Zero : Jurnal Sains, Matematika, dan Terapan
E-ISSN : 2580-5754
P-ISSN : 2580-569X
Vol. 6, No. 2, Desember 2022
pp. 147-156

Eye Aspect Ratio Adjustment Detection for Strong Blinking Sleepiness Based on Facial Landmarks with Eye-Blink Dataset.

Eswin Syahputra[1] , Irpan Nursukmi[2] , Sony Putra[3] , Bayu Sukma Sani[4] , Rian Farta Wijaya[5]
[1,2,3,4,5] Universitas Pembangunan Panca Budi, Medan, Sumatera utara, Indonesia

Article Info

Article history:

Keywords:

Blink Detections, Eye Aspect Ratio, Eye Blink, Facial Landmarks, Dlib

ABSTRACT

Blink detection is an important technique in a variety of settings, including facial motion analysis and signal processing. However, automatic blink detection is challenging due to its blink rate. This paper proposes a real-time method for detecting eye blinks in a video series. The method is based on automatic facial landmark detection trained on real-world datasets and demonstrates robustness against various environmental factors, including lighting conditions, facial emotions, and head position. The proposed algorithm calculates the position of facial landmarks, extracts scalar values using the Eye Aspect Ratio (EAR), and characterises eye proximity in each frame. For each video frame, the proposed method calculates the location of the facial landmark and extracts the vertical distance between the eyelids using the position of the facial landmark. Blinks are detected by using the EAR threshold value and recognising the pattern of EAR values in a short temporal window. According to the results from a common data set, it is shown that the proposed approach is more efficient than state-of-the-art techniques.

Corresponding Author:

Eswin Syahputra,
Master of Information Technology
Universitas Pembangunan Panca Budi, Medan, Sumatera utara, Indonesia
Email: eswin.syahputra@gmail.com

1. INTRODUCTION

Eye blink detection technology is very important and has been applied in various fields such as communication between disabled people and computers (Królak & Strumiłło, 2012), drowsiness detection (Rahman, Sirshar & Khan, 2016), computer vision syndrome (Al Tawil et al., 2020; Drutarovsky & Fogelton, 2015), anti-spoofing protection in face recognition systems (Pan et al., 2007), and cognitive load (Wilson, 2002).

Driver drowsiness detection is a technology used to detect when a driver is tired or sleepy while driving. This can be done through various methods, including monitoring the driver's eye movements, head position, and facial expressions, as well as monitoring vehicle movement and speed. Some systems use cameras and computer vision algorithms to track the driver's face and eyes, while others use sensors to monitor the driver's pulse and other biometric data.

When drowsiness is detected, the system can perform various actions, such as sounding an alarm, vibrating the seat, or even slowing down or pulling over the vehicle. The purpose of driver drowsiness detection is to prevent accidents caused by driver fatigue and to improve road safety.

In the intelligent system in this paper, a system is built to detect a person's drowsiness by checking how long the eyes are closed. If the eyes have been closed for a specified time that exceeds a certain threshold

value that can result in an accident, the system will alert the user by sounding an alarm. All scripts use Python programming language.

An individual's normal vision is characterised by the presence of spontaneous eye blinks at a certain frequency. The following elements have an impact on blinking, including the condition of the eyelids, the condition of the eye, the presence of disease, contact lenses, psychological state, the surrounding environment, medications, and other stimuli. Blink frequency ranges between 6 and 30 times per minute (Rosenfield, 2011). Furthermore, the term 'blink' refers to the rapid closing and reopening of the eyelids, which typically lasts between 100 and 400 ms. Reflex blinks occur significantly faster than spontaneous blinks, which occur significantly less frequently. Blink frequency and length can be affected by relative humidity, temperature, light, fatigue, illness, and physical activity. Real-time facial landmark detectors (Čech et al., 2016; Dong et al., 2018) are available that capture most of the distinguishing features of human facial images, including eye and eyelid corners.

One person's eye size does not match another person's eyes. For example, one has big eyes but the other has small eyes. They do not have the same eye or height values, as expected. When a person with small eyes closes his eyes, he may appear to have the same eye height as the person with large eyes. This problem will affect the experimental results. Therefore, we propose a simple yet effective technique to detect eye blinks using a newly developed facial landmark detector with Eye Aspect Ratio (EAR).

2. LITERATURE REVIEW

a. OpenCV

OpenCV (Open Source Computer Vision Library) is an open source Computer Vision and machine learning software library. It was developed by Intel in 1999 and is now maintained by a community of developers.

OpenCV provides a wide range of functionality for image and video processing, including image filtering, feature detection and description, object detection, and more. OpenCV also has a large number of trained machine learning models for tasks such as object detection and face recognition that can be easily integrated into projects.

OpenCV provides interfaces for C++, Python, and Java, making it easy to integrate with other programming languages and platforms. It also supports a wide range of platforms, including Windows, Linux, and macOS.

OpenCV is widely used in various applications, including:

1. Computer Vision
2. Image processing
3. Machine learning
4. Robotics
5. Augmented Reality
6. Medical imaging
7. Automotive
8. Surveillance system
9. and many more.

OpenCV is a powerful library with many ready-to-use functions and modules that can be used for various computer vision tasks.

b. Python

Python is a high-level, interpreted, general-purpose programming language. It was created by Guido van Rossum in the late 1980s and first released in 1991. Python is designed to be easy to read and write, and is often used as a scripting or automation language. It is also commonly used for web development, data analysis, artificial intelligence, and scientific computing.

One of the main advantages of Python is its simplicity, which makes it easy to learn and understand. It also has a large and active community, which has created a large collection of libraries and modules that can be used to perform a wide variety of tasks.

Python also supports several programming paradigms, including object-oriented, functional, and procedural programming. This makes it a versatile language that can be used for a variety of

projects, ranging from small scripts to large enterprise applications. It is also supported by many operating systems such as Windows, Linux, MacOS, etc. It is open-source and free to use.

c. Computer Vision

Computer Vision is a field of study that aims to enable computers to understand and interpret visual information from the world, such as images and videos. It involves developing algorithms and systems that can process, analyse and understand visual data using techniques from computer science, mathematics and physics. Computer Vision has a wide range of applications, including image and video analysis, object detection and recognition, medical imaging, robotics, self-driving cars, and more. Some of the main techniques used in Computer Vision include:

1. Image processing: Techniques for manipulating and analysing images, such as filtering, edge detection and feature extraction.
2. Object recognition: Techniques for identifying and classifying objects in images or videos, such as using machine learning algorithms to train models to recognise specific objects.
3. Scene understanding: Techniques for understanding the context and layout of a scene, such as identifying the objects present, their positions, and their interactions.
4. Motion analysis: Techniques for analysing the motion of objects in a video, such as tracking objects over time and estimating their 3D motion.

Computer Vision is a rapidly growing field, fuelled by the increasing availability of high-quality image and video data, as well as the growth of advanced machine learning techniques. With the help of OpenCV and other libraries, computer vision is becoming more accessible to developers, and is being applied in more areas.

3. RESEARCH METHODE

The method used for driver drowsiness detection is based on Eye Aspect Ratio (EAR) calculation. The basic idea is that when a person is sleepy or asleep, his or her eyes will be closed for a longer time. The system uses computer vision and machine learning techniques to track a person's eye movements and determine if their eyes have been closed for a long time.

The programme uses the dlib module and eye aspect ratio calculation to detect sleepiness in a person using a webcam video feed as input. In Figure 1, the programme captures video frames from the webcam using the OpenCV library module and uses dlib's facial landmark detector to find eyes in the webcam feed. It then calculates the aspect ratio of the eyes by finding the Euclidean distance between the eye landmarks. This aspect ratio is then compared to a threshold value, if the aspect ratio is less than the threshold it indicates that the eyes are closed and the person is sleepy. Once drowsiness is detected, the programme will trigger an alarm sound to alert the person and remind them to stay awake. Also, the accuracy of the programme may vary based on lighting conditions, camera resolution, and certain facial features of the person.

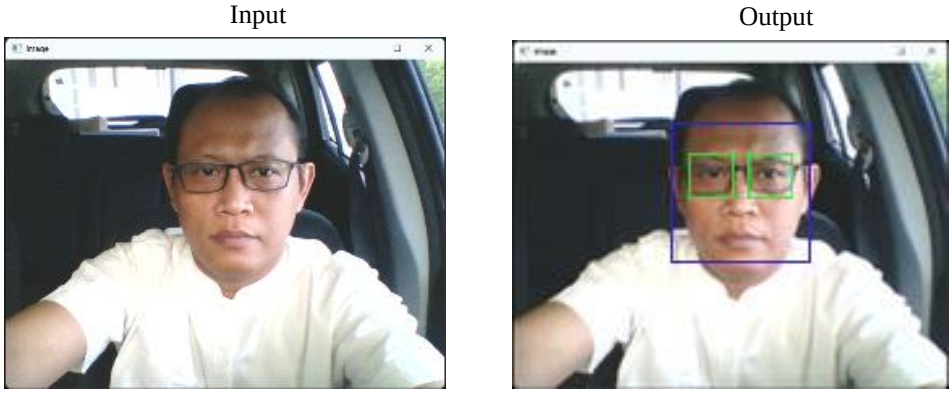

Figure 1. The process of capturing video frames from a webcam (dlib's facial landmark detector).

Some of the modules needed in this intelligent system are: scipy.spatial, imutils, numpy, pygame, time, dlib and cv2. While the dataset used in the case of this intelligent system is CascadeClassifier.

Figure 2 describes the blink detection process. The first step is to split the video into frames. Next, facial landmark features (Wu & Ji, 2019) are implemented with the help of Dlib to detect faces. The detector used here consists of clasic Histogram of Oriented Gradients (HOG) features (Dhiraj & Jain, 2019) along

with a linear classifier. A facial landmark detector is implemented in Dlib (King, 2009) to detect facial features such as eyes, ears, and nose.

After face detection, the eye area is identified using the facial landmark data set. We can identify 68 landmarks (Yin et al., 2020) on the face using this data set. A corresponding index accompanies each landmark. The targeted facial area is identified through the application of the index criteria.

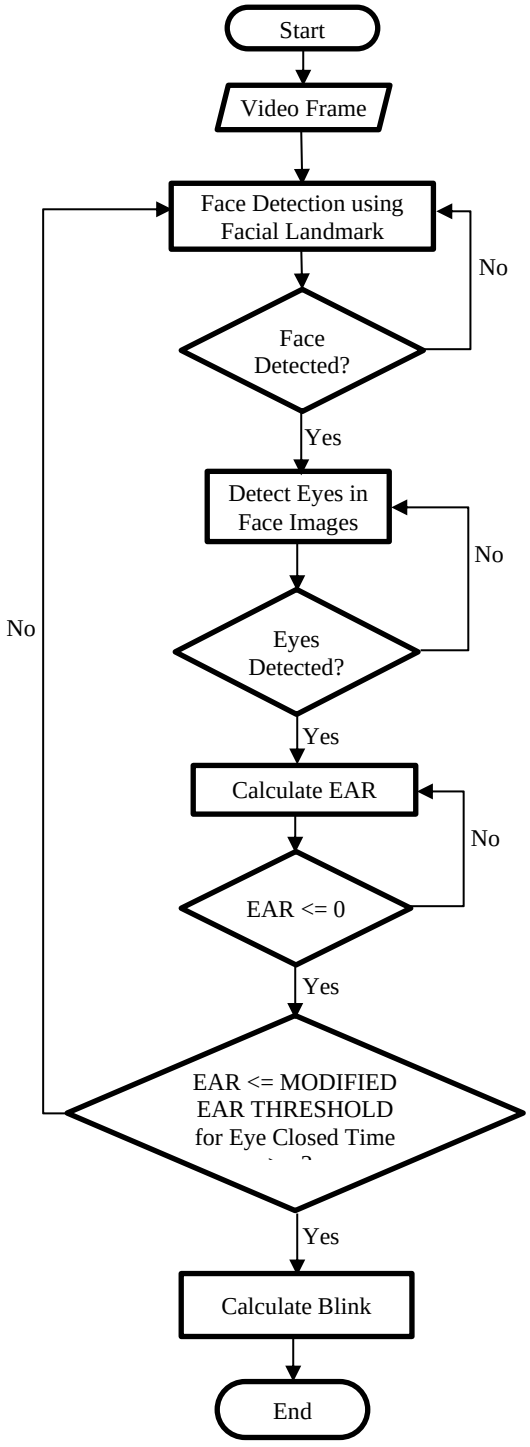

Figure 2. Flowchart of eye blink detection

face_cascade = cv2.CascadeClassifier("haarcascades/haarcascade_frontalface_default.xml")

The cv2.CascadeClassifier() function in OpenCV is used to load a pre-trained cascade classifier model. A cascade classifier is a machine learning model used for object detection. It works by training a series of simpler classifiers, called weak classifiers, and combining them to create stronger classifiers.

A cascade classifier model usually consists of several files, including:

a. **XML file**: This file contains the model's trained parameters, including the features and classifiers that the model uses to detect the object of interest. The XML file is the file that needs to be passed to the cv2.CascadeClassifier() function when loading the model.
b. **Positive sample**: These are images that contain the object of interest. The model is trained on these positive samples to learn how to detect the object.
c. **Negative sample**: These are images that do not contain the object of interest. The model is also trained on these negative samples to learn how to distinguish between object and non-object images.
d. **Dataset**: The training data is a combination of positive and negative samples on which the model is trained.
e. **Classifiers**: Classifiers are simple classifiers that are combined to create the final classifier. Each classifier is trained to detect a particular feature of the object.

When the cv2.CascadeClassifier() function is called, it loads the trained parameters from the XML file and creates a classifier instance that can be used to detect objects in the image. The classifier can be applied to the image by calling the detectMultiScale() method, which returns a rectangular list that includes any objects detected in the image.

It is important to note that cascade classifiers are trained on a specific dataset, and their performance may vary when applied to images that are very different from the images in the training dataset. Therefore, it is important to evaluate the performance of a cascade classifier on a specific dataset before using it in real-world applications.

```
def eye_aspect_ratio(eye):
    A = distance.euclidean(eye[1], eye[5])
    B = distance.euclidean(eye[2], eye[4])
    C = distance.euclidean(eye[0], eye[3])
    ear = (A+B) / (2*C)
    return ear
```

This is a function that calculates the eye aspect ratio (EAR) from a given set of eye facial landmarks. The function takes the ear coordinates of the eye as input and returns the calculated eye aspect ratio.

The function first calculates the distance between the following pairs of markers:

a. The outer edge point of the eye at the top (eye[1]) and the outer edge point of the eye at the bottom (eye[5]).
b. The inner edge point of the eye at the top (eye[2]) and the inner edge point of the eye at the bottom (eye[4]).
c. The inner edge point of the eye at the centre (eye[0]) and the inner edge point of the eye at the outer corner (eye[3]).
d. This distance is represented by the variables A, B, and C.

The function then calculates the eye aspect ratio (EAR) by taking the average of the distance between the top and bottom of the eye (A and B) and dividing it by twice the distance between the centre and outer corner of the eye (C).

$$ear = (A+B) / (2*C)$$

The eye aspect ratio is a measure of the relative height of the eyes. When the eye is open, the distance between the top and bottom of the eye is relatively large compared to the distance between the centre and the outer corner. When the eye is closed, the distance between the top and bottom of the eye decreases, and the aspect ratio of the eye becomes smaller. By comparing the calculated eye aspect ratio with a threshold value, the system can determine whether the eyes are closed or open and thus detect drowsiness.

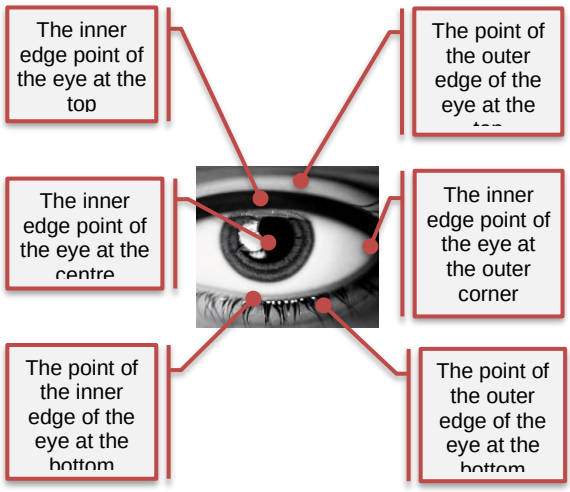

Figure 3. The function first calculates the distance between pairs of markers

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
```

The dlib.get_frontal_face_detector() and dlib.shape_predictor() functions are used to create an instance of the trained facial landmark predictor. This predictor is trained to find 68 specific facial landmarks on the face, including the eyes, nose, and mouth. The predictor takes the image and bounding box of the detected face as input, and returns the facial landmark predictor. The predictor requires a special file with the trained model, in this case it is "shape_predictor_68_face_landmarks.dat".

Once the detector and predictor are loaded, they can be used to detect faces and facial landmarks in the webcam feed. The detector is used to detect faces in the image, and the predictor is used to find facial landmarks on the detected faces. The facial landmarks are then passed to the eye_aspect_ratio(eye) function to calculate the Eye Aspect Ratio (EAR) which is used to detect drowsiness.

It is important to note that the dlib library's face detectors and face landmark predictors are trained on a specific data set and may not work well on images containing faces that are very different from the faces in the training data set. Therefore, it is important to evaluate the performance of the detectors and predictors on a specific data set before using them in real-world applications.

```
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS['left_eye']
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS['right_eye']
```

This code uses the imutils library face_utils.FACIAL_LANDMARKS_IDXS to extract facial marker indices for the left and right eyes. The FACIAL_LANDMARKS_IDXS dictionary contains predefined index ranges for several facial landmarks, including eyes, nose, and mouth. The variables (lStart, lEnd) store the start and end indices of the facial landmarks for the left eye. Similarly, the variables (rStart, rEnd) store the start and end indices of the facial landmarks for the right eye.

This index is then used to extract facial landmarks for the eyes from the shape object returned by the dlib facial landmark detector. Once the facial landmarks are extracted, they are passed to the eye_aspect_ratio (eye) function to calculate the Eye Aspect Ratio (EAR) which is used to detect drowsiness. It should be noted that the coordinates of the facial landmarks are used as input for the eye_aspect_ratio (eye) function, and the position of the facial landmarks is specific to each image, so the script needs to detect the facial landmarks in each video frame to get accurate results.

4. RESULT AND ANALYSIS

Initiate webcam video recording using the cv2.VideoCapture() function of the OpenCV library. The cv2.VideoCapture() function is used to open a connection to a camera or video file and returns a VideoCapture object that can be used to read frames from the camera.

```
video_capture = cv2.VideoCapture(0)
```

The argument passed to the function, "0", indicates that the system default camera (usually the built-in web camera) should be used. If a different camera is to be used, a different number or string with the file path can be passed as an argument.

The VideoCapture object is stored in the video_capturevariable, which can later be used in the script to read frames from the webcam. The script continuously captures new frames from the webcam, processes them, and uses them to detect faces and facial landmarks, as well as calculate the Eye Aspect Ratio (EAR) for drowsiness detection. It is important to note that the video recording runs in a loop, and the script continuously reads the webcam frames, processes them, and uses them for drowsiness detection until the script is stopped.

```
time.sleep(2)
```

A time.sleep() function to pause the execution of the script for 2 seconds before continuing. This is not required, but it is useful to give the camera time to initialise. Some cameras may take a few seconds to start up and be ready to take pictures. By pausing the script for a few seconds, we can ensure that the camera has enough time to start up and be ready to take a picture before the script starts reading frames. This line of code is useful in situations where the camera is not immediately ready to take a picture after the execution of the script, but is not necessary if the camera is fast enough to start and ready to take a picture before the script starts processing the frame. It is important to note that this line of code is optional and can be removed if not required.

```
ret, frame = video_capture.read()
frame = cv2.flip(frame,1)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Reads each frame from the webcam using the VideoCapture.read() method of the video_capture object, which returns a Boolean value indicating whether the frame was read successfully (stored in the 'ret' variable) and the frame itself (stored in the 'frame' variable). It then uses the cv2.flip() function to flip the frame horizontally. This is useful because webcams often capture images that are flipped horizontally. By flipping the frame, you can ensure that the image is orientated correctly.

The code then converts the frame to greyscale using the cv2.cvtColor() function. A greyscale image has only one channel (compared to an RGB image which has three channels) and uses less memory, which can make image processing faster. Also, greyscale images are computationally cheaper than colour images.

In figure 4, this code reads each frame from the camera, flips it horizontally and converts it to grey scale for further processing. It is important to note that, converting to greyscale is optional and can be removed if not required for a particular case, the flipping can also be removed if the image is not flipped.

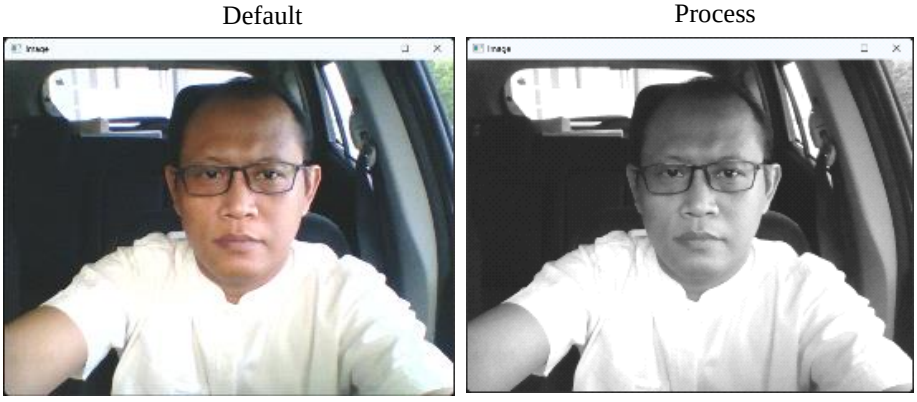

Figure 4 Check the horizontal position of the camera and convert to greyscale

```
faces = detector(grey, 0)
face_rectangle = face_cascade.detectMultiScale(grey, 1.3, 5)
for (x,y,w,h) in face_rectangle:
cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
```

Detector serves to detect facial landmarks on a "grey" grayscale image. The function returns a "face" object containing the coordinates of the detected facial landmarks. Next can be seen in figure 5, the code uses

the OpenCV function face_cascade.detectMultiScale to detect faces on the same grayscale image, using the same parameters as before (1.3 and 5). The next for loop then draws a rectangle around each detected face in the image using OpenCV's cv2.rectangle function. The arguments passed to this function include the image "frame" to draw the rectangle, the coordinates of the top left and bottom right corners of the rectangle, the colour of the rectangle (in this case, blue), and the width of the edges of the rectangle.

Default Process

Figure 5. Detect facial landmarks on the grayscale image and form a blue square facial frame parameter.

```
leftEye = shape[lStart:lEnd]
rightEye = shape[rStart:rEnd]
leftEyeAspectRatio = eye_aspect_ratio(leftEye)
        rightEyeAspectRatio = eye_aspect_ratio(rightEye)
eyeAspectRatio = (leftEyeAspectRatio + rightEyeAspectRatio) / 2
leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
        cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
        cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

The code above uses the facial landmarks detected by the dlib detector library function to find the eyes in the image. The Shape object is the output object of the detector function that contains the coordinates of the facial landmarks. The code uses the facial landmark indices corresponding to the left and right eyes, stored in variables lStart, lEnd, rStart, rEnd, to extract the coordinates of the left and right eyes from the shape object and store them in variables leftEye and rightEye, respectively.

It then uses the special function eye_aspect_ratio(eye) to calculate the aspect ratio of the two eyes. The aspect ratio is a measure of how "open" the eyes are, and is calculated as the ratio of the distance between the vertical eye landmarks to the distance between the horizontal eye landmarks.

Next, the code uses the OpenCV function cv2.convexHull to remove the mismatch in the eye contour and draw the eye shape around the eye (Figure 6). The convexHull function takes the left and right eye coordinates as input, and displays an object that represents the convex hull of the eye shape. The cv2.drawContours function is then used to draw the contours of the eye shape on the green "frame" image. The arguments passed to this function include the "frame" image for drawing contours, the contour object returned by cv2.convexHull, the index of the contour to be drawn (-1 indicates that all contours should be drawn), the colour of the contour (in this case, green), and the width of the contour.

Default Process

Figure 6. Detects the facial landmarks corresponding to the left and right eyes in the grayscale image and forms a green square left and right eye frame parameter.

```
if(eyeAspectRatio < EYE_ASPECT_RATIO_THRESHOLD):
        COUNTER += 1
if COUNTER >= EYE_ASPECT_RATIO_CONSEC_FRAMES:
            pygame.mixer.music.play(-1)
            cv2.putText(frame, "you're sleepy", (100,200),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0,0,255), 5)
Else:
        pygame.mixer.music.stop()
        COUNTER = 0
```

The code above is checking whether the eye aspect ratio, stored in the variable "eyeAspectRatio", is less than the specified threshold, stored in the constant "EYE_ASPECT_RATIO_THRESHOLD". If the aspect ratio is less than the threshold, it means that the eyes are closed, indicating that the person may be sleepy.

In figure 7, when the aspect ratio is less than the threshold, the code increases the value of the counter variable "COUNTER" by 1. If the counter variable reaches the specified threshold value "EYE_ASPECT_RATIO_CONSEC_FRAMES", it means that the eyes have been closed for a certain number of consecutive frames, indicating that the person is sleepy. When the counter variable reaches this threshold, the code plays the sound file with the help of the Pygame library, and displays the message "you are sleepy" on the image using the OpenCV function cv2.putText. This message is written on the image at position (100,200) with the font "cv2.FONT_HERSHEY_SIMPLEX" with size 2, red in colour and with a thickness of 5. If the aspect ratio is not less than the threshold, it means that the eyes are open, and the code stops the sound file and resets the counter variable to 0.

Default Process

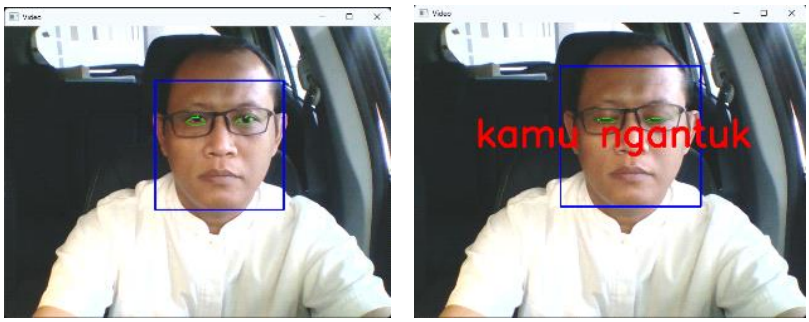

Figure 7. Detecting The Aspect Ratio of Eye.

5. CONCLUSIOON

The conclusion of this paper will summarise the results of the drowsiness detection system and its performance. It will discuss the advantages and limitations of the system, and provide suggestions for future work.

It is likely to be mentioned that the system is able to accurately detect drowsiness in real-time by analysing the aspect ratio of the eyes, and that the use of facial landmarks and a convex hull allows the system to robustly detect drowsiness even in poor lighting conditions or when the person is wearing glasses. In addition, it is also mentioned that voice alarms and text message notifications are able to alert drivers to their drowsiness.

The conclusion will also mention the limitations of the system, such as the requirement to see a person's face clearly, and the potential for false positives if the person's eyes are closed for non-drowsy reasons (e.g. blinking).

6. REFERENCES

Al Tawil L, Aldokhayel S, Zeitouni L, Qadoumi T, Hussein S, Ahamed SS. 2020. Prevalence of self-reported computer vision syndrome symptoms and its associated factors among university students. *European Journal of Ophthalmology* **30(1)**:189-195 DOI 10.1177/1120672118815110.

Čech J, Franc V, Uřičář M, Matas J. 2016. Multi-view facial landmark detection using a 3D shape model. *Image and Vision Computing* **47**:60-70 DOI 10.1016/j.imavis.2015.11.003.

Dhiraj, Jain DK. 2019. An evaluation of deep learning based object detection strategies for threat object detection in baggage security imagery. *Pattern Recognition Letters* **120**:112-119 DOI 10.1016/j.patrec.2019.01.014.

Divjak M, Bischof H. 2009. Eye blink based fatigue detection for prevention of computer vision syndrome. In: *Proceedings of the 11th IAPR conference on machine vision applications, MVA 2009.*

Fatima B, Shahid AR, Ziauddin S, Safi AA, Ramzan H. 2020. Driver fatigue detection using viola jones and principal component analysis. *Applied Artificial Intelligence* **34(6)**:456-483 DOI 10.1080/08839514.2020.1723875.

King DE. 2009. Dlib-ml: a machine learning toolkit. *Journal of Machine Learning Research* **10**:1755-1758.

Królak A, Strumiłło P. 2012. Eye-blink detection system for human-computer interaction. *Universal Access in the Information Society* **11(4)**:409-419 DOI 10.1007/s10209-011-0256-6.

Pan G, Sun L, Wu Z, Lao S. 2007. Eyeblink-based anti-spoofing in face recognition from a generic webcamera. In: *Proceedings of the IEEE international conference on computer vision*. Piscataway: IEEE, 1-8 DOI 10.1109/ICCV.2007.4409068.

Rahman A, Sirshar M, Khan A. 2016. Real time drowsiness detection using eye blink monitoring. In: *2015 National software engineering conference, NSEC, 2015*. 1-7 DOI 10.1109/NSEC.2015.7396336.

Rosenfield M. 2011. Computer vision syndrome: a review of ocular causes and potential treatments. *Ophthalmic and Physiological Optics* **31(5)**:502-515 DOI 10.1111/j.1475-1313.2011.00834.x.

Wilson GF. 2002. An analysis of mental workload in pilots during flight using multiple psychophysiological measures. *International Journal of Aviation Psychology* **12(1 SPEC)**:3-18 DOI 10.1207/s15327108ijap1201_2.

Wu Y, Ji Q. 2019. Facial landmark detection: a literature survey. *International Journal of Computer Vision* **127(2)**:115-142 DOI 10.1007/s11263-018-1097-z.

Yin S, Wang S, Chen X, Chen E, Liang C. 2020. Attentive one-dimensional heatmap regression for facial landmark detection and tracking. In: *MM 2020 - Proceedings of the 28th ACM international conference on multimedia*. New York: ACM, DOI 10.1145/3394171.3413509.