Detecting the position of feet on the floor of a large virtual reality cave in the Immersive 3D Visualisation Lab

Krzysztof Karpusiewicz

Jan Kwarcinski

Mateusz Kołakowski

Krzysztof Dubanowicz

***Abstract*—Feet position detection in a CAVE system can be important for tracking user's position during application operation. Currently, in the Immersive 3D Visualisation Lab at the Gdansk University of Technology, it is possible to determine the user's position based on the location of markers on the glasses, but this is insufficient to determine where the person's feet are. To solve this problem, we propose a method that can determine the location of the feet of people standing on the floor of the CAVE based on the shadows seen from under the floor.**

PROBLEM DESCRIPTION The Immersive 3D Visualisation Lab has a position detection system based on locating markers on glasses, allowing for simulating the parallax effect. Such a solution is optimal for accurately determining the position of the head of a person in a cave, allowing the parallax effect to be performed reliably. However, it does not determine the position of the rest of the body. While the position of the hands is easy to determine using controllers typical in VR environments, the current system has no method to determine the position of the feet of people in the cave. Our goal is to create a foot tracking method based on foot shadows on the cave floor. The semi-transparent cave floor allows the image to be projected from the outside in such a way that it is seen by the person inside. A side effect of the semi-transparency is the shadow cast by the person standing on the floor, as seen from under the cave (Fig. 1). The detection method analyses video from cameras under the cave to determine the existence and position of the shadows, allowing us to locate the user's feet.

State of the art

Due to the nature of the problem and the CAVE system, there are almost no sources presenting existing solutions. During the literature review, we used the following databases: Scopus, SpringerLink and IEEE. 766 articles were retrieved, from which 9 items were selected.

At an early stage of the work, we considered using an algorithm based on machine learning [1], but during the verification of current approaches [3], we decided to rely on classical contour finding algorithms. An additional problem with the mentioned approach would be the need to manually prepare and label a large training set. However, this approach should not be completely discarded in the future - using the foot detection

Published by the IEEE Computer Society

Figure 1. A view of the CAVE floor from below

heuristic presented in our paper, such a set can be easily generated with much less effort (only a manual correction of possible mislabelings is necessary).

Many of the papers dealt with image subtraction of images [4] , but they did not directly address the problem at hand. Most discussed the methods used for motion analysis, i.e., subtracting images from two moments in time, which makes it possible to select changes and detect areas where motion has occurred. Unfortunately, these methods cannot be directly transferred to the feet detection problem because the perspective distortion occurring between the image displayed on the CAVE floor and the image observed by the camera is not taken into account. In addition, the perceived image is distorted by reflections from the lamp light of the projector beneath the floor and (for darker floors) reflections of other equipment beneath the cave.

We considered using the previous frame in the video analysis to reduce situations where a foot is detected in a location to which it could not have moved or a foot being lost by the algorithm. The algorithm that would be used for this purpose is described in the paper [8].

One of our references during our work, was the article [9], which presented a very efficient way to detect ellipses and presented achievable times. We used the repository provided by the authors of the article to see if the detection of ellipses could be directly transferred to feet recognition in CAVE or if it could be adapted to it. The first tests ended with satisfactory results, therefore we decided to use a similar approach.

Image Subtraction Problems

A typical solution for finding changing objects in a video image is to use the image subtraction technique, which consists of finding differences between two images created at different times. Such an operation is widely used, especially in motion detection, but in the case of our problem, motion can be caused by both a change in the display image and a shadow cast by the detected feet. Instead of varying the images over time, we can extract two images, where the only differences are the elements of interest. These images are the image taken from the camera (Fig. 2) and the image projected on the cave floor (Fig. 3). Theoretically, the difference of these two images would allow the shadowed areas to be perfectly distinguished. However, out trials have shown significant problems with this approach (Fig. 4): correction of camera image distortion is difficult and inefficient.

The problems with this approach are caused by various optical distortions, both those associated with the camera and described in the "Proposed Solution" section, and distortions caused by the optics of the projector projecting the image

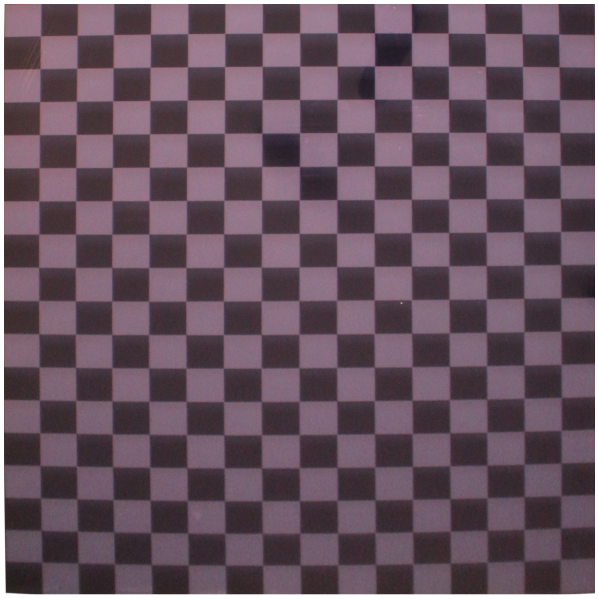

Figure 2. Image based on which image subtraction test was performed, after perspective and lens distortion correction

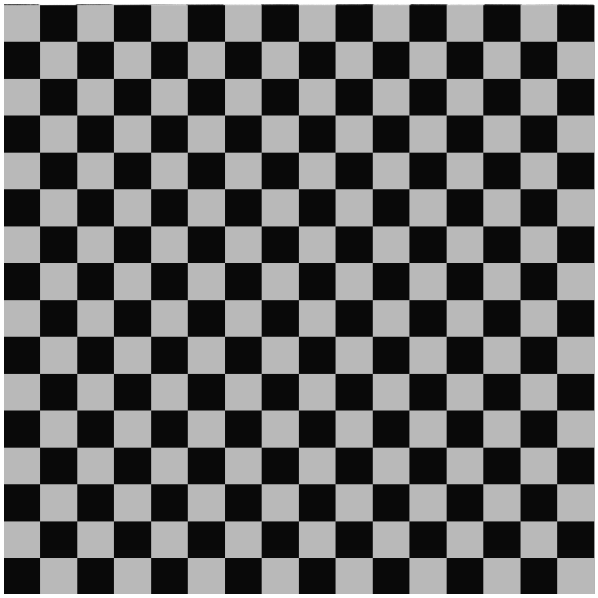

Figure 3. Image displayed on the floor of the cave

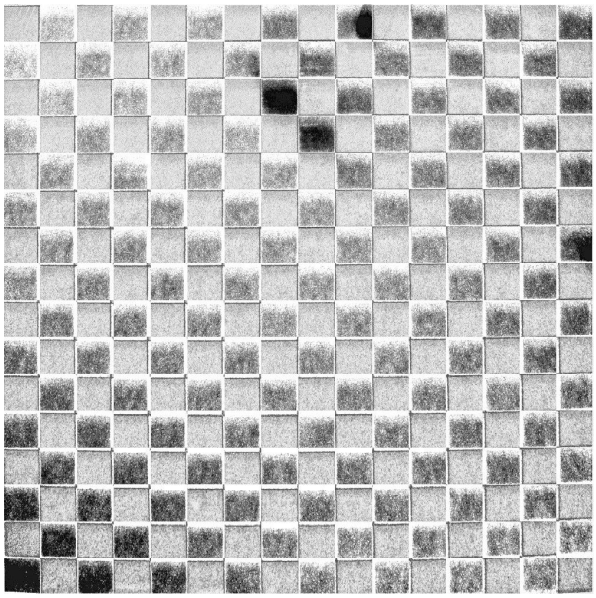

Figure 4. Image after subtraction is performed

onto the cave floor. A bigger problem, however, is the difference in brightness and color of the different parts of the floor. These differences are caused by light from the walls, ceiling, and cave environment and do not seem to be easily removable. A prominent example of such light would be the red tint in the image seen on the floor (Fig. 2), which is caused by the image displayed on the walls.

Above you can see the result of manually trying to match and subtract the images: the projected image and the extracted image from the camera. The extreme dark areas are where the difference between the images is greatest. Take note of the bottom left corner of the image, where the significant difference is caused by different brightness of the camera image in some parts of the floor.

Despite its low utility in the detection stage, the displayed image can potentially be used to find false positives found by our proposed algorithm, caused, for example, by the display image, that do not appear to be detectable by other means.

Proposed Solution

The method we used can be divided into two stages, preprocessing and actual detection. The first stage of image preprocessing (Fig. 5),

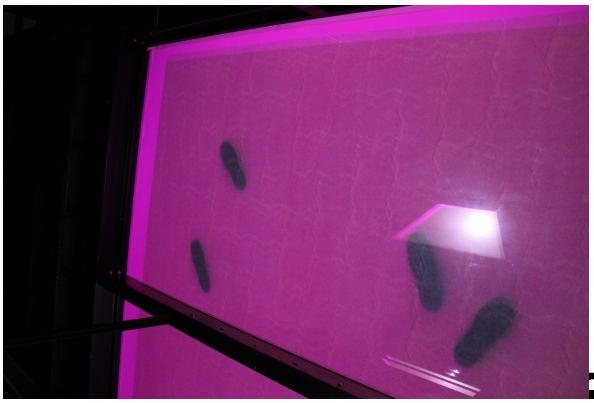

Figure 5. Example image on which the detection was performed. Note that the right part of the image, where the projector glare is located, is rejected

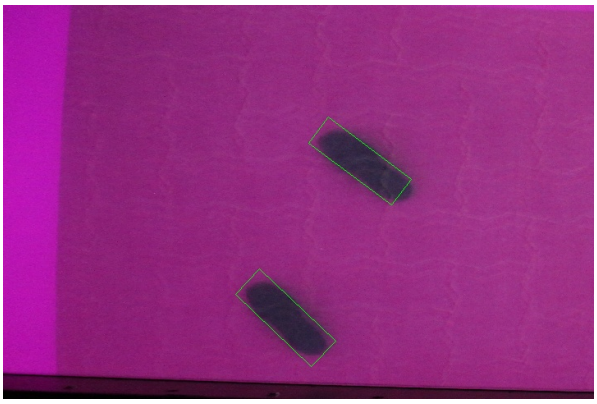

Figure 6. Image after perspective correction. The positions of the feet detected at the next stage have been plotted on it

in order to improve the performance in further stages, is to extract the area of interest in the image, and to perform the correction of distortions caused by the camera perspective (Fig. 6). The calibration defining the cave floor area was done manually, but since the image capture cameras are to be mounted under the cave, the requirement for manual calibration does not seem to be a problem.

The next step, however, which was discarded in the final processing version because of performance issues, was to perform a correction for the barrel distortion introduced by the lens of the device used to take the images. The correction was intended to improve the accuracy of matching the input image with image displayed on the

Figure 7. Binarization result

floor, but after discarding this plan, we found that the effect of barrel distortion on our data was relatively small and removing it would not affect the quality of detection.

The operations of conversion to greyscale image, low-pass filtering, and image binarization are performed on the preprocessed image (Fig. 7). Contour detection is performed on the new image, which yields a list of shapes present in the image. We consider contours inscribable in a rectangle with an empirically determined area between 0.9% and 2% of the image area and an aspect ratio between 10:21 and 1:5 to be the feet of a person standing on the floor.

Performance and processing tests

Testing Equipment

Performance tests were performed on a Dell personal laptop running Windows 10. Processor: Intel i7-7700HQ @2.80 GHz, 4 cores, 8 threads, RAM: 16 GB

Input Data

The images were divided into 3 categories (Fig. 8):

- Clean - floor visible in solid color with no details displayed.
- Partially noisy - part of the floor contains displayed elements
- Noisy - the floor shows images with high detail

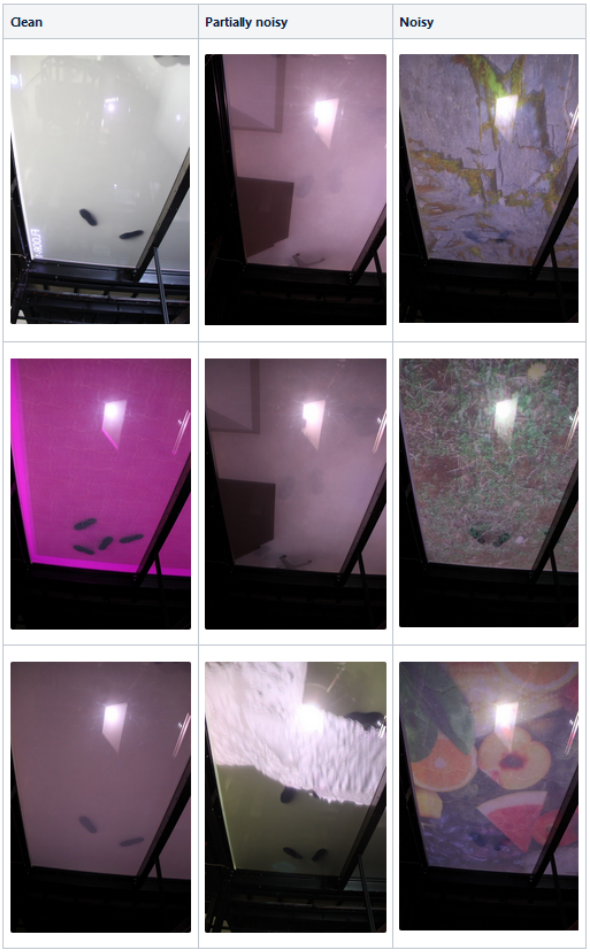

Figure 8. A table showing the cases assigned to the categories defined for the tests

Performance was tested against image categories and the target scaled image resolution. The tests verify that programmatically changing the resolution of the input data can positively affect the performance of the solution. The time to load images into memory is not included in the results.

Perspective correction performance

The following table shows the performance of perspective correction. The results do not include the time it takes to resize the image (Table 1 and Table 2).

.

Table 1. Preprocessing time performance for image types.

Image size	Clean	Partially noisy	Noisy
720x480	15.90ms	16.25ms	16.39ms
1280x720	16.06ms	16.44ms	16.41ms
1920x1080	16.57ms	16.76ms	16.85ms

Table 2. Preprocessing time performance for image types - details

Image type	Image size	Mean	Error	Deviation
Clean	1280x720	16.06ms	0.233ms	0.218ms
Clean	1920x1080	16.57ms	0.320ms	0.381ms
Clean	720x480	15.90ms	0.302ms	0.283ms
Noisy	1280x720	16.41ms	0.272ms	0.227ms
Noisy	1920x1080	16.85ms	0.319ms	0.298ms
Noisy	720x480	16.39ms	0.116ms	0.097ms
Partially noisy	1280x720	16.44ms	0.306ms	0.286ms
Partially noisy	1920x1080	16.76ms	0.274ms	0.229ms
Partially noisy	720x480	16.25ms	0.268ms	0.251ms

Foot Detection Performance

The following table shows the foot detection performance in the finished image. The results do not include the time it takes to change the resolution and correct the perspective (Table 3 and Table 4).

Table 3. Time performance of feet detection for different image types.

Image size	Clean	Partially noisy	Noisy
720x480	20.1ms	19.42ms	18.48ms
1280x720	54.57ms	56.47ms	57.27ms
1920x1080	118.93ms	125.85ms	133.09ms

Table 4. Time performance of feet detection for different image types - details

Image type	Image size	Mean	Error	Deviation
Clean	1280x720	54.57ms	0.607ms	0.538ms
Clean	1920x1080	118.9ms	1.523ms	1.425ms
Clean	720x480	20.10ms	0.238ms	0.222ms
Noisy	1280x720	57.27ms	0.593ms	0.555ms
Noisy	1920x1080	133.1ms	2.605ms	3.478ms
Noisy	720x480	18.48ms	0.505ms	1.472ms
Partially noisy	1280x720	56.47ms	0.862ms	0.764ms
Partially noisy	1920x1080	125.9ms	2.432ms	2.389ms
Partially noisy	720x480	19.42ms	0.348ms	0.325ms

Time performance of complete frame processing

The following table shows the performance of the total processing of an image frame. The results include the time taken to change the resolution and correct the perspective (Table 5 and Table 6).

Table 5. Time performance of the whole algorithm run for different image types

Image size	Clean	Partially noisy	Noisy
720x480	38.97ms	39.06ms	39.01ms
1280x720	78.72ms	77.72ms	78.54ms
1920x1080	150.1ms	151.2ms	145.0ms

Table 6. Time performance of the whole algorithm run for different image types - details

Image type	Image size	Mean	Error	Deviation
Clean	1280x720	78.72ms	1.086ms	0.963ms
Clean	1920x1080	150.1ms	2.093ms	1.855ms
Clean	720x480	38.97ms	0.615ms	0.545ms
Noisy	1280x720	78.54ms	1.357ms	1.270ms
Noisy	1920x1080	145.0ms	1.698ms	1.418ms
Noisy	720x480	39.01ms	0.651ms	0.578ms
Partially noisy	1280x720	77.72ms	0.854ms	0.757ms
Partially noisy	1920x1080	151.2ms	2.956ms	2.765ms
Partially noisy	720x480	39.06ms	0.491ms	0.460ms

Performance - summary

The amount of displayed details has no effect on time performance of feet detection. The resolution of the input images has the greatest impact on the foot detection rate (7).

Table 7. Feet recognition performance per image type

Image size	Frames per second
Full HD (1920x1080)	6 fps
HD (1280x720)	12 fps
480p (720x480)	25 fps

Accuracy for different cases

The input data were divided into 3 categories (Fig. 9). The 'clearly visible' category is characterized by high contrast between the feet and background and no visible ground texture. The 'moderately visible' category consists of images with low contrast shadows and optional ground texture. In the barely visible category, the feet shadows are almost imperceptible and the ground texture is dominant.

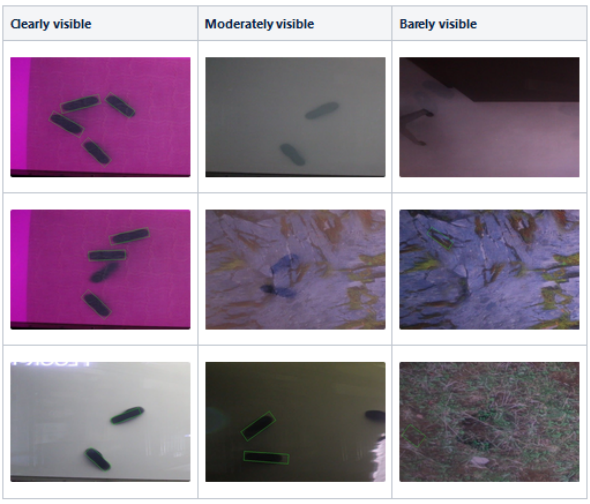

Figure 9. Table showing the quality categories of the input data

No differences in foot detection accuracy were observed for the resolutions given in the 7 table. The foot detection accuracy is satisfactory for the input data categories with good feet visibility (Table 8).

Table 8. Accuracy of feet detection for different image categories. The number of feet is in parentheses

Feet	Detected	Not detected	False positive
Clearly visible	89% (41)	11% (5)	0
Moderately visible	20% (10)	80% (39)	0
Barely visible	0% (0)	100% (14)	13

When processing the source image, the following parameters and their effect on the results were noted:

Wall and floor brightness

The brightness of the walls and floor has a critical effect on the results obtained (Fig. 10). Feet were best detected when the walls had high brightness, while the floor was dark. In the case of dark walls and floor, the resulting image was noisy, causing false detections. In the case of a light floor and walls, feet were not detected.

Figure 10. Example results of the algorithm

Figure 11. Example results of the algorithm

Displayed environment

The level of complexity of the image displayed on the floor had a significant impact on the outcome of the algorithm (Fig. 11). Numerous irregular shapes negatively affected the interpretations, often causing the feet to be invisible to the algorithm; the resulting image was noisy and false positives occurred. A surface with uniform colors and little noise positively affected the results.

Feet position

The results were negatively affected when the slippers were in contact, forming a shape that was rejected by the algorithm.

Shape of slippers

For best results, the shape and size of the slippers should be fixed. This is due to the need to establish the dimensions that the program considers to be the shapes it is looking for.

Camera tilt and position

It is necessary that the camera image is captured at one fixed angle. The calibration of the program, which should be done before the first run, is to determine the exact position of the floor in the image, which ensures correct perspective correction. Because of the planned permanently mounted cameras, this problem does not appear to be significant.

Implementation

We started our work on the library by creating a Proof of Concept in Python, using OpenCV. We used it to see if the algorithm worked for the test images collected in CAVE. After encouraging results, it was decided that the tested algorithm was to be implemented in the final version of the library.

The final implementation of the library was written in C# (as required by the system that uses Unity). We used Emgu CV, which allows usage of OpenCV functionalities in .NET. The library allows running a continuous search for feet positions for a user-specified set of cameras or for a pre-recorded video. The feet positions are acquired by the user by querying an object of the LiveFeetDetector class. The system architecture is shown in Fig. 12.

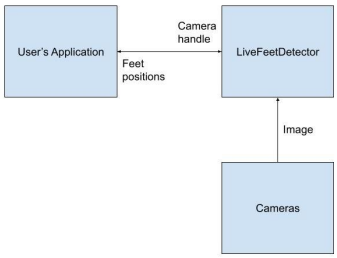

Figure 12. System architecture

The result of feet detection is a list of RotatedRect objects that consist of: position, size, and rotation angle of rectangles $(x, y, w, h, rotation)$. The coordinates and size of the feet are represented as their ratio to the size of the image obtained after perspective correction and are values between 0 and 1. The origin of the coordinate system is the upper left corner of the processed image. It is necessary to translate the positions returned by the library into coordinates in the CAVE environment.

The library consists of two classes - LiveFeetDetector and FeetDetection. The first one is used for real-time image processing. It is necessary to create an instance of the class and provide the camera ID. The client application obtains information about the positions of detected feet by calling the getBoxes method. The second class allows to run detection on a single image and can be used, for example, for testing or for tagging a set of images.

Requirements for the CAVE Environment

Due to the design of the cave, it is not possible to cover the entire cave floor with a single camera. It caused by the glares created by projectors' lamps. The optimal solution seems to be placing four cameras under the corners of the floor. Received images can, after perspective correction, be combined into a single image covering the whole floor, although it is not clear whether this would not affect the quality of detection in the area between them.

Another potential problem is the different shape and size of people's feet. In the Immersive 3D Visualisation Lab, two sizes of slippers are used. For this reason, it is important to set appropriate threshold values for feet detection.

Summary

The main factors that affect the quality of detection are the brightness of the displayed images and the level of complexity of the image displayed on the cave floor. More complex images introduce noise that can prevent detection or cause false detections. In the ideal situation, a uniform or minimally noisy image is displayed on the floor. The brightness of the projected images consists of two components: the brightness of the

image on the floor and the brightness of the image on the walls and ceiling of the cave. Since the shadow cast on the floor is caused by light coming from the walls and ceiling, increasing the brightness of these surfaces increases the contrast of the shadow and makes it easier to recognize. Floor brightness has the opposite effect - a brighter floor decreases the contrast of the shadow, making detection difficult, if not impossible.

The algorithm we have presented allows for correct detection of feet positions in good conditions. To increase the quality of detection, many modifications can be considered. Some of them are: basing the detection results on the previous frame, which may allow to increase the detection confidence. Another possibility would be to pair the feet and retrieve the head position information to determine the person's position more accurately.

REFERENCES

1. Liu, L., Ouyang, W., Wang, X. et al. Deep Learning for Generic Object Detection: A Survey. Int J Comput Vis 128, 261–318 (2020). https://doi.org/10.1007/s11263-019-01247-4.
2. Croitoru, I., Bogolin, SV. & Leordeanu, M. Unsupervised Learning of Foreground Object Segmentation. Int J Comput Vis 127, 1279–1302 (2019). https://doi.org/10.1007/s11263-019-01183-3.
3. Czygier, J., Tomaszuk, P., Łukowska, A., Straszyński, P., & Dzierżek, K. (2020). Classical algorithm vs. machine learning in objects recognition doi:10.1007/978-3-030-17798-0_58.
4. Varghese, A., G, S. Sample-based integrated background subtraction and shadow detection. IPSJ T Comput Vis Appl 9, 25 (2017). https://doi.org/10.1186/s41074-017-0036-1.
5. Cui, B., Créput, JC. A Systematic Algorithm for Moving Object Detection with Application in Real-Time Surveillance. SN COMPUT. SCI. 1, 106 (2020). https://doi.org/10.1007/s42979-020-0118-5.
6. Christie, D. A., & Sukma, T. (2018). Comparative evaluation of object tracking with background subtraction methods. Paper presented at the Proceedings of the 3rd International Conference on Informatics and Computing, ICIC 2018, doi:10.1109/IAC.2018.8780483.
7. Al-Mayyahi, M. H. N., Barnouti, N. H., & Abomaali, M. (2018). Vehicle detection and license plate recognition system. International Journal of Engineering and Technology(UAE), 7(4), 3170-3174. doi:10.14419/ijet.v7i4.19154.
8. Blokus, A., Krawczyk, H. Systematic approach to binary classification of images in video streams using shifting time windows. SIViP 13, 341–348 (2019). https://doi.org/10.1007/s11760-018-1362-1.
9. Lu, C., Xia, S., Shao, M., and Fu, Y., "Arc-support Line Segments Revisited: An Efficient and High-quality Ellipse Detection", arXiv e-prints, 2018.