COMP3065 Computer Vison Coursework

(A Multi-Functional Optical Flow Video Player)

Name: Jingjin LI

Student ID: 20127165

Email: scyjl3@nottingham.edu.cn

May. 6th, 2022, submitted

University of Nottingham Ningbo China, School of Computer Science

Descriptions

This section will mainly describe the functionalities of the finished coursework. In this coursework, a multi-functional optical flow video player was implemented efficiently, and various hyperparameters and different methods were compared with their own-implemented one and analyzed. All completed features will be shown below:

- **Iterative Lucas-Kanade optical flow with image pyramids** is implemented; see in opticalFlow.py file and implementations section for details. Users can set hyperparameters through the command line. (Basic Lucas-Kanade optical flow with two more refinements)
- Use own-implemented optical flow to track the good feature points in every frame; the length of the track line and the interval for re-detect the feature points can be changed by the user. Two directions of the optical flow calculations are performed to eliminate some wrong flow results.
- A user interface is completed to view various related information, and detailed functions are:
    - Support .mp4 and .avi files, users can freely choose their test videos from the system explorer (the coursework will also provide 6 test videos taken by the author).
    - The video can be played or paused by pressing one button (act as a media player).
    - The user can choose whether to view the track lines or not.
    - Several necessary parameters can be viewed at the top, including the length of track lines, the frame interval for detecting good features, the current number of the feature points, and the current frame.

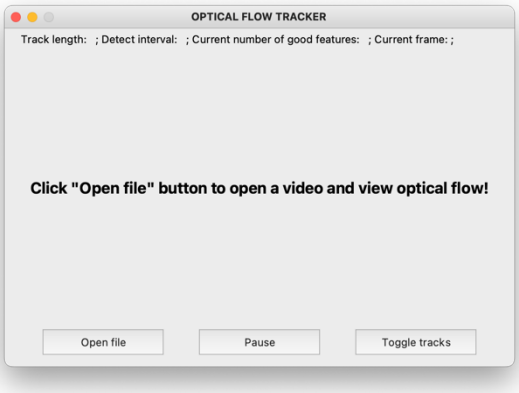
Generally, this coursework achieves the basic optical flow task and adds additional features related to algorithm and appearance. Some experiments are also performed to analyze the algorithm.

Implementations

This section is used to introduce key implementation and user instructions. We will firstly state how to run the code, and then show the pseudo-code of the implemented Lucas-Kanade algorithm. Finally, the code logic of the left features will be explained in detail.

How to run the code (some requirements and develop tools are shown at the end of the report): enter the program folder, execute the main_ui.py file, then the user interface will be pop-up for user to choose:

```
python main_ui.py
```



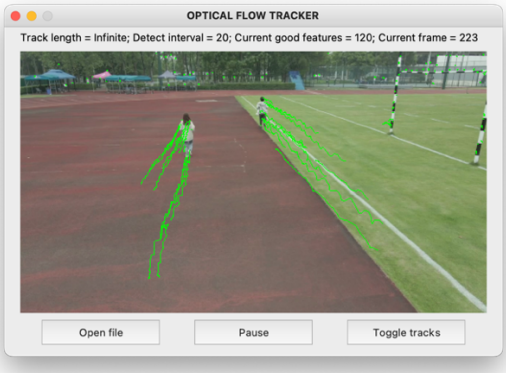
An easy UI for user to interact and view the optical flow from selected videos, user only needs to press the button and various information are also shown in the top bar.

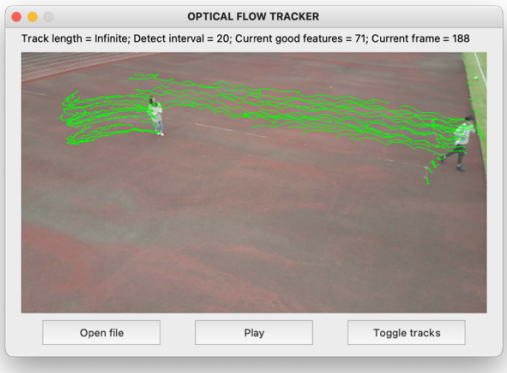
The user can also tune the parameters by using command line arguments, following command can help the user to view all arguments and their explanations:

```
python main_ui.py -h
```

Users can set the length of the track line drawn on the video, set the frame interval to re-detect the good feature of the current frame, and set window size, pyramid level and the number of iterations in the optical flow algorithm.

By clicking the "Open file" button, the user can choose test videos from the file explorer. After the video is loaded, it will start, and the track lines produced by optical flow will be automatically drawn. By clicking the "Pause" button, the user can pause the video, and the button will turn to "Play". By clicking it again, the video will continue playing. By clicking "Toggle tracks", the drawn track lines will disappear/appear. Here are some snapshots (with track lines) of the program:





The video file can be loaded by covering. By clicking up-leftmost exit, user can simply exit program.

Pseudo-code of own-implemented Lucas-Kanade algorithm:

Algorithm Lucas-Kanade:
INPUT: two consecutive frame images (I_1, I_2), feature points (F), windows size (w), pyramid levels (l), iterations (i), residue (r).
OUTPUT: feature points after flowing (P).

```
Get two lists, each list contains I_1, I_2 in all pyramid levels
Initialize an array R to save the results for each point in F
FOR each pyramid level (from small image to large image)
    Get I_1, I_2 from corresponding pyramid lists
    Transform feature points F corresponding to the pyramid level
    Calculate gradient in image horizontal and vertical direction
    FOR each feature point
        Calculate matrix M_left in the range of w          # See Equation (3)
        REAPT
            Calculate gradient in time direction
            Calculate matrix M_right in the range of w     # See Equation (4)
            Calculate the optical flow d, and accumulate the result   # See Equation (5)
            IF ||d||^2 < r THEN
                break this iteration to speed up
            ENDIF
        UNTIL i times done
        In the current pyramid level, add the accumulated iterative optical flow to R
    ENDFOR
ENDFOR
P = F + R
RETURN P
```

Generally, Lucas Kanade algorithm is to find $\underset{d}{\operatorname{argmin}} ||Ad - b||^2$, using least square method we can get:

$$AA^T \times d = A^T b \tag{1}$$

Therefore,

$$d = (A^T A)^{-1} A^T b \tag{2}$$

In equation (2), $M_{left} = AA^T, M_{right} = A^T b$, so we can get equation (3) (4), then we obtain equation (5)

The pseudo-code snippet is core algorithm part in the coursework, the equations to calculate the related matrix and optical flow (see lines with green comments) are shown below:

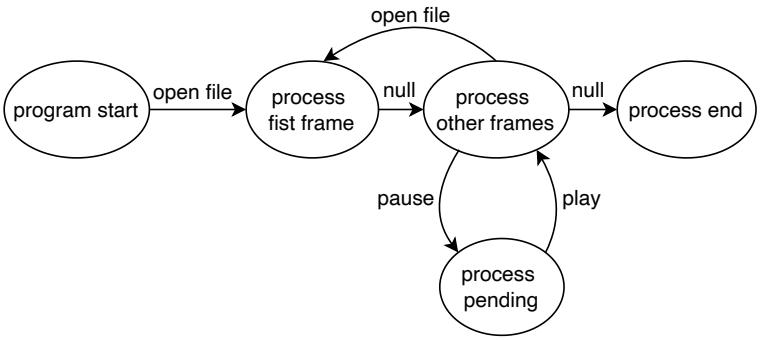
$$M_{left} = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \tag{3}$$

$$M_{right} = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \tag{4}$$

$$d = M_{left}^{-1} \times M_{right} \tag{5}$$

Using equation (5), we can simply obtain the two-dimensional displacement of the points, and the image pyramid can let this algorithm detect relatively rapid object movement.

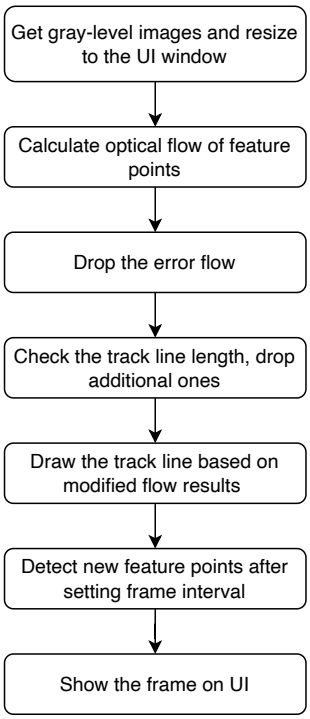
Logic of the whole project:



The above simple state diagram illustrates the control flow for user to use the program, by clicking different buttons, the state will be transited. Users can choose whether to view the track lines drawn by optical flow algorithm during video frame processing.

Flow chart of processing frame:

The following chart shows the basic process of tracking trajectories according to our implemented optical flow; other features are also embedded in this chart. Additionally, for features like showing various parameters, play/pause and importing a file, please see the UI structure in the main_ui.py file.



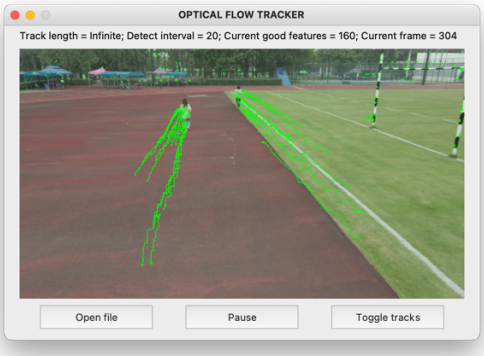
For the method to drop the error flow, we calculate the inverse direction of the two images, and check the difference between feature points and inverse flow points, if the difference in either x or y direction larger than a threshold we set, this flow points should be considered as error.

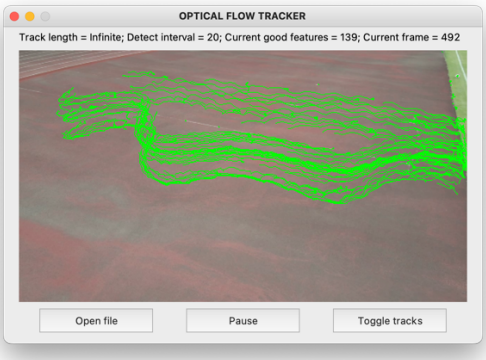
Every n frame, the feature points will be updated.

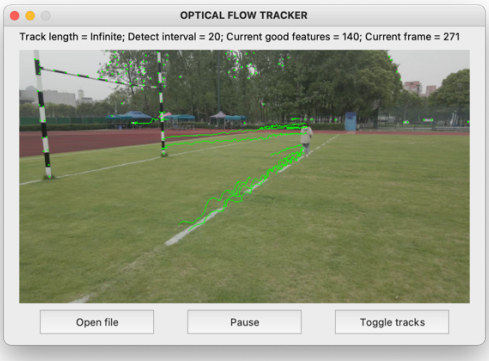
Notice: for the good feature points detection, we simply use Shi-Tomasi corner detection, which is cv2.goodFeaturesToTrack.

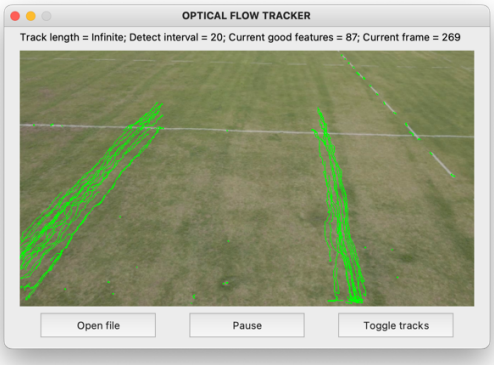
Results

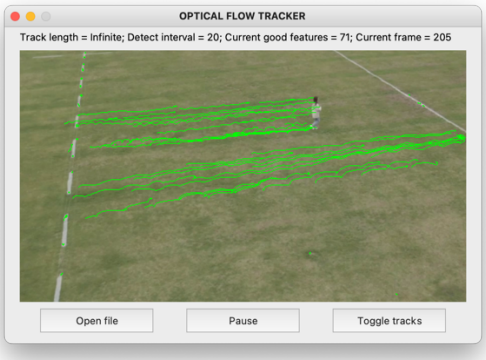
Here are the final track lines of 6 own-captured test videos (the image is near the last frame):

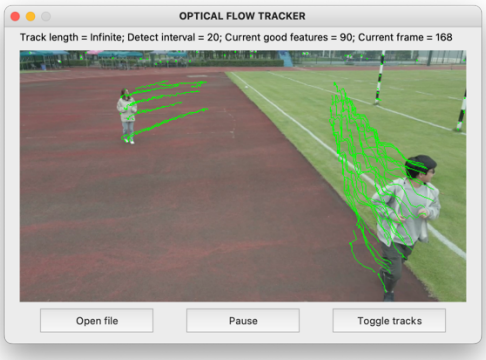






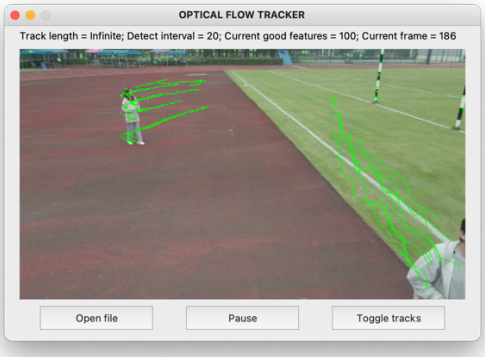


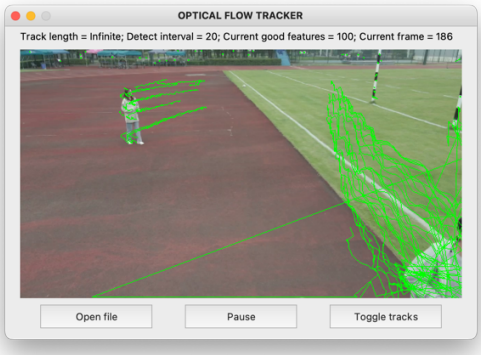




The above results show optical flow tracking for moving objects in 6 testing videos. We tracked two people moving on the grass ground or the playground in different views in these videos. Each figure could successfully track the moving object, showing reasonable flow results.

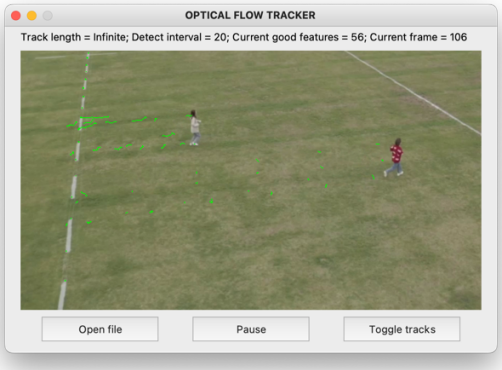
Reason for checking flow displacement error: below two figures show the optical flow tracking with or without checking flow displacement errors at the same frame of the video:

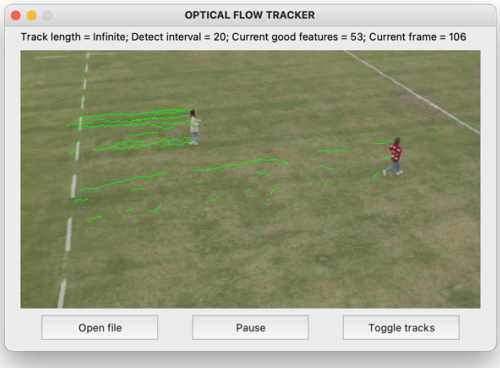


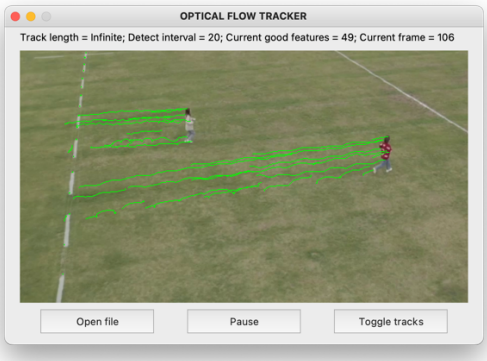


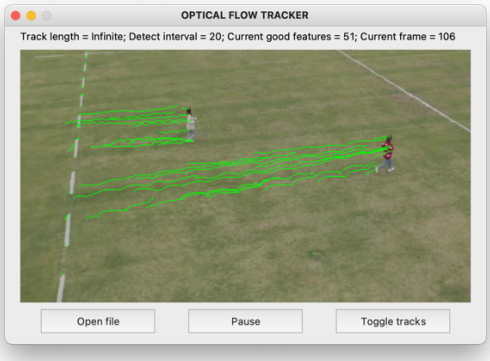
In the left figure, the algorithm checks and removes some error flow results with extremely large values. Therefore, the track line is proper and stable. However, in the right figure, the algorithm without error flows checking leads to some messy lines, which are not stable and produce some wrong lines.

Reason for implementing pyramid level: below two figures illustrate the optical flow tracking with different pyramid level (1-4) at the same frame of the video:



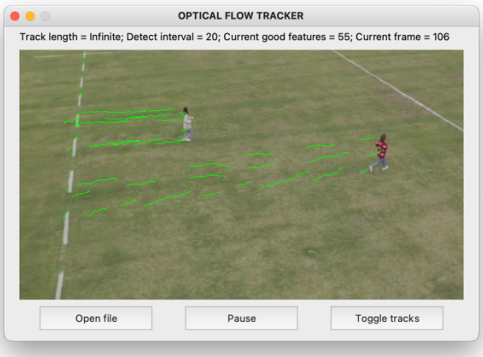


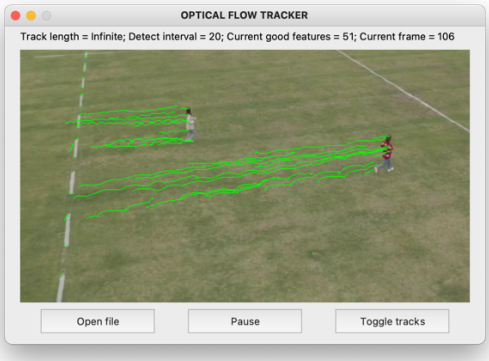




The above 4 figures show the tracking results using pyramid level 1, 2, 3, 4. For the figure with 1 pyramid level, which means the optical flow does not implement a down-sample pyramid methodology, the algorithm cannot track the moving people. When the pyramid level increases to 2, it can track relatively slow-moving objects, but it is still difficult to track the quick-moving object. Finally, increasing the pyramid level (3 and 4) can track both objects in the testing video. Therefore, using this multi-resolution method, we can track the relatively large motion of the objects.

Other parameters could influence the optical flow performance: we found that other related parameters can also influence the tracking results. One is the window size in Lucas Kanade algorithm; this window concept assumes all pixels in this window have the same speed. The default value of the window size is 5, and it should not be set too small due to the aperture problem. Besides, the corner feature point detection could determine the moving direction of the object better. More iterations lead to accurate results for the iteration times to calculate the optical flow. Because the iterative operation makes the new image warp back to the old image as near as possible, and add the accumulated d together, which achieve the refinements of the original calculated d. The default iteration time is 10. Following figures show the results with 2 iterations (left one) and 10 iterations (right one):





Hyperparameters explanations (summary):

Iterations	The degree of the optical flow calculation refinement, large value means more accurate optical flow for two consecutive frames.
Pyramid level	Used for multi-resolution optical flow, large value means it can track object with large motion.
Detect interval	Time for resampling the feature points, large value means less sampling times, which may cause tracking lost problem.

Discussions

This section discusses the strength and weaknesses of the chosen approach and method and makes a final summary of this coursework.

For the strengths of this project, we established a basic UI to view several parameters and the status of the video, which is helpful for experiments with optical flow tracking. We can also load the video conveniently and pause the video to view a specific frame. Additionally, we implement error flow checking to get better tracking results. We add the multi-resolution function to reduce the large motion detection problem, and we also refine the optical flow calculation through an iterative method. Moreover, we performed several experiments and investigated to set relatively good initial default parameters for optical flow and corner detector. All processes in the UI are smooth and without any errors.

For the weaknesses of the coursework, we use Python to implement all these functions and features. Although we use sparse optical flow to reduce the number of pixels to process, there are still many loops for calculating the optical flow, which is relatively slow in processing videos with the increase of the feature points. The built-in function cv2.calcOpticalFlowPyrLK is implemented using C++, so it can process video in real-time. Our own-implemented optical flow calculation may not process high-quality videos in real-time. The FPS (frame per second) will be low. Some acceleration methods can be considered in future (GPU, C/C++), and the UI could be more functional for tuning parameters during the process.

Appendix

Video Source: videos are captured by DJI drone, the content is different people moving on the UNNC playground, origin videos are 4K/30FPS, we use some methods to compress and resize to the UI window.

How to download videos?

- https://pan.baidu.com/s/1R9sD5OJ0nlK5Z_Bcrml4nw Code**:** 0qg9
- https://drive.google.com/drive/folders/1dLuHLuIaw_6xkr4vc1_-waqMKobE7r5k?usp=sharing

Develop Environments:

- Python 3.8.4
- PyCharm 2021.3.1 (Professional Edition)

Python Requirements:

- opencv-python 4.5.5.64
- PyQt5 5.15.4
- numpy 1.22.1