CS 490/548: Assignment 02

Programming Assignments (95%)

Your program MUST be written in **Python**.

Your code file should be named **A02.py**. The goal of this assignment is to compute optical flow.

Graduate students will also have to implement the Lucas-Kanade version of optical flow.

This assignment assumes that A01 is working properly, as the test program calls functions from there.

A02.py should contain the following enumeration:

```
from enum import Enum
class OPTICAL_FLOW(Enum):
   HORN_SHUNCK = "horn_shunck"
   LUCAS_KANADE = "lucas_kanade"
```

A02.py should also contain the following functions:

- **def compute_video_derivatives(video_frames, size)**
  - If size is 2, use the following filters:
    - $kfx = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$
    - $kfy = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$
    - $kft1 = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$
    - $kft2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
  - If size is 3, use the following filters:
    - $kfx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
    - $kfy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
    - $kft1 = \begin{bmatrix} -1 & -2 & -1 \\ -2 & -4 & -2 \\ -1 & -2 & -1 \end{bmatrix}$
    - $kft2 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$
  - Otherwise, return None

- For each pair of frames:
    - Convert the image to a grayscale, float64 image with range [0,1]
    - If the previous frame is not set, set it to the current frame (repeating the first frame)
    - Apply the filters (you may use cv2.filter2D) to get the fx, fy, and ft values across both frames.
    - Scale the results:
        - If size = 2, divide fx, fy, and ft by 4.0
        - If size = 3, divide fx and fy by 8.0 and ft by 16.0
    - Set previous frame to current frame
- Return three lists: all_fx, all_fy, all_ft

- **def compute_one_optical_flow_horn_shunck(fx, fy, ft, max_iter, max_error, weight=1.0)**
    - Compute the optical flow using the Horn-Shunck method.
    - You may use the code we did in class as a baseline. HOWEVER:
        - You will need to compute the error using a variation of the formula from slide 78 of the optical flow slides. After you have computed u and v for an iteration:
            - Compute the brightness constancy constraint part of the error:
                - $berror = \left(f_x u + f_y v + f_t\right)^2$
            - You will need to compute the partial derivatives of u and v with respect to x and y. Use the following filters to compute this:
                - $kfx = \begin{bmatrix} -1 & 1 \end{bmatrix}$
                - $kfy = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$
            - Then, compute the smoothness constraint part of the error:
                - $serror = weight * \left(u_x^2 + u_y^2 + v_x^2 + v_y^2\right)$
            - Add these two error terms together and take the AVERAGE value:
                - error = np.mean(berror + serror)
        - At the end of the loop, break under the following conditions:
            - If the number of iterations is greater than OR equal to max_iter
            - If the error is less than OR equal to max_error
    - Make a 3-channel image with the first two channels as u and v and the 3rd channel as zeroes (as in the exercise).
    - Return the compute flow, the final error, and the number of iterations
- ***CS 548 only*: def compute_one_optical_flow_lucas_kanade(fx, fy, ft, win_size):**
    - Compute the optical flow using the Lucas-Kanade method on each NON-overlapping block of the image.
        - Each block will be (win_size, win_size) in size (except perhaps at the edges, in which case just use whatever the slice gives you).
        - Compute the optical flow using the **UPDATED** formulas on slide 102.
            - If the denominator is less than 1e-6, leave the u and v values at zero.

            - Write the optical flow values to the same size block in the output (in other words, all pixels in the block will have the same optical flow).
    - As with Horn-Shunck, append an extra channel of zeros so the result is a 3-channel image.
    - Return the resulting optical flow.
- **def compute_optical_flow(video_frames, method=OPTICAL_FLOW.HORN_SHUNCK, max_iter=10, max_error=1e-4, horn_weight=1.0, kanade_win_size=19):**
    - If the method is HORN_SHUNCK, use a derivative window size of 2.
    - ***CS 548 only***: If the method is LUCAS_KANADE, use a derivative window size of 3.
    - Compute the derivatives for the video frames.
    - Compute the requested optical flow for each frame.
    - Return the list of optical flow images.

There is no required main function for A02. However, for debugging purposes, I would recommend the following:

```
def main():
    # Load video frames
    video_filepath = "assign02/input/simple/image_%07d.png"
    #video_filepath = "assign02/input/noice/image_%07d.png"
    video_frames = A01.load_video_as_frames(video_filepath)

    # Check if data is invalid
    if video_frames is None:
        print("ERROR: Could not open or find the video!")
        exit(1)

    # OPTIONAL: Only grab the first five frames
    video_frames = video_frames[0:5]

    # Calculate optical flow
    flow_frames = compute_optical_flow(video_frames, method=OPTICAL_FLOW.HORN_SHUNCK)

    # While not closed...
    key = -1
    ESC_KEY = 27
    SPACE_KEY = 32
    index = 0

    while key != ESC_KEY:
        # Get the current image and flow image
        image = video_frames[index]
        flow = flow_frames[index]
```

```
        flow = np.absolute(flow)

        # Show the images
        cv2.imshow("ORIGINAL", image)
        cv2.imshow("FLOW", flow)

        # Wait 30 milliseconds, and grab any key presses
        key = cv2.waitKey(30)

        # If space, move forward
        if key == SPACE_KEY:
            index += 1
            if index >= len(video_frames):
                index = 0

    # Destroy the windows
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

Testing Screenshot (5%)

I have provided several files for testing:

- Test_A02.py – the test program for CS 490
- Test_A02_Grad.py – the test program for CS 548
- General_Testing.py – basic testing functionality
- assign02/
    - input/
        - Input videos for testing

Copy these files/folders into the SAME directory as your python program.

Due to the size of the ground truth data, you will have to separately download it here: ground.zip

Then, unpack the zip file in your assign02 folder such that you have:

- assign02/
    - ground/
        - Individual folders per video

You can either run the testing programs directly OR you can use the testing section of Visual Code.

You MUST run the tests and send a screenshot of the test results! Even if your program(s) do not pass all the tests, you MUST send this screenshot!

Graduate students (CS 548) MUST run Test_A02_Grad! (Please note also that you do need the Test_A02 file as well, since Test_A02_Grad relies on it.)

You may have to do "Command Palette" → "Python: Configure Tests" → unitttest → root directory → test_*py

This screenshot should show clearly the final result of the test run on the command line ("OK" for all passing, "FAILED (failures=N)" for some or all failing) -OR- the testing view in Visual Code.

Grading

Your OVERALL assignment grade is weighted as follows:

- 5% - Testing results screenshot
- 95% - Programming assignments

I reserve the right to take points off for not meeting the specifications in this assignment description. In general, these are things that will be penalized:

- **Code that is not syntactically correct (up to 60 points off!)**
- Sloppy or poor coding style
- Bad coding design principles
- Code that crashes, does not run, or takes a VERY long time to complete
- Using code from ANY source other than the course materials
- Collaboration on code of ANY kind; this is an INDIVIDUAL PROJECT
- Sharing code with other people in this class or using code from this or any other related class
- Output that is incorrect
- Algorithms/implementations that are incorrect
- Submitting improper files
- Failing to submit ALL required files