ISSN: 2350-0328

International Journal of AdvancedResearch in Science, Engineering and Technology

Vol. 11, Issue 5, May 2024

# Face detection algorithm using OpenCV library

**Sabirov B I**

Assistant of the Department of Information Security, Urganch Branch, Tashkent University of Information Technologies named after Muhammad al-Khorazmi Khorazm, Uzbekistan

**ABSTRACT**: Today, information security is an important problem in information systems, especially authentication issues in information systems. These problems can be solved by means of high security such as biometrics, two-factor authentication methods, fido2 standard. The importance of an automated integrated system has proven to be highly effective in applications such as surveillance systems and privacy systems. The peculiarity of each system is the use of open source code openCV software and an efficient algorithm to target users in the presence of various vulnerabilities in password input and face detection errors in the smart door lock.

**KEY WORDS**: Integrated system, OpenCV, face detection, Authentication, face shape, Computer Vision, digital images.

## I.INTRODUCTION

Each user should use an authentication system when using various information systems, which should be easy to use, functional and legal, and this is of great importance in ensuring security. Authentication, which is based on the user's registered and personal information, helps to provide secure access to the system. One of the existing problems with passwords is that passwords are not cryptographic passwords, the integration of two-factor authentication systems in password systems, the correctness of information provided for user identification, and others can be cited as an example. Currently, the integration of various systems and services of campaigns and organizations, their automation, is very important. Efficient use of the user's time and resources in the use of the system ensures more efficient and better functioning of the system. It provides protection of personal data and systems against their nature and harmful effects. Therefore, it is necessary to first use passwords from the data entered by users, in which instructions should be given to users about the use of complex passwords or tokens, and each user should use their own biometric authentication methods as passwords. The use of automatic and interactive door lock systems in smart door lock systems provides a combination of security systems, such smart door locks can be installed in many workplaces, smart homes, office buildings, automatic doors, which increases the security of this system, in which:
Smart door locks have administrative security systems to control door opening and closing processes;
Door locks can have monitoring functions to control entry and exit processes of the building and collect information about these processes;
Door lock systems can have automatic security systems based on the system being stopped or opened.
Security problems in information technology, especially the increase in authentication time in information systems, are currently one of the biggest problems. Face recognition is considered as a new and effective approach to the previous authentication methods.

## II. SIGNIFICANCE OF THE SYSTEM

Face recognition provides accurate confirmation of the user's face by identifying the biometric data of the person. This method helps to make its authentication process efficient and easy for the user, because users are not required to memorize passwords or other authentication information.
The process of face detection is based on the unique features of the user's face, such as the eye socket, the normal shape of the eyes, the boundaries of the sun set, etc. The user's face is read by the scanner and the facial information is stored in the database. Then, a scanner is automatically launched to re-identify the user's face to log in.
Widespread use of facial recognition provides an intuitive and reliable authentication method for users. Also, this method increases the convenience of remote processing, as users can authenticate themselves even from a long distance.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 5, May 2024

Such important authentication approaches are part of the proposed approaches to increase security in information technology. Facial recognition systems provide convenient and secure authentication for users and provide a high level of security.

## III. METHODOLOGY

We'll look at implementing this security programmatically using the OpenCV library.
For this, you must first install the OpenCV library.
Write code using the OpenCV library to turn on the camera and capture the frame. A frame is retrieved using the .read () function.

```
import cv2
# Camera object
cap = cv2.VideoCapture(0)
# Get a frame
ret, frame = cap.read()
# Check if the frame has been taken
if ret:
print ("Frame taken successfully.")
otherwise:
print ("Frame failed.")
# Close the camera
cap.release()
```

The OpenCV library contains classifiers for face detection. For example, the classifier can be downloaded via the cv2.CascadeClassifier () function.

```
import cv2
# Turn on the camera
cap = cv2.VideoCapture(0)
# Load face classifier
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
while True:
# Get a frame
ret, frame = cap.read()
# Convert to grayscale image for processing
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Face detection
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
# Show faces
for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
# Output the result
cv2.imshow('Face Detection', frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
break
# Support for spacers and window closing
cap.release()
cv2.destroyAllWindows ()
```

After the frame is captured, we use the detectMultiScale() method to detect the faces. Through this method, we can detect faces on the frame.

```
# Face detection
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
```

in this case, the gary variable works depending on the changes in the image based on the data on two parameters. scaleFactor=1.1 is used to scale by how many times per step.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 5, May 2024

minNeighbors=5 is the number of lines needed to detect a face, minSize=(30, 30) is the minimum width of a face to be detected.
We use the rectangle() function to display the detected faces on the frame. This function draws a line in the frame at right angles through the coordinates, and also fills it with color.

```
# Show faces
for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

In this code, the faces list displays the faces. For each face, the rectangle() function takes the coordinates of the face and draws a line on its frame. (x, y) is the start point of the face, (x+w, y+h) is the end point of the face, (0, 255, 0) is the line color (green), and 2 is the line thickness.
To output the output frame and display it along with the faces

```
# Output the result
cv2.imshow('Face Detection', frame)
```

This string is used to output the frame variable. The variable 'detect faces' defines the title of the window. The cv2.imshow() function displays the frame on the screen. Then, a frame is displayed and it is expected that some button is pressed to close the window.

```
mport cv2
# Camera object
cap = cv2.VideoCapture(0)
# Load face classifier
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
while True:
# Get a frame
ret, frame = cap.read()
# Convert to grayscale image for processing
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Face detection
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
# Show faces
for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
# Output the result
cv2.imshow('Face Detection', frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
break
# Support for spacers and window closing
cap.release()
cv2.destroyAllWindows ()
```

## IV. EXPERIMENTAL RESULTS

We will learn to create a human face recognition program with OpenCV and Python. For this, we need the OpenCV library and an image of a human face.
To install the library, enter the following command in the terminal

```
pip install opencv-python
```

After the library is installed, download the desired image.

```
import cv2
face_shape = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

We import the Opencv library through cv2 and load the information about the structure of the face in our library into the variable named "face_shape". Based on the parameters we entered, our program can now detect a human face.

```
image= cv2.imread("Jobs.jpg")
frame_color = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

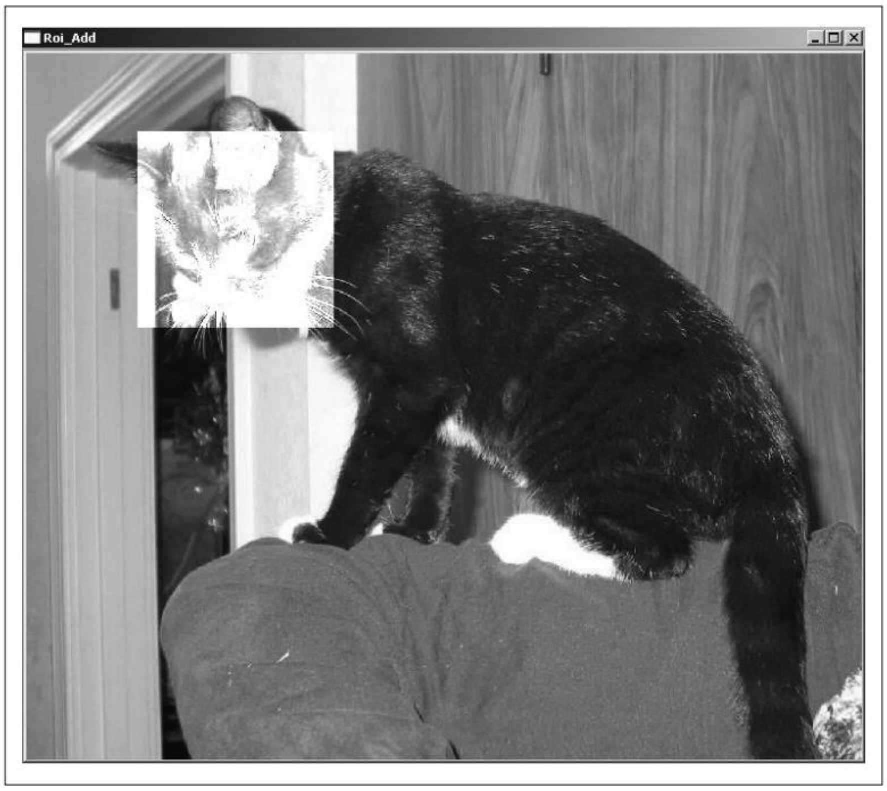

Figure 1. Use case of face detection and tagging in Opencv library.

It displays the uploaded image through the colors "BGR" and "GRAY". "BGR" stands for "Blue, Green, Red" and "GRAY" stands for "Green, Red, Azure, Yellow".

```
Faces = face_shape.\detectMultiScale(image, 1.1, 19)
for (x,y,w,h) in Faces:
cv2.rectangle(image, (x,y),(x+w,y+h), (0,255,0),2)
```

We put the data of the human face described in the step into the variable named "Faces" and draw a frame outside the faces detected by the program through the for loop.

```
import cv2
face_shape = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
image= cv2.imread("Jobs.jpg")
frame_color = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
Faces = face_shape.\
detectMultiScale(image, 1.1, 19)
for (x,y,w,h) in Faces:
cv2.rectangle(image, (x,y),
(x+w,y+h), (0,255,0),2)
print(x,y,w,h)
cv2.imshow('result', image)
cv2.waitKey()
```

Video capture by the camera scans the input frame and examines it for real-time face detection.

```
import cv2
# Real-time video capture
video_camera = cv2.VideoCapture(0)
# Loop through real time to detect faces
while True:
ret, frame = video_camera.read()
# Classifier for face detection
```

```
face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
# Detect faces in the frame
faces = face_classifier.detectMultiScale(frame, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
# Draw defined faces
for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
# Display frame
cv2.imshow('Face Detection', frame)
# Keyboard key tracking for exit
if cv2.waitKey(1) & 0xFF == ord('q'):
break
# Clean up the screen and reduce loading
video_camera.release()
cv2.destroyAllWindows()
```

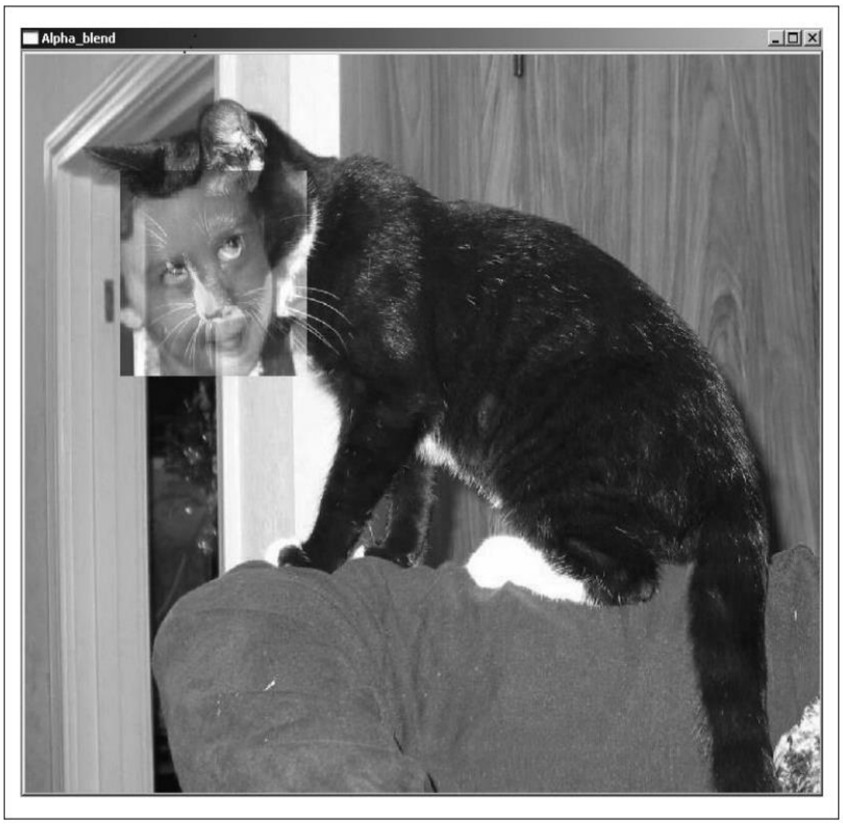

Figure 2. Face detection in the Opencv library and finding a similar facial expression from the database.

The use of cameras in identification processes is considered to be a cheap and convenient tool for monitoring, and the purpose of such systems is to offer the user an effective algorithm created using the open source software OpenCV. The experimental results show that the proposed system is more efficient, consumes less energy and is more cost-effective.

## V. CONCLUSION AND FUTURE WORK

The algorithm for detecting faces in the frame has been successfully created and implemented. I believe that this algorithm is effective and reliable and will serve as an effective solution for face detection for our users.
During work:

- To increase the effectiveness and reliability of face detection, it is considered appropriate to apply additional guides and filters;
- It is necessary to create and maintain a database to support facial recognition;
- We will have to consider the issues of optimization of face detection algorithm to increase security and speed.

So, using the OpenCV library, the experience of implementing the face detection algorithm in the frame was partially reviewed. After successful changes and optimizations in this algorithm, I will suggest to our users to use it in surveillance systems to accurately compare the facial appearance with the base and increase security.

## REFERENCES

[1] Learning OpenCV: Computer Vision with the OpenCV Library by Gary Bradski and Adrian Kaehler
[2] OpenCV Essentials by Oscar Deniz Suarez, Mª del Milagro Fernandez Carrobles, Noelia Vallez Enano, and Gloria Bueno Garcia
[3] Mastering OpenCV 4 with Python by Alberto Fernandez Villan –
[4] Programming Computer Vision with Python: Tools and algorithms for analyzing images by Jan Erik Solem
[5] A. Rosenfeld and E. Johnston, "Angle detection on digital curves," IEEE Transactions on Computers 22 (1973): 875–878.
[6] H. J. Antonisse, "Image segmentation in pyramids," Computer Graphics and Image Processing 19 (1982): 367–383.
[7] C. L. Bajaj, V. Pascucci, and D. R. Schikore, "Th e contour spectrum," Proceedings of IEEE Visualization 1997 (pp. 167–173), 1997.
[8] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," Pattern Recognition 13 (1981): 111–122.
[9] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," Proceedings of the Ninth European Conference on Computer Vision (pp. 404–417), May 2006.
[10] S. S. Beauchemin and J. L. Barron, "Th e computation of optical fl ow," ACM Computing Surveys 27 (1995): 433–466.
[11] S. Belongie, J. Malik, and J. Puzicha, "Shape context: A new descriptor for shape matching and object recognition," NIPS 2000, Computer Vision Group, University of California, Berkeley, 2000.
[12] G. Borgefors, "Distance transformations in digital images," Computer Vision, Graphics and Image Processing 34 (1986): 344–371.
[13] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," Proceedings of the Neural Information Processing Systems Conference (vol. 19, pp. 304–310), 2007.