Real-time Face Tracking in Video Content using Viola-Jones Algorithm

Dušan Đorić, Saša Crnobrnja, Marija Punt, and Mile Davidović

***Abstract* — In this paper an implementation of an application for detecting and tracking faces in real-time using the Viola-Jones algorithm and OpenCV library has been presented. Also, using the developed application it is possible to recover from failure during face tracking. Video content is obtained from a file or webcam.**

***Keywords* — OpenCV, Viola-Jones, Haar features, Object tracking, Computer Vision.**

I. Introduction

FOR the past few years there has been a significant progress in the field of computer vision. Beginning with basic algorithms for face tracking during taking a photo, today we have advanced algorithms which, by using cameras and different sensors, enable cars to reconstruct space around them, safely park and maneuver.

This paper describes implementation of an application which enables face detection, face tracking and recovery from failure during the face tracking. All those actions are done using video content from a file or webcam. Real-time face detection and tracking is done using the Viola-Jones algorithm for face detection. The algorithm is implemented in one of the functions from the OpenCV library which is used in the implementation of this application because of its many features considering image processing. Graphical user interface of this application is realized using Java programming language, whereas the part of the code considering video and image processing is written in C++ programming language. For those two parts to be able to communicate, Java Native Interface is used.

The paper is presented in the following sections. In section II the Viola-Jones algorithm is presented. Graphical user interface and the way of using the application are presented in section III. Section IV describes the architecture of the realized system while section V gives implementation details of the main processing of video frame. Section VI presents the achieved performances of the realized system and section VII concludes the paper.

Paper received May 01, 2019; revised July 19, 2019; accepted July 20, 2019. Date of publication July 31, 2019. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Miroslav Lutovac.

This paper is revised and expanded version of the paper presented at the 26th Telecommunications Forum TELFOR 2018 [11].

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, under grant number: III44009.

Dušan Đorić, Saša Crnobrnja Mile Davidović, Istraživačko-razvojni Institut RT-RK, Narodnog fronta 23a, 21000 Novi Sad, Srbija (e-mail: dusan.djoric@rt-rk.com, sasa.crnobrnja@rt-rk.com, mile.davidovic@rt-rk.com).

Marija Punt, Elektrotehnički fakultet u Beogradu, Bulevar Kralja Aleksandra 73, 11120 Beograd, Srbija (telefon: 381-11-3218-392, e-mail: marija.punt@etf.bg.ac.rs).

II. Viola-Jones algorithms

The Viola-Jones algorithm is used for real-time object detection in images. Using this algorithm, it is possible to detect, track and recover from failure during tracking the face. Face detection is realized by finding Haar features which are distinctive for human face [1]. The finding of Haar features is optimized by using Integral images.

Modified versions of Viola-Jones algorithms can be used for object detection in noisy ultrasound images. The paper [2] describes detection of carotid artery longitudinal section in ultrasound B-mode images.

In paper [3] using the Viola-Jones algorithms for detecting pedestrians is presented. The implementation described runs at about 4 frames/second, but has a very low false positive rate.

In paper [4] is presented an automatic hand gesture recognition system operating on video stream. The system consists of two modules: hand gesture detection module and hand gesture recognition module. The detection module is mainly based on the Viola-Jones method for object detection.

A. Haar features

In this subsection Haar features are presented because of their importance as the core component of the Viola-Jones algorithm.

Haar feature of an image is the ratio of light and dark surfaces on a specific position in the image. Every Haar feature is determined by mutual position of light and dark surfaces and its position in the image. Basic Haar features are shown in Fig. 1.

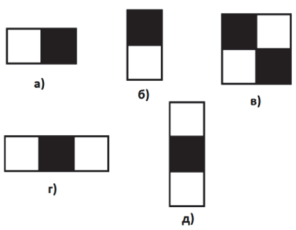

Fig. 1. Basic Haar features.

Derived Haar features can be obtained from basic Haar features in two ways. The first one is to increase the number of pixels which compose light and dark surfaces. The second one is to replace the positions of light and dark

surfaces (Fig. 2). If two Haar features with the same size and same mutual position of light and dark surfaces are not located on the same position in the picture (different coordinates of the most top-left points of the features), then those two Haar features are considered different.

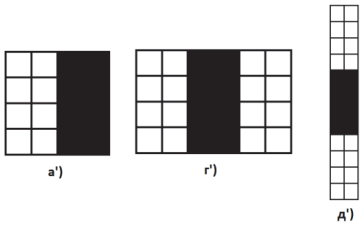

Fig. 2. Derived Haar features.

The picture must be black and white before searching for Haar feature in it. Every pixel from black and white picture has a specific value x, $0 \leq x \leq 255$. In order to find a specific Haar feature, the difference between the average value of light surfaces and the average value of dark surfaces of a potential Haar feature must be equal or higher than the previously specified threshold. This threshold is the same for all Haar features with the same size and the same mutual position of the light and the dark surfaces. In Fig. 3 is shown an example of 4 Haar features located in image. Although Haar features located near eyebrows are the same size and have the same mutual position of light and dark surfaces, they are considered two different Haar features because they are not located on the same position in the picture.

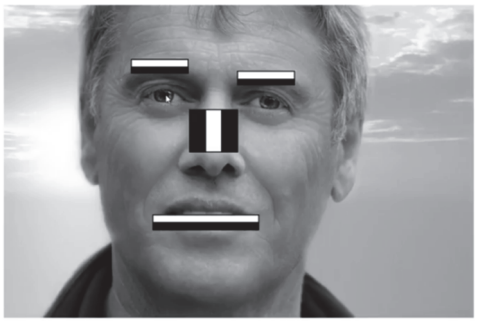

Fig. 3. Haar features located in image.

B. *Integral image*

This subsection presents the concept of Integral image because of its importance in improving the time complexity of Viola-Jones algorithm.

Integral image is a picture where the values of elements of the pixel matrix are changed compared to their previous values. New value of the pixel X is calculated using formula (1).

$$X = p + p_l + p_u + p_{ul} \tag{1}$$

where:

- p – old value of the pixel X,
- p_l – new value of the pixel left from X,
- p_u – new value of the pixel up from X,
- p_{ul} – new value of the pixel up and left from X.

If some of the pixels (left, up or up and left from X) do not exist, then their value is considered to be 0. In Fig. 4 is shown the transformation of the original pixel value matrix into pixel value matrix of the integral picture.

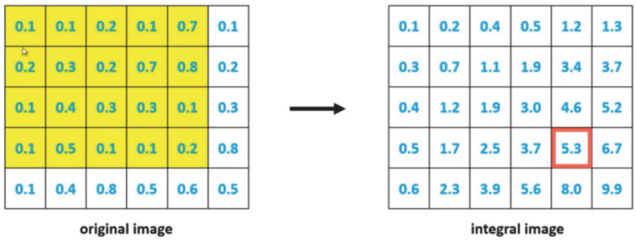

Fig. 4. Transformation of the original pixel value matrix into pixel value matrix of the integral image.

Integral image is used for the calculation of average value of light and dark surfaces of Haar features. Before starting the search for Haar features in a picture, pixel value matrix is transformed into pixel value matrix of the integral image because of the time complexity of the calculation of values of light and dark surfaces. If the original pixel value matrix is used, this time complexity equals *O(n)*, where *n* is the number of pixels in surface. However, if the pixel value matrix of the integral image is used, this time complexity equals *O(1)* [5]. Having in mind that the number of Haar features needed to be detected is large, the time which is lost during the mentioned transformation is incomparable to the time which is saved during the process of finding Haar features [5]. In Fig. 5 is shown pixel value matrix of an integral picture. Calculating the surface value below the orange rectangle is done using formula (2).

$$V = d - c - b + a \tag{2}$$

where:

- V – surface value below the orange rectangle,
- d – value of pixel d) in Fig. 5,
- c – value of pixel c) in Fig. 5,
- b – value of pixel b) in Fig. 5,
- a – value of pixel a) in Fig. 5.

Fig. 5. Calculating surface value in integral image.

C. Algorithm for training the classifier

In this subsection the preparation process necessary for the object detection algorithm to be able to work properly is presented. The main part of the preparation process is training the classifier used by the object detection algorithm.

Classifier is the most important file necessary for the object detection algorithm to be able to function properly. During object detection process, almost in every step, something is read from the classifier. The main components of the classifier are:

- Detection frame,
- Haar features which are distinctive for human face,
- Groups of Haar features (stages) with defined priorities and thresholds.

During the object detection process, algorithm does not check for faces once in the whole picture, but many times in many different parts of the picture. Detection frame is a rectangle which slides over the picture while algorithm searches for objects in the part of the picture it encompasses. The size of the detection frame determines the size of the pictures which will be used for the learning data set.

Before starting with classifier training, it is necessary to provide a learning data set. In this case, a learning data set is a set of pictures with exact size that represent or do not represent the desired object. The resolution of pictures should be very small, usually 24x24 or 48x48 pixels.

The goal of this algorithm is to find Haar features which are distinct for human face, group them and put them into the classifier.

At the start of this algorithm all pictures from the learning data set are granted the same weight value. Weight value of a picture is a number and is essential for the algorithm (usage is explained in the following steps). Then, next steps are followed:

1. For each Haar feature that exists (basic and derived, taking into consideration the size of a detection frame) error weight is calculated. Error weight is the sum of weight values of all pictures from the learning data set that are misclassified by current Haar feature. Misclassified pictures are those pictures that represent desired object but do not contain current Haar feature or those pictures that do not represent desired object but contain current Haar feature.
2. Haar feature with the smallest error weight is selected and added to the set as a multiplication of priority coefficient and Haar feature existence function.
3. Priority coefficient is changed.
4. Weight of pictures successfully classified by current Haar feature is decreased.
5. Weight of pictures misclassified by current Haar feature is decreased.
6. If the desired number of selected Haar features is not achieved, return to step 1.

After executing those steps, the result will be the set of multiplications $a_x * h_x()$, where a_x is a priority coefficient for Haar feature x and $h_x()$ is an existential function of Haar feature x.

The final part of this algorithm is the grouping of Haar features in stages. Stages are prioritized. Ones with a higher priority contain multiplications $a_x * h_x()$ with a higher priority coefficient. Every stage has its stage value which is the sum of all multiplications $a_x * h_x()$ that stage contains. Stages with a higher priority contain less multiplications than stages with a lower priority. For example, the most prioritized stage contains 5 multiplications, whereas some less prioritized stages can contain around 1000 multiplications.

D. Algorithm for object detection

This subsection presents the Viola-Jones algorithm for object detection used in this application for detecting, tracking and recovery from failure during tracking the human face.

For the object detection algorithm to start executing, the picture must be black and white. Also, it is necessary that the classifier trained for finding the wanted objects exists. Classifier is defined by the set of data which is obtained as a result of Viola-Jones algorithm for training classifier. It exists as a file where all of the information required for regular functioning of the algorithm are written. Besides the classifier, it is necessary to provide values for two more parameters: scaleFactor and minNeighbourNum.

Algorithm is executed in the following way. First, the detection frame is set in the top-left corner of the picture. The detection frame is a rectangle whose size is defined in the classifier. After that, in the part of the picture which the detection frame encompasses, starts the search for Haar features for the current stage defined in the classifier. In the classifier, Haar features are grouped in stages according to the priority and the checking for the existence of Haar features is done in order from high to low priority stages. The checking is done by calculating the stage value with the given formula (3).

$$V_f = a_1 * h_1 + a_2 * h_2 + a_3 * h_3 + \cdots + a_m * h_m \quad (3)$$

where:

- a_x – priority of Haar feature x.
- h_x – existential function of Haar feature x. If Haar feature x exists in the picture then this function returns 1. Otherwise it returns 0.
- m – total number of Haar features in the stage.

$$V_f \geq (a_1 + a_2 + \cdots + a_m)/2 \quad (4)$$

If the expression given in formula (4) is correct, the current stage is successfully passed and the algorithm proceeds with checking the next stage. Otherwise, the stage is not passed successfully and the part of the picture the detection frame encompasses does not contain the searched object. If all of the stages are passed successfully then the part of the picture

detection frame encompasses contains the searched object. After the checking, the detection frame is moved some pixels to the right and the checking is done again for the part of the picture below the detection frame. When the detection frame touches the right edge of the picture and the checking is done, then it is moved to the left edge and some pixels down and the whole process is repeated. When the detection frame touches the most bottom-right pixel and the checking is done, the size of the whole picture is reduced in accordance with parameter scaleFactor, detection frame is put in the most top-left corner and the whole process is repeated. The reduction of size of the picture is done in order to reduce the size of the objects in the picture relative to the detection frame so that the bigger objects could fit into the detection frame. The picture size is reduced as long as it is big enough for the detection frame to fit inside. After that, all found objects that have the number of neighbor objects smaller than *minNeighborNum* are removed from the set of found objects. This is done because if the object does not have the minimal number of neighbors, it is considered as an error (Fig. 6).

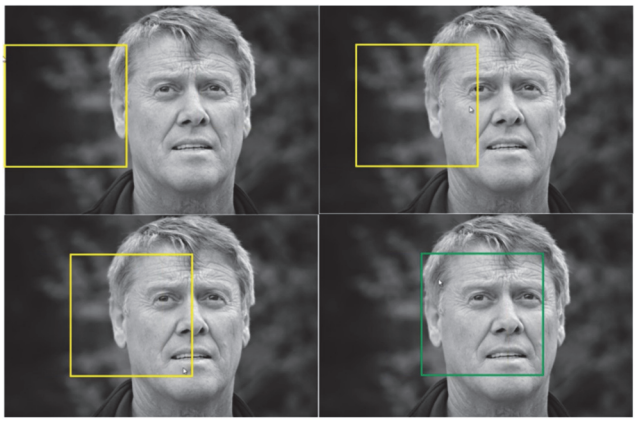

Fig. 6. Moving the detection frame.

III. Graphical User Interface

Graphical user interface is written in Java programming language using Swing programming library. GUI consists of two main components – a command panel (left part of Fig. 7) and a preview panel (right part of Fig. 7). Application is used in the following way: First, user chooses whether the video content should be obtained from a file or web camera. Then, the preview starts and the video content is being showed in the preview panel. During the preview, user can pause/resume the preview. Also, user can click on the preview panel and start tracking the face depending on whether the user clicked on the face or not. Furthermore, user can then click on the face which is being tracked and stop the tracking of that face. User is able to change the tracking mark (rectangle, circle, etc.) and change the resolution of previewed content together with the resolution of GUI. A status message label is located at the bottom of the command panel. All application notifications are written here (e.g. “New Face Found”, “Old Face Removed”, “Preview Paused”, etc.). Preview can be stopped at any time by clicking on the STOP button in the command panel. If video content is obtained from a file then the preview stops when it comes to the end of the video content.

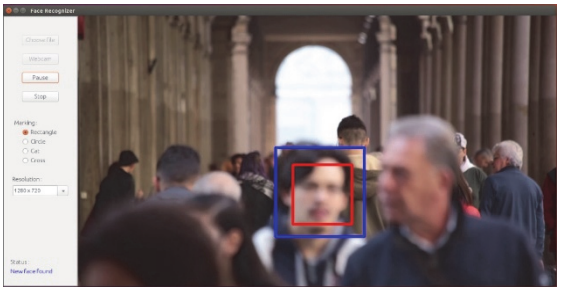

Fig. 7. Graphical user interface.

IV. Architecture of the Realized System

All threads of the system are written in Java programming language. Frame processing executed in those threads is written in C++ programming language. Because of this, JNI and native methods are used during the thread execution. The application starts with the initialization of the part of the application responsible for frame processing written in C++ programming language. The main part of this initialization is loading the classifier used for detecting faces with yhr Viola-Jones algorithm.

Every time the user choses the option for preview of the video content from a file or web camera, new threads are created. If user has chosen the preview from a file, then 3 threads are created. The first thread reads frames from the video content [6]. The second one changes the size of the frame if necessary [7]. The third one does the main processing of the frame - detecting, tracking and recovery from failure during tracking the face in the frame. After those 3 actions, the third thread draws the frame on a graphical user panel [8]. Threads are communicating using two buffers. All three threads are synchronized using standard producer/consumer synchronization [9]. Using threads has led to an improvement in performances by 30% relative to the architecture of the application where mentioned 3 actions are executed in a single thread. The total wait time between showing two consecutive frames equals the longest of the read time, resize time and main processing time. Without using threads and buffers, the total wait time would be equal to the sum of the mentioned times.

If a user has chosen the preview from web camera, only one thread is created. First, the frame is read, then it is resized if necessary [6], [7]. After that the frame is processed and drawn on a graphical user panel [8]. If the application architecture with 3 threads (as mentioned before) is used in combination with the preview from web camera, it would cause latency in showing real-time picture.

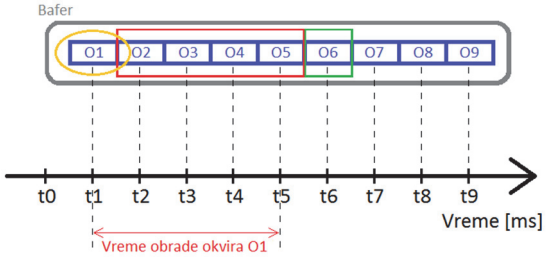

Fig. 8. Latency while using web cam.

The example of latency in showing the picture from web camera is shown in Fig.8. Suppose that the architecture is the same as the one mentioned before with 3 threads and that resize time and draw time equal 0 ms. Moments t1, t2, t3, ... , t9 are moments when the frames O1, O2, O3, ... , O9 are read. Moments t0, t1, t2, ... , t8 are moments where starts the reading of frames O1, O2, O3, ... , O9. In the moment t1 starts the processing of the frame O1. During the frame processing, new frames are read from the web camera. When the processing is finished, next frame to be processed should be O6 because it shows the picture from the web camera right after the last frame is drawn. O6 is the current, real-time frame. However, next frame to be processed is the first frame from the buffer – the frame O2, which does not show the current real-time picture but the picture from the past. The latency is proportionate to the buffer size. For buffer with size X and longest read time T_{read}, maximal latency is defined with formula (5).

$$T_{Latency_{Max}} = X * T_{read} \tag{5}$$

The higher the difference between processing time and read time the faster the buffer will fill and get to the state where latency is maximal.

V. Main processing of the frame in the application

When user clicks on the graphical panel, the coordinates of the pixel where he clicked are inserted into coordinates buffer. Main processing of the frame consists of 3 actions. The first is face detection. For every coordinate from the coordinates buffer it is checked whether a face exists in the area around coordinates or not. The checking is done using the Viola-Jones algorithm. If the face exists and is not already inside the detected faces buffer, the position of that face is inserted into the detected faces buffer as a square defined by its top-left coordinate and the length of its side. The second action is face tracking. If the face exists but is already inside the detected faces buffer, then it is removed from the detected faces buffer. Tracking is done for every square from the detected faces buffer in parts of the frame determined by squares which are made by increasing the size of squares from the detected faces buffer by 50%. It is supposed that the face will not move enough to be outside the enlarged square. If the face is detected, the old square from the detected faces buffer is replaced with the new one which corresponds to the new position of the face. In case the face is not detected, it is marked for recovery from tracking failure.

Recovery starts from processing of the next frame. It is the third of the actions. During this action, application is trying to find again the face which was tracked in the past but not found in some of the previous frames. This action lasts no more than 30 frames after the face is marked for recovery. In every of those frames, the application is trying to find the face in those parts of the frame where it should have been if it had continued to move in the same manner as it did before failure in tracking. The manner of movement is obtained by calculating average difference between two consecutive positions of the face and average difference between two consecutive sizes of the face, taking into consideration only 15 last frames. Every detected face has its manner of movement calculated and available in every moment while the application is working. The face is searched in the part of the picture below the square whose top-left point is moved by X_{offset} and Y_{offset} relative to the top-left point of the square which represents the last known position of the face. Values X_{offset} and Y_{offset} are calculated using formulas (6) and (7).

$$X_{offset} = N * O_x \tag{6}$$

$$Y_{offset} = N * O_y \tag{7}$$

where:

- N – ordinal number of the frame relative to the frame when the face is marked for recovery,
- O_x – average difference between x coordinates
- O_y – average difference between y coordinates

Size of the square below which the face is being found is calculated using formula (8).

$$S_{new} = 1.5 * S_{old} + N * O_{size} \tag{8}$$

where:

- S_{old} – size of the square representing the last known position of the face,
- N – ordinal number of the frame relative to the frame when the face is marked for recovery,
- O_{size} – average difference between sizes.

While processing every frame, the face is being searched in the part of the picture where it was found last time. If the face is found again, it remains in face detection buffer and it is unmarked for recovery. However, if it is not found 30 frames after it is marked, then it is considered forever lost and removed from the face detection buffer.

VI. Performances of the realized system

The total time passed between showing two consecutive frames depends on the following 4 time components: frame reading time, frame resizing time, frame processing time and frame drawing time. The duration of those 3 time components depends on the characteristics of the PC the application is running on. During these tests, these were the characteristics of the pc:

- Processor : Intel i7-6700 3.40GHz x 8.
- GPU : Intel HD Graphics 530.

Tests were made in the office using 720p webcam, and video files with different resolutions.

Frame reading time mostly depends on whether the video content is read from a file or from web camera. In the first case, this time mostly depends on the frame size, whereas in the second case the frame size has a much smaller effect on the length of frame reading time. If the frames are read from web camera, dependencies are a little bit different. The length of the reading times of two frames with different content can be different by even 1000%. In Fig. 9 are shown two frames with different content. The reading time of the left frame is about 40ms, but the reading time of the right frame is 4ms.

Fig. 9. Reading time of the left frame is longer than reading time of the right frame.

The input source does not affect frame resizing time. Frame content has a small effect on this time component. The biggest effect on frame resizing time has the size of the frame.

Frame processing time does not depend on the input source. The size of the frame also has a little effect on it. The biggest effect on frame processing time has the content of the frame. Processing time of the frames whose content does not contain high contrast (sky, darkness, light, etc.) is very short because the Viola-Jones algorithm for object detecting is able to quickly determine that those frames do not contain human faces. However, frames that contain whole or partial faces are processed considerably slower. In Fig. 10 are shown two frames with very different content. Processing time of the left frame is 45 ms, whereas processing time of the right frame is 21 ms. Also, the number of faces that are tracked in one moment is directly proportionate to the frame processing time.

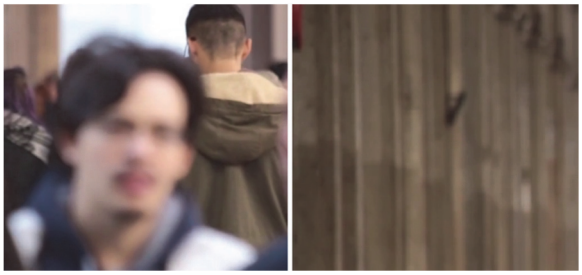

Fig. 10. Processing time of the left frame is longer than processing time of the right frame.

Frame drawing time is usually very small. It depends on the size of the frame and also the content of the frame has a little effect on it.

VII. Conclusion

In this paper is shown the implementation of an application which uses the Viola-Jones algorithm for detecting objects in the image and an already trained classifier. The algorithm is implemented in one of the functions from OpenCV library which proved very useful because of its many features considering image processing.

For a video frame to be processed as fast as possible, frame processing is written in C++ programming language. Because GUI of the application is written in Java programming language, Java Native Interface is used. To make the processing faster, multithreading was used whenever it was possible. With PC characteristics given in the previous section, application is able to preview video content with 32 fps while tracking the object, which was its goal – real-time object tracking.

Using the Viola-Jones algorithm for object detection, the best improvement in performances can be achieved by improving the hardware which runs the application. Because of this, it is recommended to use faster algorithms based on neural networks while developing new applications with the same goal as this one [10]. Google has created a library for machine learning called TensorFlow. Inside it, there are implementations of many algorithms based on neural networks. One of the faster and inexpensive ones is “Faster Region Convolutional Neural Networks (Faster RCNN)” which is able to classify multiple objects from the different object class present in the picture in real time.

References

[1] OpenCV development team, “Cascade Classifier”, 11.09.2008, https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html, accessed July 2018.

[2] Jan Masek, Radim Burget, Jan Karasek, Vaclav Uher, Selda Güney, “Evolutionary improved object detector for ultrasound images”, 2013 36th International Conference on Telecommunications and Signal Processing (TSP), July 2013, Rome, Italy.

[3] Viola, P., Jones, M.J. & Snow, D., “Detecting Pedestrians Using Patterns of Motion and Appearance”, Int J Comput Vision (2005) 63: 153. https://doi.org/10.1007/s11263-005-6644-8.

[4] Liu Yun, Zhang Peng, An Automatic Hand Gesture Recognition System Based on Viola-Jones Method and SVMs, 2009 Second International Workshop on Computer Science and Engineering, October 2009, Qingdao, China.

[5] Paul Viola, Michael Jones, Robust Real-time Object Detection, Second International Workshop on Statistical and Computional Theories of Vision – Modeling, Learning, Computing and Sampling, 2001, Vancouver, Canada.

[6] OpenCV development team, „Reading and Writing Images and Video“,11.09.2008, https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html, accessed July 2018.

[7] OpenCV development team, „Geometric Image Transformations“, 11.09.2008, https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html, accessed July 2018.

[8] OpenCV development team, „Drawing functions“, 11.09.2008, https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html, accessed July 2018.

[9] Radivojević Zaharije, Ikodinović Igor, Jovanović Zoran, „Konkurentno i distribuirano programiranje“, prvo izdanje, Akademska misao, Beograd 2008.

[10] Arthur Ouaknine, „Review of Deep Learning Algorithms for Object Detection“, 05.02.2018., https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852, accessed July 2018.

[11] D. Đorić, S. Crnobrnja, M. Punt and Mile Davidović, “Implementation of an Application for Real-Time Video Face Tracking,” (in Serbian) *2018 26th Telecommunications Forum (TELFOR)*, Belgrade, 2018, pp. 1-4.