VIDEO_FLAWER: REALISTIC VIDEO DISTURBER FOR ARTIFICIAL VIDEO

***Yang Hong-Bin* [1], *Matous Cejnek* [2]**

[1] ***FEL, CTU, Technická 2, Praha 6, Czechia, petingo.yang@gmail.com***
[2] ***DICE, FME, CTU, Technicka 4, Praha 6, Czechia***

Abstrakt:
Klíčová slova:

Abstract: Although people try hard to minimize defects in a video most of the time, sometimes it is needed to augment or simulate them for some reasons. We created a program which can add realistic effects to videos, including camera shake, auto focus blur, and noise simulation.
Keywords: noise, synthetic data, digital video processing

1 Introduction

More or less, videos always come with defects. In most cases, people try their best to eliminate them or minimize the affect. Camera and lens manufacturers develop optical image stabilizer to reduce the vibration, continuously updating their noise reduction algorithm in order to attract consumers. Film makers use tripods to shot steady videos, adjusting the parameters in order to decrease the ISO sensitivity. However, in some cases, it is desired to add those defect to the video. For example, if a director wants to make a filmed scene more vivid, adding camera shake effect on it may make audiences feel like they are in the scene personally. Another common example is old video simulation where different kinds of noise is added. Or, as the title of this paper, it it possible to disturb an video generated by computer graphic technology, making it more realistic. This disturber may have further application, which is generating training data for deep learning-based stabilization or denoising method.

In this paper is proposed new python package - *video_flawer* - to create aritificial noise-like effects in video records. Three major effects is implemented: camera shake, auto-focus blur, and noise simulation. The program is done by Python with the help of OpenCV and Numpy.

2 Effects

2.1 Camera Shake

The first effect is camera shake. This effect appears in almost every video which is shot by a hand-held camera due to the inevitable minor movement of human arms. That is to say, given a steady video (perhaps a tripod shot), our program can convert it into one with artificial vibrations as if it is recorded by a hand-held camera. To accomplish this, the movement is first broken down into four parts: left-right (x-axis), up-down (y-axis), front-back (z-axis), and rotation (r). Moreover, movement in each axis can be further split into rhythmic movement and random movement. By simulating them separately as a 1-dimensional action, a smooth and realistic camera shake effect is created.

For the four axes, identical formulas with different parameters are used to determine the quantity of the simulated movement. Commencing with rhythmic movement, the following equation is used:

$$qh_i = a \cdot \sin(f * i) \tag{1}$$

where qh_i denotes the quantity of rhythmic movement of frame number i, while a and f controls amplitude and frequency respectively. In implementation, a and f are variables which can be altered by users based on their preferences. On the other hand, an algorithm which composes a wave with inconstant amplitude and frequency is developed to simulate the quantity of random movement. This algorithm takes five parameters as input, including total frame amount, minimum value of amplitude, maximum value of amplitude, minimum value of wavelength, and maximum value of wavelength. The program would generate waves one by one randomly based on the parameters and concatenate them altogether until every frame is assigned a value qr_i.

With qh_i and qr_i, total movement quantity q_i can be obtained by simply summing them up as:

$$q_i = qh_i + qr_i \tag{2}$$

For each axis, a unique sequence q is generated, and a corresponding affine transformation matrix is then built to express either translation, scale operation, or rotation. In the following equations, $M_{(a,i)}$ and $q_{(a,i)}$ denotes the transformation matrix and movement quantity of axis a, frame number i respectively. For x- and y-axis, the following matrixes are used to express translation:

$$M_{(x,i)} = \begin{bmatrix} 1 & 0 & q_{(x,i)} \\ 0 & 1 & 0 \end{bmatrix} \tag{3}$$

$$M_{(y,i)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & q_{(y,i)} \end{bmatrix} \tag{4}$$

For z-axis, scaling is used to simulate the front-back movement, and the following matrix is used to express the operation:

$$M_{(z,i)} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \end{bmatrix} \tag{5}$$

where s denotes the scaling factor

$$s = (100 + q_{(z,i)})/100 \tag{6}$$

For rotation, the following matrix is calculated in order to rotate the frame by $q_{(r,i)}$ degrees around the center point:

$$M_{(r,i)} = \begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot c_w - \beta \cdot c_h \\ -\beta & \alpha & \beta \cdot c_w - (1-\alpha)\cdot c_h \end{bmatrix} \tag{7}$$

where

$$\alpha = \cos(q_{(r,i)}) \tag{8}$$

$$\beta = \sin(q_{(r,i)}) \tag{9}$$

$$c_w = (\text{frame width})/2 \tag{10}$$

$$c_h = (\text{frame height})/2 \tag{11}$$

After constructing the four matrixes, they are merged to M_i so that only 1 affine transformation is required to performed instead of 4.

$$M_i = (M_{(x,i)} + M_{(y,i)} + M_{(r,i)} - 2I)\cdot M_{(z,i)} \tag{12}$$

After applying affine transformations on all the frames, camera shake would be successfully added to the video.

2.2 Auto Focus Blur

The second effect is lens blur. If a video is shot with auto-focus function, when the lens try to refocus, the focal point would change and cause image blur. In this program, average blur is used to simulate this effect. An average blur is done by doing a convolution between a frame and a $n \times n$ normalized box filter K.

$$K = \frac{1}{n \times n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \tag{13}$$

In other words, the value of every pixel in the blurred image would be the average of its neighborhood in the original image. Therefore, the bigger n is, the blurrier the output image will be.

The intensity change is simulated by the combination of two waves: one wave with larger amplitude and longer wavelength following with another smaller one. The idea is, when a lens tries to refocus, it would adjust the focal length until the target object is the clearest. However, it would know that it has reached the desired point after exceeding a little. In implementation, since the intensity should be an integer to perform average blur, the figure would be rounded.

2.3 Noise

It is common to see noise when watching a video, especially in the past when videos are recorded by films. Noise are caused by many reasons and have different form. In our program, three kinds of noise simulation is implemented, including pixel noise, line noise, and pattern noise.

2.3.1 Pixel noise

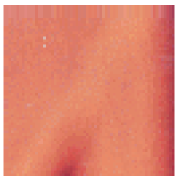
Pixel noise is an inevitable noise, and it is especially obvious if ISO sensitivity is high. This effect makes a frame contain bright spots or randomly-spaced bright pixels. To simulate this effect, two random number is used. The first value controls whether the pixel value should be alter, and the second value controls the quantity of alternation. In order to make it more realistic, the RGB channel are processed separately. The probability and the minimum and maximum of the value can be adjusted by the user.

2.3.2 Line noise

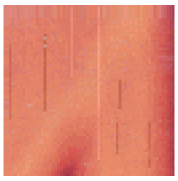
Similar to pixel noise, one random number is used to determine whether a line should be altered. However, in the next step, two random values are required where one controls the length and the other controls the quantity of alternation.

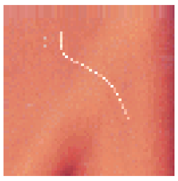
2.3.3 Pattern noise

Pattern noise takes an extra image as input. The image is the reference of the pattern which would be added to the video. It should come with black or transparent background. Otherwise, unnatural edge would appear. The program would randomly scale and rotate the pattern and put it in a randomly selected place.


Original

Pixel Noise


Line Noise


Pattern Noise

3 Running Program

In order to run the program, 2 packages are required: opencv-python [1] and numpy [2].

3.1 Installation

Simplest way to install the *video_flawer* is via pip:

```
pip install video_flawer
```

3.2 Usage as utility

To flaw a video (with path ./test.mp4) with preset defects call:

```
python –m video_flawer test.mp4
```

The output (for example /data/out.avi) file can be setup as:

```
python –m video_flawer test.mp4 /data/out.avi
```

For customization of video defects you should create your custom config file (JSON format). You can provide the file (for example ./my_config.json as follows:

```
python –m video_flawer text.mp4 /data/out.avi my_config.json
```

3.3 Usage in script

You can use video_flawer directly in your script with preset defects as follows

```
import video_flawer

INPUT_PATH = "test.mp4"
OUTPUT_PATH = "out.avi"

video_flawer.run(INPUT_PATH, OUTPUT_PATH)
```

The defect details can be provided as nested dictionary

```
import video_flawer

INPUT_PATH = "test.mp4"
OUTPUT_PATH = "out.avi"

CONFIG = {
    "shift_sin_x": {
        "max_shift": 3,
        "frequency": 0.25
    },
}

video_flawer.run(INPUT_PATH, OUTPUT_PATH, config_data=CONFIG)
```

Or the defects can be described in JSON file:

```
import video_flawer

INPUT_PATH = "test.mp4"
OUTPUT_PATH = "out.avi"

video_flawer.run(INPUT_PATH, OUTPUT_PATH, config_path="config_example.json")
```

4 Conclusion

In this paper the *video_flawer* Python package is introduced. The effects implemented in the library are explained. Usage examples are provided. The project repository (all source codes) are accessible under MIT license at `https://github.com/matousc89/video_flawer`.

Acknowledgement

This work was partially supported by the Grant Agency of the Czech Technical University in Prague, grant No. *SGS18/177/OHK2/3T/12*.

Literature

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

ISBN 978-80-01-06776-5