Multimedia Tools and Applications (2021) 80:14941–14960
https://doi.org/10.1007/s11042-020-10385-8

Realtime flicker removal for fast video streaming and detection of moving objects

Jarosław Nowisz[1] · Michał Kopania[1] · Artur Przelaskowski[1]

Received: 29 April 2020 / Revised: 16 September 2020 / Accepted: 22 December 2020 /
Published online: 30 January 2021

Abstract
High-speed cameras are used in computer vision systems to track balls, shuttlecocks, or players in many different sports. Collected information is used for statistics, as coaches' and players' aids to improve technique and tactics or as referees' aids to verify their decisions or to enrich television broadcasts. Sports arenas in which games are played are often equipped with lights generating flickering effects in captured movies. A fast and yet effective enough algorithm is necessary to remove flickering so movement detection and object tracking algorithms could be used. In this paper, we propose a fast flicker removal algorithm working as an online filter on frame streams at speeds exceeding 200 frames per second. Most of the solutions found in literature concentrate on effectiveness and accuracy and not on the speed of operation. In contrast, our original solution is designed with speed in mind with sufficient accuracy to be used before calculating differential frames to detect movement in streams. Our algorithm is adaptive and works when lighting conditions are changing (new light sources) and performs well with various light sources that are causing flickering. The results of the experiments carried out show the high effectiveness of the method implemented on CPU and GPU, allowing effective tracking of objects of interest in preliminary applications of a commercially offered instant review system for badminton.

Keywords Flickering removal · Video processing · Movement detection · Intensity flicker · Object tracking

✉ Jarosław Nowisz
j.nowisz@gmail.com

Michał Kopania
michal.kopania@mini.pw.edu.pl

Artur Przelaskowski
arturp@mini.pw.edu.pl

[1] Faculty of Mathematics and Information Science, Warsaw University of Technology, 00-661 Warszawa, Poland

1 Introduction

For many years, big sports organizations like the Olympic committee and UEFA have required from tournament organizers to use only non-flickering lighting [22], but such lighting systems are expensive and are not common. Most sports venues where tournaments are organized are equipped with lighting causing flickering effect, thus for most tournaments, a workaround solution that removes flickering must be used. There are different sources of flickering in movies. In this paper, we focus on the flickering effect in high-speed acquisition cameras caused by artificial lighting.

Existing flicker removal solutions concentrate mostly on accuracy but not on speed. On the contrary, we needed a fast solution just accurate enough to properly detect a moving small object as a shuttlecock. A moving shuttlecock is a small object that, depending on the position of the camera in relation to the court, may be represented by as small a number of pixels as around a dozen. The velocity of a shuttlecock varies between 30 km/h to 450 km/h. At higher speeds, pictures of the shuttlecocks are blurred even when using 200fps cameras. The number of features of such shuttlecocks is too small to build a robust tracker, so it is necessary to extract additional features from detected movement and trajectories. The proposed flicker removing algorithm reduces flickering only to the extent that it is sufficient to detect movement and analyze trajectories.

We have built a challenge system for badminton competitions that take place in sports halls where lights are the source of flickering effects in frame streams recorded by high-speed cameras. If a player does not agree with the referee's decision, she or he can challenge it by asking for video verification. The system [7] consists of servers running object movement detection and fast moving shuttlecock tracking algorithms. Images are captured by 14–20 cameras working at 150–200 frames per second.

The algorithm presented in this paper is part of the challenge system. The algorithm runs before shuttlecock detection and tracking algorithms and allows the next ones to work properly. If the flickering was not removed, then the motion detection part of the system would not be able to distinguish flickering from moving objects, and the shuttlecock movement detection would not be possible.

We have made several assumptions:

- mains AC frequency and frame rate are known,
- the main purpose of the algorithm is to distinguish flickering from movement so the differential frame between two adjacent frames should show only the movement,
- the algorithm should work online and cannot use any future frames,
- the scene may have many light sources,
- lighting conditions may change (new light sources, for example),
- cameras are black and white and stationary,
- cameras produce a stream of frames with speeds ranging above 150 frames per second,
- once set, the camera's speed is constant,
- in rare cases, some frames can be dropped due to transmission errors,
- each frame gets a timestamp before leaving a camera,
- flicker remover requires a pixel-wise granularity,
- the algorithm should run as fast as possible leaving hardware computing resources to other parts of the system.

Our algorithm measures fluctuations between frames and then removes these fluctuations. It aims to remove fluctuations in a way that, if there is no movement in a scene, the difference between consecutive frames should be equal to zero, but if there is a movement, then the differential frame should consist of pixels indicating only this movement. The whole challenge system works in real-time and delays in processing should be as small as possible, so any buffering frames to remove flickering in an actual frame based on data from future frames should be avoided.

Two versions of the proposed algorithm were implemented. Both produce the same quality results but have different optimizations. The first one runs only on CPU and uses Threading Building Blocks [8] to parallelize operations on pixels. The second one runs mostly on GPU using OpenCL [17]. We compared their performance and applicability.

2 Related work

There are different sources of flickering effects in movies and researchers come up with different methods to reduce them or remove them completely. Old movies recorded manually by turning the crank have frames with varying luminosity because of varying speed of moving the crank. Old films can also change the brightness due to aging and uneven loss of physical and chemical properties of the material they were recorded on. Description of age-related artifacts may be found in [19].

To remove the first effect that is usually spatially even on a single frame, but unevenly distributed across consecutive frames, global methods [15, 24] can be used. Some kind of global measure is calculated for every frame. Based on differences in values of this measure for consecutive frames, intensities of pixels of every frame are adjusted to reduce differences between frames. In a single frame, every pixel is adjusted by the same value, hence these methods are global. Such a technique can be also successfully used in outdoor time-lapse movies. There are plugins for popular video editing software which can remove flickering, for example Digital Anarchy's Flicker Free Removal Software [5].

Mentioned earlier local flickering can be caused not only by small damages of old films but also by many different sources of flickering lights or different reflection properties of materials in the scene in slow motion movies. It may also be caused for example by turned on a tv set filmed by a camera working with a different frequency than the tv set [1].

The best approach to remove local differences in luminosity is to use some kind of local method. There are two main categories of such methods. Frames can be divided into smaller, adjacent areas, usually rectangular. Each area is treated independently by one of the global versions of the algorithm mentioned above. The second approach is to use a kernel and compare pairs of pixels together with their near neighborhoods between different frames. Usually, a Gaussian kernel is used. Local methods are presented for example in [18, 21].

Local methods work well when the camera is not moving and when there is no movement in the scene or the movement is slow and sporadic. In the case of global movement (moving camera) or local movement (movement in the scene) undesired artifacts called "ghosts" start to appear in places where the scene changed just a moment ago.

The reason that the "ghosts" are produced by the local flicker removal algorithms is the inability to distinguish changes of the luminosity of pixels caused by a movement from changes caused by flickering. To solve this problem, consecutive frames are compared and vectors of shifts of image fragments between these frames are calculated. Optical flow

methods, like Lucas-Kanade [14] or Farneback [6] or feature detection methods like SURF [2], SIFT [13], or ORB [20] can be used. Calculated vectors are used to find corresponding areas of the consecutive frames where there was a movement. Comparing corresponding areas rather than static same areas from different frames greatly reduces undesired artifacts ("ghosts").

The best results are achieved by combining global methods with local methods and movement vectors estimation [10, 23]. There can be some other small improvements, like for example detection of scene changes in a movie and deflickering different scenes independently.

The method presented in [25] is interesting from our perspective because it solves the problem with some similar constraints. In both cases cameras do not move, flickering may be caused by many local sources of artificial lights and the application of the algorithm is necessary to enable proper further processing. An introduction and an overview of classes of flicker removal methods can be found in [9].

Authors of the mentioned, advanced methods focus on the effectiveness of flicker removal, but not on the speed of execution. Our work is concentrated on the speed of execution with still reasonable and applicable results.

3 The flicker problem

The flicker problem in videos has many slightly different definitions depending on the source of the flickering effect. In general, it can be defined as a sudden luminosity change in a video sequence causing the impression of the instability of visual perception. More definitions and comparisons can be found in [9]. Flickering may be caused by:

- sun flicker effect - underwater image acquisition,
- video surveillance - the presence of fluorescent lamps in the footage,
- camcorded videos - scrolling stripes appear because the higher frequency backlight of the screen is sampled at a lower rate by the camcorder,
- old archived movies - brightness change due to different shutter speed and loss of physical and chemical properties of the material they were recorded on,
- video coding and compression - similar regions between two frames are differently encoded,
- time-lapse videos - in natural light during the day brightness changes (clouds, time of the day),
- high-speed videos in artificial light.

We focus on the flicker problem in high-speed acquisition cameras (over 150 fps) caused by artificial lighting in sports halls.

The flickering effect is caused by oscillating current, which, according to the norm IEC 60038, has 50 Hz frequency (in Europe) with accuracy ± 1% for 95% of the time and ± 4% for all the time. Sports halls are usually illuminated by lamps powered by all three phases, so the flickering effect may not be spatially uniform in captured streams.

Figure 1(a) shows how the luminosity of one random pixel is changing. Data was recorded in a sports hall at the speed of 150fps for the duration of 1 s in a static scene without any movement. The sports hall is illuminated with lamps that generate a flickering effect.

Figure 1(b) shows the brightness of two pixels from different parts of an image (pixel 1 - blue circle, pixel 2 - purple diamond) and the absolute difference of brightness of these two pixels (yellow square). The X-axis represents the frame number, Y-axis the luminosity (or luminosity change for yellow square).

In our case, the absolute differences between pairs of spatially close pixels that display the same surfaces usually change values similarly over time. On the other hand, the absolute differences between pairs of pixels displaying different surfaces usually do not show similarities in changes over time. Moreover, comparing differential images of pairs of consecutive frames, like in Fig. 3, shows the wave-like effects, often with many origins, of differences between pixel luminosities changing values spatially in waves. This is the reason that any global methods for flicker removal will not work in this case. Different models of flickering, more in-depth analysis of local and global methods and their comparison can be found in [9].

3.1 Requirements

The requirements for deflickering algorithm solving the described problem are as follows:

- the algorithm should run as fast as possible leaving hardware computing resources to other parts of the system,
- the algorithm should process 500 frames in no more than 1 s for a video with resolution 800x600px on an Intel Core i7 machine (12 threads),

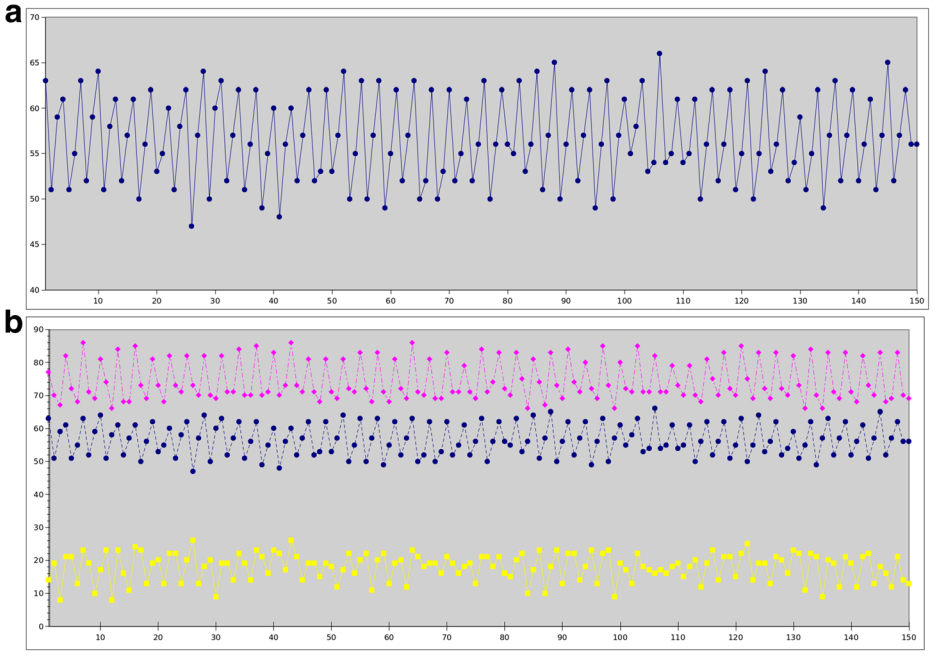

Fig. 1 The brightness of one random pixel (**a**) and two pixels from different parts of an image (**b**): pixel 1 - blue circle, pixel 2 - purple diamond and the absolute difference of brightness of these two pixels (yellow square). The X-axis represents the frame number, Y-axis the luminosity (or luminosity change for yellow square)

- deflickering process should return binary differential images of pairs of consecutive frames with white pixels representing movement (as much as possible true positives) and black pixels representing areas without movement; especially brightness change because of flickering should result in black pixels and not white pixels (as little as possible false positives),
- after a flicker removal process, video footage does not have to be 100% visually correct, because we use our flicker remover as a preprocessor for movement detection and object tracking, we assume that these parts of the scene which are moving may still flicker.

4 Algorithm

Our algorithm processes one-channel, grayscale frames from a stream captured by a camera. The output of the algorithm is a stream of binary differential frames as a result of comparing consecutive frames from the input stream. The algorithm can also output a stream of modified input frames with removed flickering effect, but these frames, have two flaws: parts of the image with moving objects can still flicker and there may appear temporal flickering "ghosts" in places from which objects just moved. It will be described in more detail later. We use differential frames to verify the effectiveness and robustness of the algorithm. As mentioned earlier, this algorithm is part of the bigger computer vision system. In that system differential frames are used in further processing to detect moving objects.

The main steps of the algorithm are presented in Fig. 2.

The most important step is "Update masks" - in this step a matrix that corrects the flickering effect is calculated.

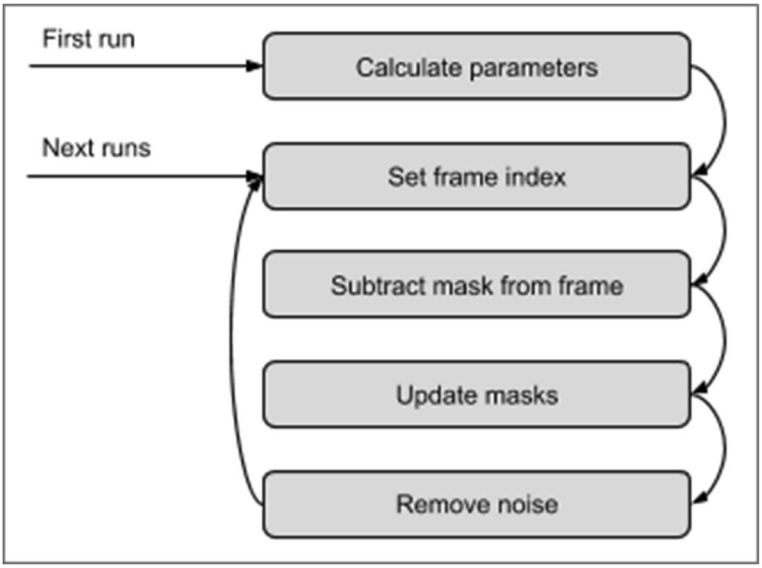

Fig. 2 Main steps of the algorithm

4.1 First step: Calculate parameters

We use the fact that flickering caused by artificial lighting is a phenomenon repeated at a constant frequency. Knowing frames per second captured by the camera (*FPS*) and power line frequency (*Q*) we calculate the number of frames (*N*) after which the pattern of flickering repeats itself. At the beginning of the program, we assume that both Q and FPS are constant and integer. In real life they are not, that is why the algorithm is adaptive and modifies masks in the "Update masks" step.

To calculate *N* we use the below formula (1) where *gcd()* is the greatest common denominator of two numbers.

$$\mathrm{N} = \max\left(\frac{Q}{gcd(Q, FPS)}, \frac{FPS}{gcd(Q, FPS)}\right) \tag{1}$$

The algorithm uses masks to correct the flickering effect. Masks have the same dimensions as frames. At the beginning of the algorithm, all masks are filled with zeros. Every first frame of the block of *N* consecutive frames is treated as a reference frame for the next *N* - 1 frames. For each frame after the first in a block of *N*, the corresponding mask is selected and subtracted from the frame. Therefore we need Z masks, where Z is defined as:

$$Z = N{-}1 \tag{2}$$

4.2 Second step: Set frame index

The use of flicker periodicity in the algorithm requires that no frames should be omitted or lost before being processed by the algorithm. The hardware that was used for tests did not guarantee that every frame captured by the camera was transmitted successfully to the computer running the algorithm. During our tests, some of the frames were lost. Fortunately the camera we used marks every frame that leaves it with a timestamp. Knowing the FPS of the camera, we can calculate the expected next timestamp after receiving the current frame. If the timestamp of the next frame is not equal to the expected timestamp then the count of skipped frames and the current index *ri* of the frame is calculated based on the timestamp of the previously received frame and FPS of the camera. If we are not missing any frame then *ri* is simply the next number.

4.3 Third step: Subtract mask from a frame

If the current frame is the first in the block of *N* consecutive frames, then nothing happens - this frame is not modified and is used as a reference for the next *N* - 1 frames. If the current frame is one of the *N* - 1 next frames, then the corresponding mask is selected and subtracted from the frame. All pixels of all the masks are regularly modified in the "Update masks" step. If *R[m]* is the m-th frame, r_i is the index of the current frame, and *M[k]* is the k-th mask, then the current frame $R[r_i]$ is modified as follows (*mod* is a modulo operator):

$$R[r_i] = R[r_i]{-}M[(r_i \bmod N){-}1], \quad if\,(r_i \bmod N) > 0 \tag{3}$$

4.4 Fourth step: Update masks

Masks have to be adjusted because of two reasons. First, the scene can change, for example, a person that was standing on the left side of the scene now can be on the right side. Pixels where the person was standing and where the person is standing now, have changed. Their color and flickering characteristic have changed, so the masks have to be adjusted in both regions where the person was earlier and where the person is now.

The second reason, the masks have to be updated, are small changes in Q and FPS values or their deviations from the integer values used during the initialization of the algorithm. If the scene had no changes and masks were not adjusted, then after a while flickering would have increased because the calculated value N is just an integer approximation of the real length of the period of the flickering.

First, the similarity indicators (4) of corresponding frames are counted for each pixel with coordinates x,y:

$$similar(a,b) = 1 \; when \mid a-b \mid < T_1 \; else \; similar(a,b) = 0 \tag{4}$$

$$P(x,y) = \sum_{j=r_i-Z}^{r_i} similar(R[j](x,y), R[j-N](x,y)) \tag{5}$$

T_1 is constant. For pixel brightness value varying from 0 to 255, we use the experimentally selected value of 10 as T_1 value.

In (5) we calculate the similarity level of pixel *P(x,y)*. If a pixel has changed because of a local movement in a scene then the similarity level would be low. As mentioned earlier the algorithm modifies masks to track if their fragments should be updated or not. A single matrix P with dimensions equal to the dimensions of every frame is used to store counts of corresponding frames that were similar. Corresponding frames are frames from different blocks but in the same positions. For example frames with indexes: *2, 2+N, 2+2 N, 2+3 N*, and so on, are corresponding. Similarities of corresponding frames are counted in P on the level of every pixel independently.

Similarly, a single matrix L with dimensions equal to the dimensions of every frame is used as a flicker counter. Pairs of consecutive frames are compared pixel-wise and, if differences are bigger than some constant, then the counter is increased. At the end of each block of N consecutive frames *P(x,y)* is compared with $T_2 * N$ value. If *P(x,y)* is lower or equal to the threshold, then we interpret it as the result of a movement in these areas and we reset the flicker counter *L(x,y)* for these areas to 0. As stated before, we do not remove flickering from pixels marked as a movement. If similarity levels are higher than the threshold, then we interpret it as the flickering effect which should be reduced. In such case the flickering level for the area is increased:

$$\begin{aligned} &if \; (r_i \; mod \; N) == 0 \; then \\ &if \; (P(x,y) < T_2 * N) \; then \; L(x,y) = 0 \\ &else \; L(x,y) = L(x,y) + \left(\sum_{j=r_i-Z+1}^{r_i} similar \; (R[j-1](x,y), R[j](x,y)) < Z \right) \end{aligned} \tag{6}$$

After experiments, value 0.7 was selected for constant T_2. At last, if the levels of flickering exceed threshold T_3 (experimentally set to the value of 3), the algorithm adjusts masks M to reduce flickering:

$$\begin{aligned} &\textit{if } L(x,y) > T_3 \wedge (r_i \bmod N) == Z \textit{ then} \\ &\quad L(x,y) = 0 \\ &\quad M[j](x,y) = R[j+1](x,y) - R[r_i - Z](x,y) \quad \textit{where } j \textit{ is from } r_i - N \textit{ to } r_i \end{aligned} \tag{7}$$

4.5 Fifth step: Remove noise

The current frame with the removed flickering effect is compared with the previous one also with removed flickering. The result of this comparison is a binary differential frame, with white pixels, where the movement was detected. Our algorithm operates on single pixels and does not take into account their direct neighbors. In a differential frame, there may appear pixels that are white but should be black. We can remove such noise by using simple observation: white pixels should appear in groups since they represent a movement of objects much larger than a single pixel.

For each pixel in a differential frame, all 8 neighboring pixels are checked. The count of the neighboring white pixels is calculated. If this number is smaller than some threshold T_4, then the white pixel is treated as noise and turned into a black one.

When using this algorithm with streams generated by cameras pointed at badminton court to track shuttlecocks we have experimentally chosen two values for T_4: 8 for cameras set close to the court, where the shuttlecock when in the view, is quite big, and 6 for cameras located high above the court, where the flying shuttlecock is represented by just a few pixels.

Let $R[r_i - 1]$ and $R[r_i]$ be consecutive frames with indexes $r_i - 1$ and r_i, let D be a binary frame in which 1 indicates detected movement, and 0 indicates no movement. T_1 is the same constant that was used for similarity detection. D is calculated as (11):

$$iswhite(A(x,y)) = 1 \textit{ when } A(x,y) > T_1 \textit{ else } iswhite(A(x,y)) = 0 \tag{8}$$

$$B = |R[r_i - 1] - R[r_i]| \tag{9}$$

$$\begin{aligned} W = {} & iswhite(B(x-1,y-1)) + iswhite(B(x-1,y)) + iswhite(B(x-1,y+1)) \\ & + iswhite(B(x+1,y-1)) + iswhite(B(x+1,y)) + iswhite(B(x+1,y+1)) \\ & + iswhite(B(x,y-1)) + iswhite(B(x,y)) + iswhite(B(x,y+1)) \end{aligned} \tag{10}$$

$$\textit{if } W < T_4 \textit{ then } D(x,y) = 0 \textit{ else } D(x,y) = 1 \tag{11}$$

For every frame, calculated matrix D is returned to the object detection and tracking algorithm.

5 Experiments

As explained earlier our algorithm was designed specifically as the first part of an object detector of small and fast-moving objects in fast video streams. Other flickering removal solutions found in literature and available commercially focus on the visual improvement of movies. Our algorithm produces differential frames to be used by object detector, but can also

produce visually improved movies, which we compare to solutions found in literature and available commercially.

The algorithm was implemented in two highly optimized versions in C++. Both produce the same results which are in line with the presented above algorithm but change some parts to run faster. For example, similarity levels are not recounted all the time from zeros but instead use partial sums counted once.

The first version uses all available cores of the CPU to perform all operations on pixels in parallel. Intel's Threading Building Blocks are used as a parallel framework.

The second implemented version uses GPU to process all operations on images and CPU for logic, masks selection, and control. It adds 7 custom OpenCL kernels. We chose OpenCL as the most portable way to implement parts of the algorithm on GPU.

Both versions were tested on a laptop with Intel Core i7-7500U 2.70GHz with integrated graphics card Intel HD Graphics 620.

5.1 Example 1

A video recorded in Spodek arena in Katowice during World Senior Championships. Data available at: https://1drv.ms/u/s!ApYchjX9LRlxjxyaUNrckiq6Orn4?e=VeD1Tc It has the following properties: different movement - small and big objects, 190 FPS, "ghosts" are visible, lighting is from many artificial sources. The light intensity varies in different areas of the scene and changes cyclically. It looks like the lighting flows through the scene (Fig. 3).

In the differential image after removing flickering, correctly, only the movement of the flying shuttlecock and fragments of the player were detected. A part of a court line was also detected as "moving" because a player casts a shadow that moves over the line (Fig. 4).

5.2 Example 2

A video recorded at a junior tournament in Warsaw. Data available at: https://1drv.ms/u/s!ApYchjX9LRlxjx30jepAl6u24O78?e=qFgtY5. The movement is properly detected near and far from the camera. The camera operates at 150 FPS. "Ghosts" are much less visible than in example 1 (see: observation 1 below). The scene is illuminated by natural and artificial light.

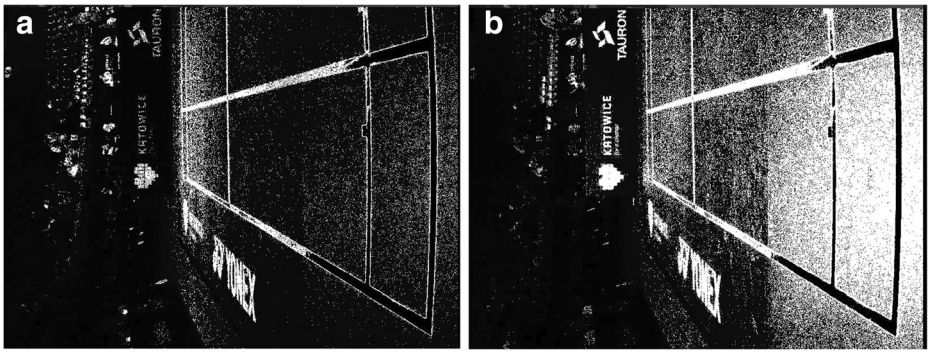

Fig. 3 Differential images calculated for two distinctive pairs of consecutive frames

Fig. 4 Top left: original image, top right: image after applying flicker remover, bottom left: differential image without flicker removal, bottom right: differential image after flicker removal

5.3 Example 3

A video recorded during training. Data available at: https://1drv.ms/u/s!ApYchjX9LRlxjx7jSC62KXkUqvpe?e=pPVpqS.The movement is far from the camera, and tracked objects are small, especially the shuttlecock. The camera operates at 150 FPS. The scene is illuminated by mostly flicker-free artificial light. Only the upper right corner is illuminated by a flickering light source. "Ghosts" are less visible. Because the shuttlecock is a very small moving object, it is sometimes treated as noise and is removed. To overcome this problem we propose a different value of threshold T_4, for cameras far away from the court (see: observation 2 below).

5.4 Example 4

Another video recorded in Spodek arena in Katowice during World Senior Championships. Data available at: https://1drv.ms/u/s!ApYchjX9LRlxkAvqWPQq1Cq6JBBf?e=xE2ShP. In one clip we have three different situations: (a) court is empty, and there is only a movement in the background, on the courts behind the observed court, (b) a moving player is visible, but there is no shuttlecock in the scene, (c) there is a player and the shuttlecock, that we want to detect and track. The movie has the same properties as Example 1: different movement - small and big objects, 190 FPS, "ghosts" are visible, lighting is from many artificial sources.

6 Results

We compare our algorithm with 3 other solutions:

- flicker removing plugin: DeFlicker V2 [4] for the commercial video editing program Nuke [16],
- flicker removing plugin: FlickerFree [5] for the commercial video editing program Nuke [16],
- CPU implementation of an algorithm introduced in [15], we call this method 'Histograms'.

We have chosen the above solutions because they are mentioned in literature - for example, Bonneel [3] compares his results to [4], Kanj [9] compares results to [15], which is especially interesting for us, because it is a very fast algorithm.

We designed and conducted 3 types of tests:

1. Speed test - some solutions produce better results than our, but they are way too slow to be used as an online real-time flicker remover.
2. Accuracy test - for each method we compare movies before and after flicker removal by calculating 3 different norms.
3. Applicability test - we measure how effectively the motion detector works after applying each of the compared algorithms.

6.1 Speed test

We measured the time of executing all algorithms on 1000 consecutive frames from all examples. We did 5 runs for each setting and calculated the average.

Speed comparisons are presented in Table 1.

6.2 Accuracy test

We calculated the temporal stability metric proposed in [11]. Since our cameras do not move and our algorithm internally detects which pixels belong to static, not moving objects and background and which to moving parts of the image, we do not use flow detection and

Table 1 Algorithm speed comparison

	Time to process 1000 frames 800×600 on ASUS Notebook Intel i7 7th Gen
Our CPU	3.66 s
Our GPU	1.19 s
DeFlicker CPU	1193.00 s
DeFlicker GPU	252.22 s
FlickerFree CPU	1108.50 s
Histograms CPU	1.31 s

warping before calculating an error. Based on these simplifications, the error between two frames is defined as:

$$E(V_t, V_{t+1}) = \frac{1}{\sum_{x=1}^{W}\sum_{y=1}^{H} P_t^{(x,y)}} \sum_{x=1}^{W}\sum_{y=1}^{H} P_t^{(x,y)} \left(V_t^{(x,y)} - V_{t+1}^{(x,y)}\right)^2 \quad (12)$$

where V_t and V_{t+1} are consecutive frames and $P \in \{0, 1\}$ is a mask denoting pixels that have not changed between frames (equals 1) because of a local movement in a scene (all "moving" pixels have value 0). W and H are the width and height of the images, t is a frame index, and x, y are indexes of the pixels.

The error of a video is calculated as:

$$E(V) = \frac{1}{T-1} \sum_{t=1}^{T-1} E(V_t, V_{t+1}) \quad (13)$$

where T is a number of frames in the video.

In our case, this metric shows how well the flickering effect is removed from pixels that the algorithm classified as belonging to not moving background and objects. Since the algorithm is used as a preliminary filter to movement detection based on calculating differential frames from pairs of consecutive frames, it is not important if the flickering is removed from pixels detected as belonging to moving objects. The error is calculated after some number of initial frames which are processed to learn the flickering pattern, and after which flickering is being removed.

A comparison of values of calculated errors for example movies (1–3 from above) is shown in Table 2.

We also calculated Peak Signal To Noise (PSNR) and Mean Structural Similarity (MSSIM) for all the examples. Results are shown in Tables 3 and 4.

Other metrics used to measure the effectiveness and accuracy of flickering removal algorithms like Temporal Consistency Model proposed in [12] or Perceptual similarity used in [11] cannot be used for our examples and video streams for which this algorithm is used as part of the system mentioned in the introduction. There are no ground-truth samples without flickering effect to compare to.

6.3 Applicability test

For this test, from a large library of short clips that we collected during the tournaments, we selected a representative movie: Example 4. We manually marked positions of the shuttlecock

Table 2 Temporal stability metric comparison (less is better)

	E(V) without flicker removal	E(V) after flicker removal our solution	E(V) after flicker removal with DeFlicker	E(V) after flicker removal with FlickerFree	E(V) after flicker removal with Histograms
example 1	6.79	1.67	1.86	1.08	6.06
example 2	8.64	4.37	0.36	0.31	6.21
example 3	1.92	1.15	0.41	0.69	1.22
example 4	8.22	1.73	2.74	0.96	7.79

Table 3 Peak Signal To Noise comparison (more is better)

	PSNR without flicker removal	PSNR after flicker removal our solution	PSNR after flicker removal with DeFlicker	PSNR after flicker removal with FlickerFree	PSNR after flicker removal with Histograms
example 1	29.84	42.48	39.09	43.63	29.76
example 2	30.74	35.60	57.10	49.13	33.45
example 3	42.17	46.25	50.01	46.72	43.86
example 4	30.14	41.34	35.08	44.31	30.28

on all frames of the movie when it starts being visible to the moment of the first contact of the shuttlecock with the ground. Our tracker tracks shuttlecock only to the first contact with the ground, so this way we prepared data to compare the effectiveness of our tracker when using different methods of flicker removing as a preprocessing step. Each method removed flickering, and then we calculated differential frames that were passed to the moving object detection algorithm, which is part of the system [7]. The results are summarized in Table 5.

As can be seen the best results were achieved with FlickerFree, which is a bit better than our solution. The worse is a histogram method.

Figure 6 explains the poor true positive rate (TPR) for DeFlicker. DeFlicker produces "ghosts" for small fast-moving objects. This is for us highly undesirable because we track small objects later on. In our solution, we do not observe "ghosts" for such objects. Other norms presented in the above tables show that DeFlicker is a bit better than our solution, but for solving our problem, because of low TPR value, our algorithm outperforms DeFlicker. The histogram method is very fast but has the worst results, especially false positive is high.

7 Discussion

Our approach addresses the problem of removing the flickering effect from movies recorded by stationary high-speed cameras in artificial lighting. It is designed as a preprocessor for small objects tracker and is not meant to visually correct movies with flickering, although it does that pretty well.

FlickerFree plugin [5] produces better visual results than our solution. Results from DeFlicker [4] are comparable with ours. Neither FlickerFree nor DeFlicker can be used for real time-flicker removal because they are way too slow. Our solution is orders of magnitude faster than the plugins. Unlike the plugins, our method was not designed to improve the visual quality of films, but as shown by the results of experiments, its use significantly reduces the

Table 4 Mean Structural Similarity comparison (more is better)

	MSSIM without flicker removal	MSSIM after our flicker removal	MSSIM after flicker removal with DeFlicker	MSSIM after flicker removal with FlickerFree	MSSIM after flicker removal with Histograms
example 1	92.37%	97.93%	98.53%	98.61%	95.82%
example 2	89.30%	92.45%	99.72%	99.73%	94.05%
example 3	98.02%	98.79%	99.70%	99.42%	99.10%
example 4	93.19%	98.01%	97.37%	99.03%	96.20%

Table 5 Results of applicability test

	Ground truth	No flicker remover	Our	DeFlicker	FlickerFree	Histogram
True positive (TP)	45	32	43	23	45	8
True negative (TN)	4704	4573	4704	4704	4704	4630
False positive (FP)	0	0	0	0	0	74
False negative (FN)	0	176	2	22	0	37
Sensitivity (TPR)		15.38%	95.56%	51.11%	100%	17.78%
Specificity (TNR)		100%	100%	100%	100%	98.42%
Accuracy (ACC)		96.32%	99.96%	99.54%	100%	97.77%
Balanced accuracy (BA)		57.70%	97.78%	75.56%	100%	58.10%

flicker effect. Although values of norms calculated for selected examples can be comparable or sometimes even better, than for other methods, our solution leaves temporal flickering "ghosts" in places left by the moving objects (see: Fig. 5). Still, for many cases, it can be acceptable or even unnoticeable by the viewer.

DeFlicker plugin very poorly deals with small, fast-moving objects (which we want to detect and track). The example of magnified fragments of frames produced by DeFlicker plugin and our algorithm are compared in Fig. 6. As you can see the shuttlecock in DeFlicker frame is blurred and artificially stretched, while the same shuttlecock after using our algorithm is perfectly visible, has proper, original shape, and is much less blurred.

We implemented an algorithm from [15] based on histograms, because it had the potential to be fast. Indeed it is fast - it is 2.8 times faster, than ours on CPU, and only slightly slower than our version on GPU. Our method removes flickering better than [15]. Not only metrics calculated for this method and shown in Tables 1, 2, and 3 indicate overall poorer performance than our method, but also differential frames show some unwanted artifacts and noise - see: Figs. 7 and 8.

Fig. 5 The "ghost" effect and the noise

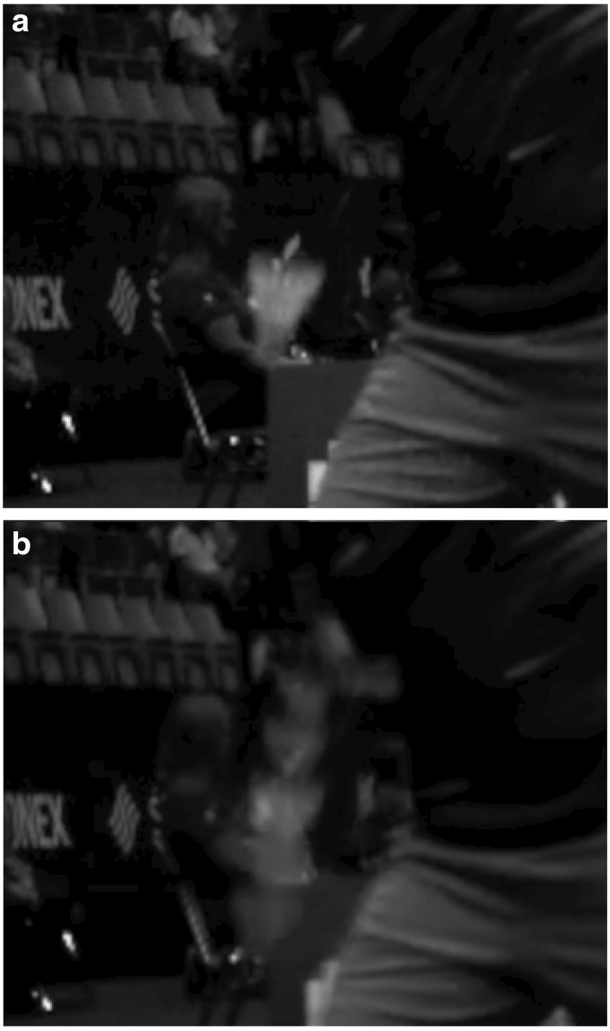

Fig. 6 Shuttlecock: a fast-moving object in frames with removed flickering by our algorithm (upper), and by DeFlicker plugin (lower)

In Fig. 7 we compare selected frames from example 2. In the first row, there are differential frames with our algorithm applied. In the second row, there are the same differential frames calculated from the results of the algorithm [15], and in the third row are the same differential frames but without flicker removal. The shuttlecock approaching and hitting the ground is visible in all sequences but distinguishing it from the noise caused by flickering needs additional computing steps sensitive to misclassification and errors. After flickering removal, only the moving shuttlecock is visible in differential frames. As can be seen in the same sequence, both algorithms removed flickering at a level allowing detection of moving objects in differential frames, but using algorithm [15] produces slightly worse results than ours.

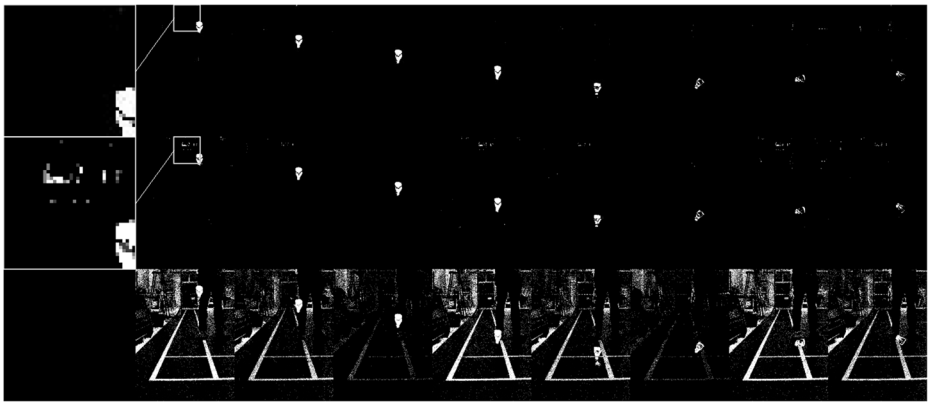

Fig. 7 Comparison of selected differential frames with our flicker remover (upper), flickering removal by algorithm [15] (middle), without flickering removal (lower)

Calculating differential frames from the results of the algorithm [15] can sometimes produce unexpected artifacts, like big, static areas detected as moving objects. Such a situation is shown in Fig. 8. In two middle frames, there are visible big advertisement boards. For comparison, our algorithm worked correctly for the same sequence.

These unexpected artifacts are the main cause of false positives generated using [15] in the applicability test. In this test, our algorithm performed only slightly worse than [4] and better than the rest. We conducted tests on many more examples, and this selected one is representative.

The data set is unbalanced: there are many more frames without a shuttlecock than there are with a flying one. In scenes analyzed by our system [7], this is common. Most of the time in the streams of images captured by cameras, there is a movement only in the background, sometimes moving players are visible, and only once in a while the shuttlecock approaches and hits the ground. In this very moment of hitting the ground,

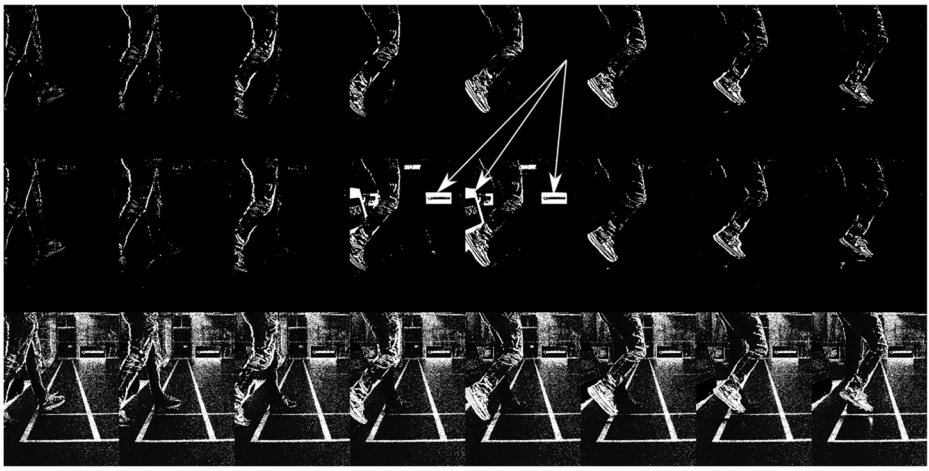

Fig. 8 Second example of comparison of selected differential frames with our flicker remover (upper), flickering removal by algorithm [15] (middle), without flickering removal (lower) [15]

our system [7] should properly detect and calculate the exact position of the shuttlecock in relation to the court. To do it we need to track the shuttlecock before touching the ground and analyze its trajectory. Flickering removal is the first, very important step of the robust detection of shuttlecocks approaching the ground.

Sometimes [15] has more true positives, but almost always produces the most false positives. In Example 4 it was possible to detect the shuttlecock without removing flickering (of course with much worse effectiveness, than when the flickering is removed), but in most cases, our detector cannot detect a shuttlecock if the flickering is not removed. The tracker sometimes does not detect the shuttlecock in single frames when using our algorithm to remove flickering. We deal with this problem by calculating trajectories of the shuttlecock and interpolating missing positions as a next step after the tracker in the system [7]. The ratio of missing detections is as low as a few percent, making interpolations accurate and robust.

7.1 Observation 1

- The relationship between flicker frequency and camera speed should be an exact multiple.
- Example: 50: 197 is worse than 50: 190, which is worse than 50: 150.
- The above relationship affects the length of the masks set.
- The above relationship also affects the size of "ghosts" and the time of its disappearance.

7.2 Observation 2

To track the movement of small objects, reduce the limit on the number of neighbors used when removing noise from 8 to 6 or even 3.

8 Conclusion

We have described the algorithm for real-time flicker removal from fast video streaming. Our approach uses fast methods of flickering reduction to increase the effectiveness of the detection of fast-moving objects.

For most scenes, our algorithm cannot be used for repairing movies with a flicker effect because of "ghosts" that may be visible. It does not produce visually satisfactory results for all scenes. It was designed for when one needs to detect a movement in a scene. It is very fast and when a frame is processed the algorithm does not analyze future frames. That makes it useful for removing flickering in real-time from captured images from a camera. Comparing to other solutions ([4, 5]) our algorithm is over 300 times faster and produces better results than [5, 15]. It is only slightly less effective than [4], but [4] and similar solutions are too slow to be used as a preprocessing step for moving object detector in our system [7].

Our algorithm

- effectively removes flickering sufficiently for flawless motion detection,
- works correctly regardless of the expected speed of the camera (faster than main frequency),
- adjusts to small changes in flicker frequency,
- adapts to the small changes in camera speed,

- correctly removes flickering caused by different artificial light sources,
- has a satisfactorily low level of erroneous movement indications,
- works very fast,
- uses the local correction method without the kernel, but with areas reduced to single pixels.

As stated earlier our algorithm is useful for real-time movement detection process, we see improvement which can be done in further work:

1. support for lost frames not only for mask selection but also for further processing and updating of masks,
2. expanding the algorithm to use a small Gaussian kernel instead of individual pixels when we count the value of the similar() function - it will probably be slower (maybe on the GPU imperceptibly), but the last step of rejecting false positives can be omitted,
3. "ghosts" detection and removal.

Availability of data and material

1. Source frames movie used for example 1 and result flicker-free movies: https://1drv.ms/u/s!ApYchjX9LRlxjxyaUNrckiq6Orn4?e=VeD1Tc.
2. Source frames and movie used for example 2 and result flicker-free movies: https://1drv.ms/u/s!ApYchjX9LRlxjx30jepAl6u24O78?e=qFgtY5.
3. A movie used for example 3 and result flicker-free movies: https://1drv.ms/u/s!ApYchjX9LRlxjx7jSC62KXkUqvpe?e=pPVpqS.
4. A movie used for example 4 and result flicker-free movies:https://1drv.ms/u/s!ApYchjX9LRlxkAvqWPQq1Cq6JBBf?e=xE2ShP.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Code availability Source code of the algorithm is available at https://github.com/flyeyesport/flicker_remover.

References

1. Baudry S, Chupeau B, De Vito M, Doerr G (2015) Modeling the flicker effect in camcorded videos to improve watermark robustness. In IEEE National Conference on Parallel Computing Technologies, pages 42–47, Bengaluru, India
2. Bay H, Tuytelaars T, Van Gool L (2006) Surf: Speeded up robust features. In European Conference on Computer Vision, pages 404–417. Springer, Graz, Austria
3. Bonneel N, Tompkin J, Sunkavalli K, Sun D, Paris S, Pfister H} (2015) Blind Video Temporal Consistency}, ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2015), volume 34, numer 6
4. DEFlicker – RE:Vision Effects [accessed 2020 Mar 20] https://revisionfx.com/products/deflicker/.

5. Digital Anarchy's Flicker Free Removal Software. Flicker Free Plugin: Deflicker Time Lapse, LED and Slow motion / High Frame Rate [accessed 2020 Mar 20]. https://digitalanarchy.com/Flicker/main.html.
6. Farnebäck G (2003) Two-frame motion estimation based on polynomial expansion. In: Bigun J, Gustavsson T (eds) Image Analysis. SCIA 2003. Lecture Notes in Computer Science, vol 2749. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45103-X_50
7. FlyEyeSport instant review system for badminton. [accessed 2020 Mar 20]. https://en.flyeyesport.com.
8. Intel Threading Building Blocks. [accessed 2020 Mar 20]. https://software.intel.com/en-us/tbb.
9. Kanj A (2017) Flicker Removal and Color Correction for High Speed Videos. Other [cs.OH]. Université Paris-Est. English. ffNNT : 2017PESC1115ff. fftel-01744714f
10. Kanj A, Talbot H, Luparello RR (2017) Flicker removal and superpixel-based motion tracking for high speed videos. 2017 IEEE International Conference on Image Processing (ICIP) 245–249. https://doi.org/10.1109/ICIP.2017.8296280.
11. Lai WS, Huang JB, Wang O, Shechtman E, Yumer E, Yang MH (2018) Learning Blind Video Temporal Consistency. In: Ferrari V, Hebert M, Sminchisescu C, Weiss Y (eds) Computer Vision – ECCV 2018. ECCV 2018. Lecture Notes in Computer Science, vol 11219. Springer, Cham. https://doi.org/10.1007/978-3-030-01267-0_11
12. Li C, Chen Z, Sheng B et al (2020) Video flickering removal using temporal reconstruction optimization. Multimed Tools Appl 79:4661–4679. https://doi.org/10.1007/s11042-019-7413-y
13. Lowe DG (1999) Object recognition from local scale-invariant features. In The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999., volume 2, pages 1150–1157, Kerkyra, Greece
14. Lucas BD, Kanade T (1981) An iterative image registration technique with an application to stereo vision. In International Joint Conference on Artificial Intelligence, volume 81, pages 674–679, Vancouver, BC, Canada
15. Naranjo V, Albiol A (2000) Flicker reduction in old films. Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101), 2:657–659
16. Nuke, NukeX, Nuke Studio | VFX Software | Foundry [accessed 2020 Mar 20] https://www.foundry.com/products/nuke.
17. OpenCL Overview The Khronos Group Inc. [accessed 2020 Mar 20]. https://www.khronos.org/opencl/.
18. Piti'e F, Kent B, Collis B, Kokaram A Localised deflicker of moving images. In Proceedings of the 3rd IEE European Conference on Visual Media Production (CVMP'06)
19. Richardson P, Suter D (1995) Restoration of historic film for digital compression: a case study. Proceedings, International Conference on Image Processing, Washington, DC, pp 49–52 vol.2. https://doi.org/10.1109/ICIP.1995.537412
20. Rublee E, Rabaud V, Konolige K, Bradski G (2011) ORB: an efficient alternative to SIFT or SURF. 2011 International Conference on Computer Vision, Barcelona, pp 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544
21. Tai Y-W, Jia J, Tang C-K (2005) Local color transfer via probabilistic segmentation by expectation-maximization. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 1, pages 747–754, San Diego, USA
22. UEFA Stadium Lighting Guide 2016. [accessed 2020 Mar 20]. https://www.uefa.com/MultimediaFiles/Download/uefaorg/General/02/36/26/72/2362672_DOWNLOA D.pdf.
23. Wong KK, Das A, Chong MN (2004) Improved flicker removal through motion vectors compensation. Third International Conference on Image and Graphics (ICIG'04), Hong Kong, China, pp 552–555. https://doi.org/10.1109/ICIG.2004.85
24. Zeng YC, Lin SY, Shih YP, Liao HYM (2009) Intensity Flicker Removal in Digitized Old Films Using Global-Local Histogram Transform. In: Muneesawang P, Wu F, Kumazawa I, Roeksabutr A, Liao M, Tang X (eds) Advances in Multimedia Information Processing - PCM 2009. PCM 2009. Lecture Notes in Computer Science, vol 5879. Springer, Berlin, Heidelberg
25. Zhang XL, Xu J (2014) A Background Subtraction Algorithm Robust to Intensity Flicker Based on IP Camera, Ranran J Multimed, vol. 9, NO. 10