```

```
for i in range(1, len(file_name_list)):
    file_name = file_name_list[i]
    file_path = os.path.join(dir_path, file_name)
    img2 = cv.imread(file_path)
    img2_gray = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
    img2_gray = np.float32(img2_gray/255.)

    initFlow = np.zeros(keyPoints.shape)
    flowVectors, validPoints = pyrLucasKanade(img1_gray, img2_gray, keyPoints, winSize, numIter,\
                                              levelNum, levelScale, errThr)

    keyPoints += np.int32(flowVectors)

    upper_left_corner = upper_left_corner + np.mean(flowVectors[validPoints, :], axis=0)
    upper_left_corner = np.int32(upper_left_corner)

    for j in range(keyPoints.shape[0]):
        if keyPoints[j, 0] < upper_left_corner[0] or keyPoints[j, 0] > upper_left_corner[0]+width:
            validPoints[j] = False
        if keyPoints[j, 1] < upper_left_corner[1] or keyPoints[j, 1] > upper_left_corner[1]+height:
            validPoints[j] = False

    keyPoints = keyPoints[validPoints, :]
    img1_gray = img2_gray

    img_out = cv.rectangle(img2, (upper_left_corner[0], upper_left_corner[1]),\
                        (upper_left_corner[0]+width, upper_left_corner[1]+height),\
                        (12, 250, 5), 1)

    result1.write(img_out)

    img_out = mark_points_on_image(img_out, keyPoints)

    result2.write(img_out)

    if reacq == 0:
        keyPoints = cv.goodFeaturesToTrack(cut_rectangle(img1_gray, upper_left_corner, height, width), maxCorners,
        keyPoints += upper_left_corner
        reacq = 20
    else:
        reacq -= 1
```

```
result1.release()
result2.release()
```

```
OpenCV: FFMPEG: tag 0x5634504d/'MP4V' is not supported with codec id 12 and format 'mp4 / MP4 (MPEG-4 Part 14)'
OpenCV: FFMPEG: fallback to use tag 0x7634706d/'mp4v'
OpenCV: FFMPEG: tag 0x5634504d/'MP4V' is not supported with codec id 12 and format 'mp4 / MP4 (MPEG-4 Part 14)'
OpenCV: FFMPEG: fallback to use tag 0x7634706d/'mp4v'
```

Komentar:

Performase algoritama su zadovoljavajuce, ondosno ne gubi se detekcija lica coveka.

BONUS ZADATAK – Stabilizacija videa

Snimiti ili preuzeti neku video sekvencu u kojoj jasno postoje globalni pokreti kamere. Dobar primer je snimanje videa iz ruke u toku hodanja ukolko uređaj kojim se snima nema uključenu stabilizaciju. Pre obrade izvršiti skaliranje frejmova kako bi se ubrzalo procesiranje. U toku procesiranja potrebno je odrediti i globalno kompenzovati dominantne pomeraje u video sekvenci. Dominantni pomeraji nisu najveći pomeraji već oni koji se konzistentno javljaju na najvećem broju piksela. U izveštaju detaljno opisati primenjeno rešenje i prikazati međurezultate procesiranja. Stabilizovanu video sekvencu snimiti u poseban video fajl.

Opis i implementacija

Kao i u zadatku 4 potrebno je da se prate kljucne tacke. Kao procena dominantnog poremraja dva susedna frejma koristi se aritmeticka sredina, svih validno detektovanih vektora pomeraja. Ideja je da se vecina vektora pomeraja imati dominantu komponentu pomeraja i da ce odvuci srednju vrednost ka sebi, sto znaci da ce ova estimacija biti pomerena u zavisnosti od broja i vrednosti vektora pomeraja koji nisu nemaju dominantnu komponentu. Ocekivani rezulatat je da se dominanti pomeraj nece brzo menjati odnosno da ce biti sporo promenljiva velicina zbog toga je uveden filtar prvog reda koji treba da potisne sum, jednacina filtra je:

$$X[t] = (1-\alpha)\hat{X}[t] + \alpha X[t-1],$$

Gde je $X = median(flowVectors)$. Kao algoriram procene optickog toga koristi se piramidalni Lucas-Kanade algoritam, a kad broj validnih kljucnih tacaka padne ispod praga pokrece se trazenje kljucnih tacaka.

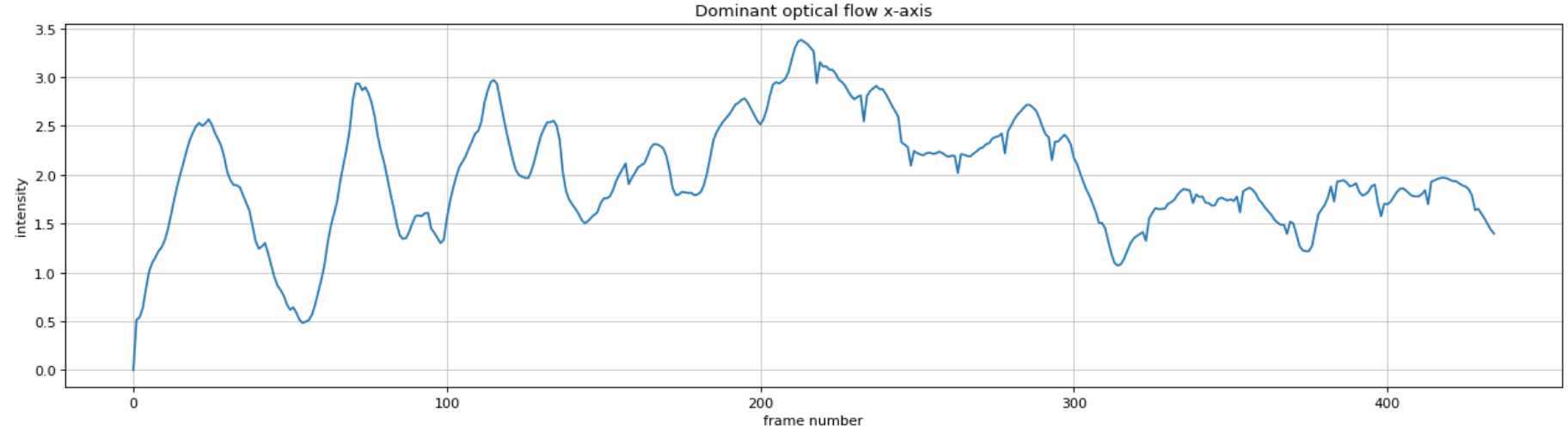
In [ ]:
```
input_video_path = 'imgs/unstabilized_video.mp4'

cap = cv.VideoCapture(input_video_path)
FPS = cap.get(cv.CAP_PROP_FPS)
# FPS = 10
FRAME_SHAPE = (250, 250)
winSize = 9
numIter = 7
levelNum = 3
levelScale = 2
errThr = 0.5

ret, frame = cap.read()
frame = cv.resize(frame, FRAME_SHAPE, interpolation = cv.INTER_LINEAR)
frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
frame_gray = np.float32(frame_gray/255.)

h, w = frame_gray.shape

result_file_name = "results/stabilized_video.mp4"
result = cv.VideoWriter(result_file_name, cv.VideoWriter_fourcc(*'MP4V'),
                        FPS, (2*w, h))
maxCorners = 0
qualityLevel = 0.3
```

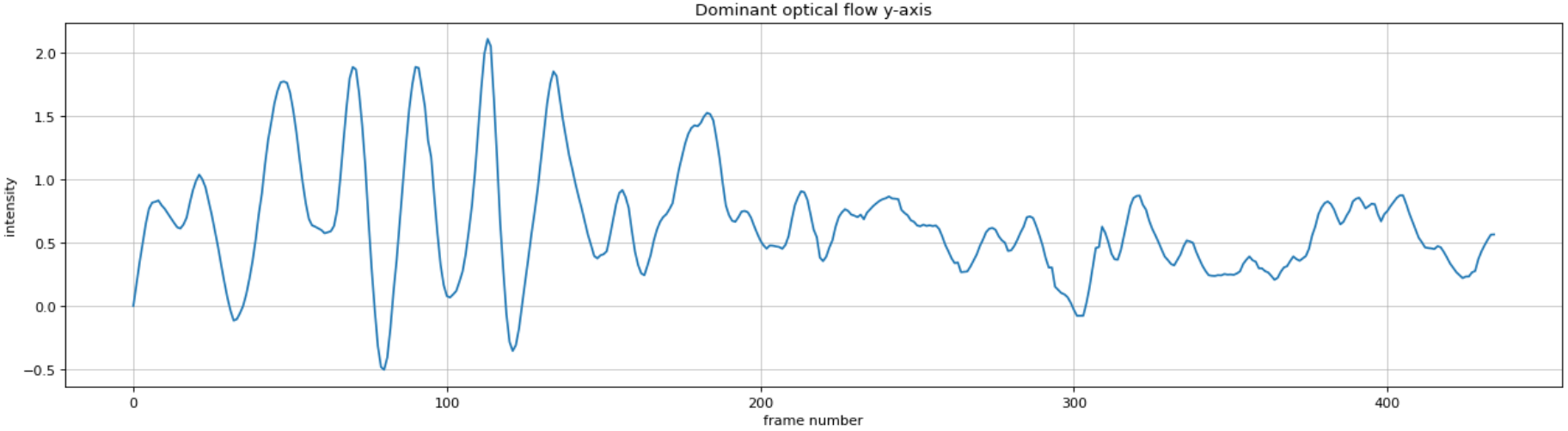
```
minDistance = 0
keyPoints = cv.goodFeaturesToTrack(frame_gray, maxCorners, qualityLevel, minDistance)[:, 0, :]
fv = []
ALPHA = 0.9
u_last, v_last = 0, 0
img_out = np.zeros(shape=(h, w*2, 3), dtype=np.uint8)

key_points_num = 5
while(cap.isOpened()):
    ret, frame_new = cap.read()
    if ret:
        frame_new = cv.resize(frame_new, FRAME_SHAPE, interpolation = cv.INTER_LINEAR)
        frame_new_gray = cv.cvtColor(frame_new, cv.COLOR_BGR2GRAY)
        frame_new_gray = np.float32(frame_new_gray/255.)
        flowVectors, validPoints = pyrLucasKanade(frame_gray, frame_new_gray,\
                                                  keyPoints, winSize, numIter,\
                                                  levelNum, levelScale, errThr)

        keyPoints += np.int32(flowVectors)
        keyPoints = keyPoints[validPoints, :]


        if (flowVectors[validPoints, :].shape[0] > key_points_num):
            u, v = np.mean(flowVectors[validPoints, :], axis=0)

        else:
            keyPoints = cv.goodFeaturesToTrack(frame_new_gray, maxCorners,\
                                               qualityLevel, minDistance)[:, 0, :]
            u, v = 0, 0

        u, v = ALPHA*u_last+(1-ALPHA)*u, ALPHA*v_last+(1-ALPHA)*v
        u_last, v_last = u, v

        fv.append([u, v])

        T = np.float32([[1, 0, -u], [0, 1, -v]])
        stabilized_frame = cv.warpAffine(frame_new, T, (w, h))
        frame = frame_new
        frame_gray = frame_new_gray

        img_out[:, 0:w, :] = frame
```

```
        img_out[:, w::, :] = stabilized_frame
        result.write(img_out)
    else:
        break

cap.release()
result.release()
```

```
OpenCV: FFMPEG: tag 0x5634504d/'MP4V' is not supported with codec id 12 and format 'mp4 / MP4 (MPEG-4 Part 14)'
OpenCV: FFMPEG: fallback to use tag 0x7634706d/'mp4v'
```

In [ ]:
```
fv = np.array(fv)
fig, ax = plt.subplots(figsize=FIGSIZE, dpi=DPI, nrows=2)
ax = ax.ravel()
ax[0].plot(fv[:, 0])
ax[0].set_title("Dominant optical flow x-axis")
ax[0].grid("on", alpha=0.7)
ax[0].set_xlabel("frame number")
ax[0].set_ylabel("intensity")

ax[1].plot(fv[:, 1])
ax[1].set_title("Dominant optical flow y-axis")
ax[1].grid("on", alpha=0.7)
ax[1].set_xlabel("frame number")
ax[1].set_ylabel("intensity")

plt.tight_layout()
```


Dominant optical flow x-axis
intensity
frame number
0
100
200
300
400
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Dominant optical flow y-axis
intensity
frame number
0
100
200
300
400
−0.5
0.0
0.5
1.0
1.5
2.0