optical flow

February 18, 2024

[ ]:
```
import glob
import skimage
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import scipy
pngs = glob.glob('toyProblem_F22/frame_*.png')
```

[ ]:
```
print(f"skimage version: {skimage.__version__}")
print(f"matplotlib version: {matplotlib.__version__}")
print(f"numpy version: {np.__version__}")
print(f"scipy version: {scipy.__version__}")
```

```
skimage version: 0.22.0
matplotlib version: 3.6.3
numpy version: 1.26.3
scipy version: 1.12.0
```

1 Problem 1 - Loading and displaying a toy problem:

When images are read, it is read as (y,x) instead of (x,y).

[ ]:
```
# make image list
ims_with_color = []
ims = []
for i in pngs:
    # read in image in grayscale
    ims_with_color.append(plt.imread(i))
    # read in image in grayscale
    ims.append(skimage.color.rgb2gray(plt.imread(i)))

# make image array
V = np.dstack(ims)
V.shape
```

[ ]:
```
# animation of images in an qt window
%matplotlib qt
```

```
for im in ims:
    plt.imshow(im, cmap='gray')
    plt.pause(0.1)
%matplotlib inline
```

2 Problem 2.1 - Low level gradient calculation:

[ ]:
```
# Compute the low level gradient in x, y and t direction
Vy = V[1:, :, :] - V[0:-1, :, :]
Vx = V[:, 1:, :] - V[:, 0:-1, :]
Vt = V[:, :, 1:] - V[:, :, 0:-1]
Vx.shape, Vy.shape, Vt.shape
```

[ ]:
```
# Select some images
indexes = [0, 10, 20]

# Show gradient images
for i in indexes:
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    plt.title('frame {}'.format(i))
    ax[0].imshow(Vx[:, :, i], cmap='gray')
    ax[0].set_title('Vx')
    ax[1].imshow(Vy[:, :, i], cmap='gray')
    ax[1].set_title('Vy')
    ax[2].imshow(Vt[:, :, i], cmap='gray')
    ax[2].set_title('Vt')
    fig.suptitle('frame {}'.format(i))
    plt.show()
```

3 Problem 2.2 - Simple Gradient Filters:

3.1 The kernel

[ ]:
```
prewitt_h_kernel = np.array([
    [-1, 0, 1],
    [-1, 0, 1],
    [-1, 0, 1]
])
prewitt_v_kernel = np.array([
    [-1, -1, -1],
    [0, 0, 0],
    [1, 1, 1]
])

prewitt_h_kernel, prewitt_v_kernel
```

[ ]:
```
# Select some images
indexes = [0, 10, 20]

# Show gradient images
for i in indexes:
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    plt.title('frame {}'.format(i))
    ax[0].imshow(ims[i], cmap='gray')
    ax[0].set_title('Original')
    ax[1].imshow(scipy.ndimage.prewitt(ims[i], axis=1), cmap='gray')
    ax[1].set_title('Prewitt horizontal')
    ax[2].imshow(scipy.ndimage.prewitt(ims[i], axis=0), cmap='gray')
    ax[2].set_title('Prewitt vertical')
    fig.suptitle('frame {}'.format(i))
    plt.show()
```

[ ]:
```
# Select some images
indexes = [0, 10, 20]

# Show gradient images
for i in indexes:
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    plt.title('frame {}'.format(i))
    ax[0].imshow(ims[i], cmap='gray')
    ax[0].set_title('Original')
    ax[1].imshow(scipy.ndimage.sobel(ims[i], axis=0), cmap='gray')
    ax[1].set_title('Sobel horizontal')
    ax[2].imshow(scipy.ndimage.sobel(ims[i], axis=1), cmap='gray')
    ax[2].set_title('Sobel vertical')
    fig.suptitle('frame {}'.format(i))
    plt.show()
```

[ ]:
```
sobel_h_kernel = np.array([
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]
])
sobel_v_kernel = np.array([
    [-1, -2, -1],
    [0, 0, 0],
    [1, 2, 1]
])
```

Both give vertical gradient.

3.2 Image filtering:

[ ]:
```
# Apply prewitt filter to the video along the x, y and t axis
Vx_prewitt = scipy.ndimage.prewitt(V, axis=1)
Vy_prewitt = scipy.ndimage.prewitt(V, axis=0)
Vt_prewitt = scipy.ndimage.prewitt(V, axis=2)
for i in indexes:
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    plt.title('frame {}'.format(i))
    ax[0].imshow(Vx_prewitt[:, :, i], cmap='gray')
    ax[0].set_title('Vx_prewitt')
    ax[1].imshow(Vy_prewitt[:, :, i], cmap='gray')
    ax[1].set_title('Vy_prewitt')
    ax[2].imshow(Vt_prewitt[:, :, i], cmap='gray')
    ax[2].set_title('Vt_prewitt')
    fig.suptitle('frame {}'.format(i))
    plt.show()
```

4 Problem 2.3 - Gaussian Gradient Filters:

$$G(x,y,z) = \frac{1}{(2\pi\sigma^2)^{\frac{3}{2}}} e^{-\frac{x^2+y^2+z^2}{2\sigma^2}}$$

[ ]:
```
# Gaussian filter
sigma = 1
gaussian_V = scipy.ndimage.gaussian_filter(V, sigma, order=1)

# animation of images in an qt window
%matplotlib qt
for i in range(gaussian_V.shape[2]):
    plt.imshow(gaussian_V[:,:,i], cmap='gray')
    plt.pause(0.1)
%matplotlib inline

# Apply first derivative gaussian filter to the video along the x, y and t axis
Vx_gaussian = scipy.ndimage.gaussian_filter(V, sigma, order=1, axes=1)
Vy_gaussian = scipy.ndimage.gaussian_filter(V, sigma, order=1, axes=0)
Vt_gaussian = scipy.ndimage.gaussian_filter(V, sigma, order=1, axes=2)
Vx_gaussian.shape, Vy_gaussian.shape, Vt_gaussian.shape
```

[ ]:
```
# Select some images
indexes = [0,10,20]

# Select standard deviation
sigma = 1

# Show first derivative gaussian filter
```

```
for i in indexes:
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    plt.title('frame {}'.format(i))
    ax[0].imshow(Vx_gaussian[:, :, i], cmap='gray')
    ax[0].set_title('Vx_gaussian')
    ax[1].imshow(Vy_gaussian[:, :, i], cmap='gray')
    ax[1].set_title('Vy_gaussian')
    ax[2].imshow(Vt_gaussian[:, :, i], cmap='gray')
    ax[2].set_title('Vt_gaussian')
    fig.suptitle('frame {} with sigma {}'.format(i, sigma))
    plt.show()
```

[ ]:
```
# Select some images
indexes = [0,10,20]

# Select another standard deviation
sigma = 0.5

# Show first derivative gaussian filter
for i in indexes:
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    plt.title('frame {}'.format(i))
    ax[0].imshow(Vx_gaussian[:, :, i], cmap='gray')
    ax[0].set_title('Vx_gaussian')
    ax[1].imshow(Vy_gaussian[:, :, i], cmap='gray')
    ax[1].set_title('Vy_gaussian')
    ax[2].imshow(Vt_gaussian[:, :, i], cmap='gray')
    ax[2].set_title('Vt_gaussian')
    fig.suptitle('frame {} with sigma {}'.format(i, sigma))
    plt.show()
```

5 Problem 3.1 - Local and low level solution:

[ ]:
```
# Select pixel located in (10,10,10)

x,y,t = [10,10,10]
# N * N is the size of the region, n is a helper variable
N = 5
n = int((N-1)//2)

# Show the region
im = ims[t]
x_lower, x_upper = np.maximum(0, x-n), np.minimum(im.shape[0], x+n+1)
y_lower, y_upper = np.maximum(0, y-n), np.minimum(im.shape[1], y+n+1)
plt.imshow(im[x_lower:x_upper, y_lower:y_upper],cmap='gray')
plt.show()
```

[ ]:
```
# Lucas-Kanade solution
Vx_col = Vx_gaussian[x_lower:x_upper, y_lower:y_upper, t].reshape(-1,1)
Vy_col = Vy_gaussian[x_lower:x_upper, y_lower:y_upper, t].reshape(-1,1)
A = np.hstack([Vx_col, Vy_col])
b = -Vt_gaussian[x_lower:x_upper, y_lower:y_upper, t].reshape(-1,1)
dp = np.linalg.lstsq(A, b, rcond=None)[0]
# Show the optic flow in this pixel
plt.imshow(im[x_lower:x_upper, y_lower:y_upper], cmap='gray')
plt.quiver(x-x_lower,y-y_lower,dp[0, 0],dp[1, 0], color='r', scale=1, width=.03)
plt.show()
```

6 Problem 3.2 - Apply densely to full volume:

[ ]:
```
# Apply Lucas-Kanade solution to one selected frame
## choose kernel size N, step size(every n'th pixel is visited), and a frame in␣
 ↪time 1.
N = 5
n = int((N-1)//2)
step_size = 1
t = 1

# Apply de
im = ims[t]
dps_image = np.empty((2, V.shape[0], V.shape[1]))
for x in range(0,V.shape[0],step_size):
    for y in range(0,V.shape[1],step_size):
        x_lower, x_upper = np.maximum(0, x-n), np.minimum(im.shape[0], x+n+1)
        y_lower, y_upper = np.maximum(0, y-n), np.minimum(im.shape[1], y+n+1)
        Vx_col = Vx_gaussian[x_lower:x_upper, y_lower:y_upper, t].reshape(-1,1)
        Vy_col = Vy_gaussian[x_lower:x_upper, y_lower:y_upper, t].reshape(-1,1)
        A = np.hstack([Vx_col, Vy_col])
        b = -Vt_gaussian[x_lower:x_upper, y_lower:y_upper, t].reshape(-1,1)
        dp = np.linalg.lstsq(A, b, rcond=None)[0]
        dps_image[:,x,y] = dp[:,0]
plt.imshow(im, cmap='gray')
X, Y = np.meshgrid(np.arange(0, im.shape[0], step_size), np.arange(0, im.
 ↪shape[1], step_size))
plt.quiver(Y, X, dps_image[1,X,Y], dps_image[0,X,Y], scale=500, width=.003)
```

[ ]:
```
# Apply Lucas-Kanade solution to all frames
string = input("It might take a long time to run this part... write \"y\" if␣
 ↪you insist:")
if string == "y":
    ## choose kernel size
    N = 5
    n = int((N-1)//2)
```

```
    step_size = 1
    t = 1

    # initialize the arrow array
    dps_images = np.empty((2, V.shape[0], V.shape[1], V.shape[2]))

    # loop over all pixels in all frames
    for t in range(V.shape[2]):
        im = ims[t]
        dps_image = np.empty((2, V.shape[0], V.shape[1]))
        for x in range(0,V.shape[0],step_size):
            for y in range(0,V.shape[1],step_size):
                # Lucas-Kanade solution
                x_lower, x_upper = np.maximum(0, x-n), np.minimum(im.shape[0],␣
 ↪x+n+1)
                y_lower, y_upper = np.maximum(0, y-n), np.minimum(im.shape[1],␣
 ↪y+n+1)
                Vx_col = Vx_gaussian[x_lower:x_upper, y_lower:y_upper, t].
 ↪reshape(-1,1)
                Vy_col = Vy_gaussian[x_lower:x_upper, y_lower:y_upper, t].
 ↪reshape(-1,1)
                A = np.hstack([Vx_col, Vy_col])
                b = -Vt_gaussian[x_lower:x_upper, y_lower:y_upper, t].
 ↪reshape(-1,1)
                dp = np.linalg.lstsq(A, b, rcond=None)[0]
                dps_image[:,x,y] = dp[:,0]
        dps_images[:,:,:,t] = dps_image
        print(t, end=' ')

    # A large numpy array will be saved to avoid running the code again
    np.save("toy.npy", dps_images)
```

[ ]:
```
# plot without filtering out noises
dps_images = np.load("toy.npy")
X, Y = np.meshgrid(np.arange(0, im.shape[0], step_size), np.arange(0, im.
 ↪shape[1], step_size))

# animation of images in an qt window
%matplotlib qt
for t in range(V.shape[2]):
    plt.imshow(V[:,:,t], cmap='gray')
    plt.quiver(Y, X, dps_images[1,X,Y,t], dps_images[0,X,Y,t], color='r',␣
 ↪scale=500, width=.003)
    plt.pause(0.1)
    plt.clf()
%matplotlib inline
```

[ ]:
```
# Filter out noises by setting a threshold to the magnitude of the arrows

# Prepare the array
dps_images = np.load("toy.npy")
dps_images_filtered = dps_images.copy()

# loop over all pixels in all frames
for x in range(0,V.shape[0],step_size):
  for y in range(0,V.shape[1],step_size):
        for t in range(V.shape[2]):
            ## filter out arrows with small magnitude (<= 5)
            if np.linalg.norm(dps_images_filtered[:,x,y,t]) <= 5:
                dps_images_filtered[:,x,y,t] = np.zeros(2)

# A large numpy array will be saved to avoid running the code again
np.save("toy_filtered.npy", dps_images_filtered)
```

[ ]:
```
# plot images with filtered arrows
dps_images_filtered = np.load("toy_filtered.npy")
step_size = 2
X, Y = np.meshgrid(np.arange(0, V.shape[0], step_size), np.arange(0, V.
 ↪shape[1], step_size))

# animation of images in an qt window
%matplotlib qt
for t in range(V.shape[2]):
    plt.imshow(ims_with_color[t])
    plt.quiver(Y, X, dps_images_filtered[1,X,Y,t],␣
 ↪dps_images_filtered[0,X,Y,t], scale=500, width=.003)
    plt.pause(0.1)
    plt.clf()
%matplotlib inline
```

7 Appendix C: code for applying optical flow analysis on other videos.

7.1 Convert a new video to image sequence:

[ ]:
```
import av
container = av.open(r"C:\Users\ivanf\OneDrive\VID_20240206_222634.mp4")

for frame in container.decode(video=0):
    frame.to_image().save('videoframe-%04d.png' % frame.index)
```

8 Optical flow analysis

(As applying optical flow analysis on the "toilet paper" images is somewhat the same as "hand" images, we will only attach code for "toilet paper" here.)

The following code is applied to our new images sequences. It consists of "flow1.py" and "flow2.py". The first one is used to load the picture and apply first derivative gaussian filter on it and the second one is used to filter optical flow and display it:

8.1 flow1.py:

[ ]:
```
# Please run flow1.py then run flow2.py

import glob
import skimage
import matplotlib.pyplot as plt
import numpy as np
import scipy

path = r".\toilet-paper"
pngs = glob.glob(path + "/*.png")

# make image list
ims_with_color = []
ims = []
print("Reading images...")
for status, i in enumerate(pngs):
    # read in image in grayscale
    ims_with_color.append(plt.imread(i))
    # read in image in grayscale
    ims.append(skimage.color.rgb2gray(plt.imread(i)))
    print(f"Progress: {(status+1)/len(pngs)*100:.2f}%", end="\r")

# make image array
V = np.dstack(ims)
print(f"Reading images done! {len(ims)} images read.")

# Gaussian filter
print("Creating Gaussian filter...")
sigma = 1
gaussian_V = scipy.ndimage.gaussian_filter(V, sigma, order=1)
Vx_gaussian = scipy.ndimage.gaussian_filter(V, sigma, order=1, axes=1)
Vy_gaussian = scipy.ndimage.gaussian_filter(V, sigma, order=1, axes=0)
Vt_gaussian = scipy.ndimage.gaussian_filter(V, sigma, order=1, axes=2)
print("Gaussian filter done!")

print("Calculating optical flow...")
N = 5
```

```
n = int((N-1)//2)
step_size = 3
t = 1
print("\tPreallocating memory...")
dps_images = np.empty((2, V.shape[0], V.shape[1], V.shape[2]))
print("\tPreallocating memory done!")
total_volume = V.shape[0]*V.shape[1]*V.shape[2]
for t in range(V.shape[2]):
    im = ims[t]
    dps_image = np.empty((2, V.shape[0], V.shape[1]))
    for x in range(0,V.shape[0],step_size):
        for y in range(0,V.shape[1],step_size):
            x_lower, x_upper = np.maximum(0, x-n), np.minimum(im.shape[0],␣
 ↪x+n+1)
            y_lower, y_upper = np.maximum(0, y-n), np.minimum(im.shape[1],␣
 ↪y+n+1)
            Vx_col = Vx_gaussian[x_lower:x_upper, y_lower:y_upper, t].
 ↪reshape(-1,1)
            Vy_col = Vy_gaussian[x_lower:x_upper, y_lower:y_upper, t].
 ↪reshape(-1,1)
            A = np.hstack([Vx_col, Vy_col])
            b = -Vt_gaussian[x_lower:x_upper, y_lower:y_upper, t].reshape(-1,1)
            dp = np.linalg.lstsq(A, b, rcond=None)[0]
            dps_image[:,x,y] = dp[:,0]
            print(f"Progress: {(t*V.shape[0]*V.shape[1] + x*V.shape[1] + y)/
 ↪total_volume*100:.2f}%", end="\r")
    dps_images[:,:,:,t] = dps_image
np.save("flow.npy", dps_images)
print("All Done! Program terminates.")
```

8.2 flow2.py:

[ ]:
```
# Please run flow1.py then run flow2.py

import glob
import skimage
import matplotlib.pyplot as plt
import numpy as np

step_size = 5

path = r".\toilet-paper"
pngs = glob.glob(path + "/*.png")

# make image list
ims_with_color = []
```

```
ims = []
print("Reading images...")
for status, i in enumerate(pngs):
    # read in image in grayscale
    ims_with_color.append(plt.imread(i))
    # read in image in grayscale
    ims.append(skimage.color.rgb2gray(plt.imread(i)))
    print(f"Progress: {(status+1)/len(pngs)*100:.2f}%", end="\r")

# make image array
V = np.dstack(ims)
print(f"Reading images done! {len(ims)} images read.")

print("Filtering out noises...")
# plot with filtering out noises
dps_images = np.load("flow.npy")
dps_images_filtered = dps_images.copy()
for x in range(0,V.shape[0],step_size):
  for y in range(0,V.shape[1],step_size):
        for t in range(V.shape[2]):
            ## filter out arrows with small magnitude (<= 5)
            if np.linalg.norm(dps_images_filtered[:,x,y,t]) <= 40 or np.linalg.
 ↪norm(dps_images_filtered[:,x,y,t]) >= 60:
                dps_images_filtered[:,x,y,t] = np.zeros(2)
np.save("flow_filtered.npy", dps_images_filtered)
print("Filtering out noises done!")

print("Plotting...")
dps_images_filtered = np.load("flow_filtered.npy")
X, Y = np.meshgrid(np.arange(0, V.shape[0], step_size), np.arange(0, V.
 ↪shape[1], step_size))
for t in range(V.shape[2]):
    plt.imshow(ims_with_color[t])
    plt.quiver(Y, X, dps_images_filtered[1,X,Y,t],␣
 ↪dps_images_filtered[0,X,Y,t], scale=700, width=.003)
    #plt.pause(1)
    plt.savefig('./toilet_paper_result/toilet_paper_with_quivers_%04d.png' % t)
    plt.clf()
print("Plotting done!")
```