Eye Blink Detection for Coma Patients

P.JAYASIMHA REDDY[1], N. AMARNATH REDDY[2], G.C PRABHAKAR[3]
UG Scholar[1], Assistant Professor[2], Assistant Professor[3]
Department of Electrical and Electronics Engineering[1,2,3],
VNR VJIET - HYD, Telangana India[1,2,3]

Abstract - Coma, often known as unconsciousness, is a state in which the patient is completely ignorant of both himself and their surroundings, making it impossible for them to respond meaningfully to external stimuli. This can happen as a result of a complication of an underlying condition, as well as injuries from accidents, such as head trauma, and other factors such as sickness or infection, as well as drunkenness that affects the central nervous system. The Glasgow Coma Scale is used to treat people who are in a coma (GCS). Patients who are unconscious are difficult to manage, and a methodical team approach is essential. As a result, they must be closely monitored, which is challenging to achieve. This project aids in the treatment of a coma patient based on one of GSC's most critical factors, ocular movement. Calculating the Eye aspect ratio can be useful in diagnosing an unconscious patient (EAR). The OpenCV library calculates the Eye Aspect Ratio (EAR) using an algorithm that recognizes facial features and attributes. When the patient blinks, this system notifies the medical assistant using the LabVIEW application.

Keywords: Raspberry Pi forecasting, Eye aspect ratio, LabVIEW, rehabilitation

I. INTRODUCTION

There are numerous accidents occurring in and across the world. The most serious accidents, according to a survey, are those involving brain injury. Given the necessity for intense medical Care and long-term rehabilitation, it is one of the most serious challenges in general health Injuries. The majority of head injuries do not result in death, but they can put a person in a coma for a lengthy time. A coma is an unconscious state that can arise as a result of a traumatic event such as a medical Ailment or a blow to the head. Even if a person in a coma looks to be normal, they are unable to respond to external commands. As a result, doctors will examine the patient's motions and reflexes, as well as his or her response to painful stimuli and pupil size. The coma patients are helped to determine the patient's health status with great care. However, because the change is so minor, ongoing monitoring is required, which is a time-consuming task.

In the current hospital system, a healthcare worker must continuously watch and record all essential information of a specific subject while manually preserving all of that comatose's records. Continuous supervision by a paramedical assistant is an error-prone strategy that might lead to complications owing to human mistake. In the event of critically ill patients, vital parameters must be measured every 15 seconds or more until the patient's condition stabilises.

As a result, coma patients must be monitored differently than regular patients. Because the staff-to-patient ratio is so low, it is extremely difficult for paramedical staff to continuously check each patient's condition 24 hours a day, seven days a week. As a result, regularly monitoring each and every patient is a difficult effort.

This technique is designed to reduce the stress of constant supervision by only alerting the doctor or paramedical staff when assistance is required. This method will assist the doctor in determining whether the patient's condition is stable or unstable, as well as monitoring the comatose on a regular basis to see if there are any changes in the vegetative state patient's physical movement.

We created a project that deals with the blinking mechanism of the eyes. We utilised a Raspberry PI to detect the difference in eye locations and send the signal to LabVIEW, a graphical programming language tool that sends an alarm message to the person who is worried everytime the patient blinks. There's also a buzzer to alert a nearby worried guardian or medical professional. This happens on a regular basis, and the data is saved for future use.

II. RELATED WORK

This study describes the development of a small gadget for continuous eye blinking and face detection. Patients in comas cannot receive enough treatment; consequently, an accurate, inexpensive, and portable eye blink device is

Required for timely action and reliable data collecting. In a circumstance where there is no doctor, carer, or concerned individual, such a device is even more important. This study's built system includes a Raspberry Pi quad-core microprocessor system, an observation monitor, serial data transfer, and LabVIEW code to save data and deliver an alert message to the appropriate person. The system collects data on eye activity on a portable device in real time and displays it on a connected monitor with an attached Raspberry Pi in real time. Due to the use of a readily available Raspberry Pi and a small monitor as

a monitoring device, the produced system is more inexpensive than other developed devices. When compared to other measurement equipment, the created gadget has demonstrated satisfactory results.

Using image processing techniques such as pattern identification and pattern rejection algorithms, Naveen Kansal et al. [1] proposed a study titled "Advanced Coma Patient Monitoring System."

Malika et al. [2] offer a "Indoor Wireless Zigbee based Patient Monitoring system for Hospitals|." This system includes a PIR sensor, temperature sensor, humidity sensor, and smoke sensor that are all wirelessly connected to the patient's body, alerting the doctor in the event of a medical emergency.

Flux sensors are fixed in two hands of the patient, and an eyeball sensor is fixed in the patient's eye, according to Josphine Leela et al. [3] in their paper "Body Movement and Heart Beat Monitoring For Coma Patient Using IoT." When there is any movement in the hands or the eye, these sensors send the information to the doctors via wireless communication using IOT.

Navya et al. [4] built a "Zigbee-based wireless sensor-based network for patient health monitoring." This system includes a heartbeat sensor, MEMS sensor, body temperature sensor, and saline level sensor, with data transferred by Zigbee, which is commonly used to monitor the elderly.

"Real Time Eye Blinking Detection and Tracking Using Opencv," by Dhaval Pimplaskar et al. [5], is a system suggested by Dhaval Pimplaskar et al. Based on the initial centroid analysis technique, this research provided a novel method for estimating eye position and orientation.

Chopade et al. [6] presented a "Remote patient health monitoring system using Zigbee protocol" to track patients that require continuous monitoring, such as those with heart failure. Temperature and heartbeat sensors are included in this system, and these readings are sent utilising ATMEGA and the Zigbee unit.

Chen et al. [7] created a "Wearable inertial sensor for human motion analysis to continuously track motions and positions of elderly individuals." For motion tracking, this system includes an inertial measurement unit, such as a MEMS sensor.

AlSharqi et al. [8] created a "System for monitoring the health status of old persons." The doctor receives the reports via Zigbee. This system includes a local monitoring computer, a pulse sensor, an ECG sensor, and a muscle sensor, as well as a Zigbee transmitter and receiver.

III. BLOCK DIAGRAM

Block Diagram of the Eye Blink monitoring system

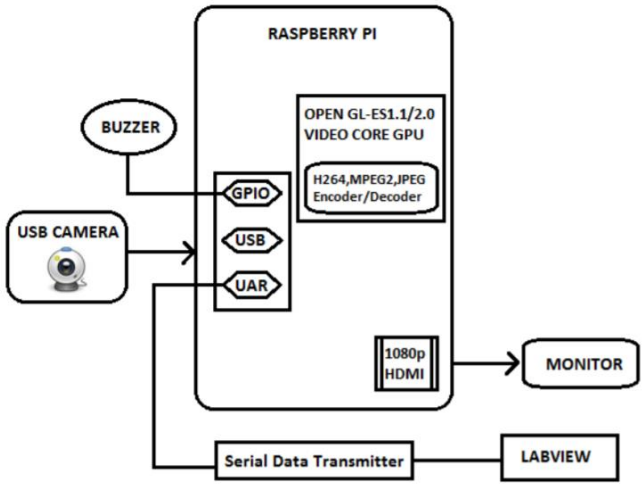


Figure 1: Block Diagram of the Eye Blink monitoring system

IV. PROPOSED METHODOLOGY

Steps for implementation

The hardware model or setup of the project is showed in the below figure.

Figure 2: Raspberry Pi with a buzzer

Step 1: Connect to a terminal or SSH server.

Following are the procedures to set up a terminal or SSH connection:

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op  Shell
1 $ workon cv
2 $ pip install RPi.GPIO
3 $ pip install gpiozero
```

From there, if you want to check that everything is installed properly in your virtual environment you may run the Python interpreter directly:

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op  Python
1 $ workon cv
2 $ python
3 >>> import RPi.GPIO
4 >>> import gpiozero
5 >>> import numpy
6 >>> import dlib
7 >>> import cv2
8 >>> import imutils
```

Because NumPy, dlib, OpenCV, and imutils are already installed in my cv virtual environment, I was able to access all six libraries by using pip to install the RPi.GPIO and gpiozero to install the relevant GPIO packages. Each of the packages can be installed using pip (except for OpenCV). Later, an optimised OpenCV and dlib are installed on your Raspberry Pi.

Step 2: To begin, we'll use OpenCV's HOG extractor to recognise the face in an image, which boils down to determining the face's bounding box (x, y)-coordinates. We can use dlib's facial landmark predictor to get 68 salient points to locate the eyes, eyebrows, nose, mouth, and jaw line given the bounding box of the face.

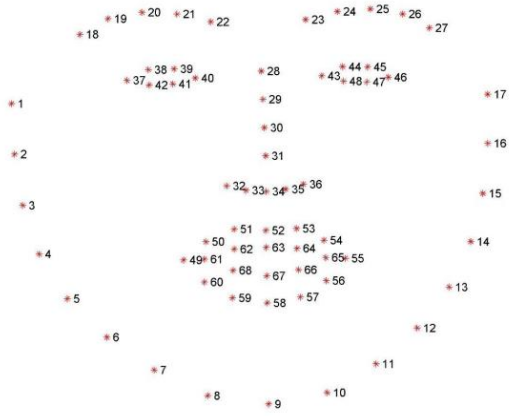


Figure 3: Visualizing the 68 facial landmark coordinates

Because Dlib's 68 face landmarks are indexable, we can use basic Python array slices to extract the various facial components.

Step 3: Using the Eye Aspect Ratio (EAR) technique, we can determine the facial landmarks linked with each eye.

The coordinates of both eyes are crucial to us since they will allow us to determine whether or not the eyes are open. Each eye contains six distinct coordinates: two at the top, two at the bottom, and one on each horizontal edge of the eye.

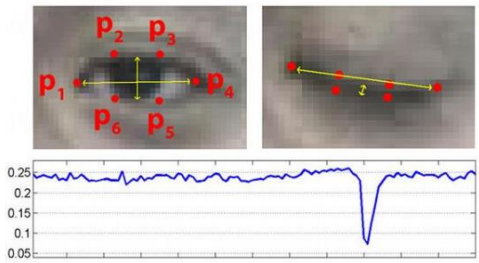


Figure 4: Eye Aspect Ratio

When the eye is open, top-left: A representation of eye landmarks.

Eye landmarks while the eye is closed, top-right.

Bottom: The eye aspect ratio throughout time is plotted.

A blink is indicated by a decrease in the ocular aspect ratio.

Starting from the left side of the eye and moving clockwise with the remaining coordinates, each of the six locations is labelled P1 to P6. 21.

$$\mathbf{EAR} = \frac{||p2 - p6|| + ||p3 - p5||}{2||p1 - p4||}$$

Eye Aspect Ratio Equation

The distance between two points, such as p2 and p6, is determined using the NumPy library's Euclidean distance function. Calculating the linear distance between two places using the Euclidean distance is a popular method. The distance between vertical coordinates is computed by the numerator of the EAR formula in fig 6, whereas the distance between horizontal coordinates is computed by the numerator.

This algorithm saves us time in determining whether or not the eyes are closed, because the traditional way of determining whether or not they are closed requires additional picture processing. The use of EAR is noninvasive, and it allows us to detect the amount of time the eyes are open and whether they are open or closed. Other ways for determining whether or not the eyes are closed necessitate greater processing power. The employment of an illuminated pupil is one of the approaches. The eyes are exposed to IR rays and reflect them, resulting in a bright pupil effect in Ghosh, Nandy, and Manna's (2015) Real Time Eye Detection and Tracking Method for Driver Assistance System [3].

This approach states that if a bright pupil is identified, the eyes are open, and if no bright pupils are found, the eyes are closed. The processor will have to do more effort to find out if a pupil

is bright, which will slow down the system. EAR, on the other hand, is based on simple mathematics, and the findings are quick and accurate.

We have an eye that is fully open and the eye facial landmarks plotted on the top-left. Then there's a closed eye in the top-right corner.

The eye aspect ratio is then plotted over time at the bottom. As can be seen, the eye aspect ratio is constant (showing that the eye is closed), then swiftly increases to near the threshold, then rapidly decreases, indicating that a blink has occurred. In the instance of our eye blink detector, we'll watch the eye aspect ratio to see if it increases but does not drop, indicating that the driver or user has opened their eyes.

Once developed, our algorithm will begin by extracting the eye region and then localising the face landmarks.

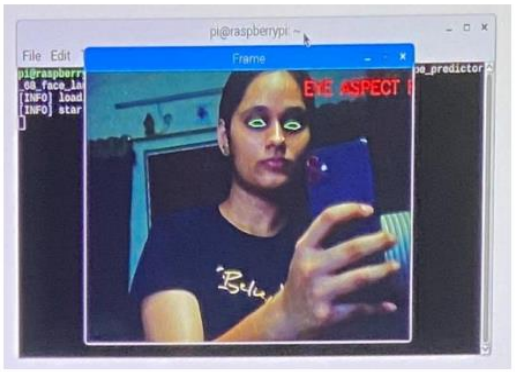


Figure 5: eyes closed — normal coma patient condition

The eye aspect ratio can then be monitored to see if the eyes are open.

Figure 6: EAR is high because the eyes were opened — the patient has blinked.

Finally, if the eye aspect ratio exceeds a pre-defined threshold for an extended period of time, an alarm will be raised.

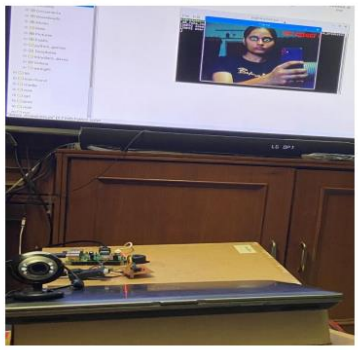


Figure 7: EAR has been below the threshold

Long enough for the blinking eye alarm to go off. We'll use OpenCV, dlib, and Python to develop the optimised eye blink detection algorithm described above on the Raspberry Pi in the next part.

4th Step: Create a new file called pi drowsiness detection.py in your IDE.

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
1  # import the necessary packages
2  from imutils.video import VideoStream
3  from imutils import face_utils
4  import numpy as np
5  import argparse
6  import imutils
7  import time
8  import dlib
9  import cv2
```

To handle the imports — make sure you have each of these installed in your virtual environment.

Let's define a distance function in **step 5:**

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
11 def euclidean_dist(ptA, ptB):
12     # compute and return the euclidean distance between the two
13     # points
14     return np.linalg.norm(ptA - ptB)
```

We define a NumPy convenience function for computing the Euclidean distance on lines 11-14. The most well-known is Euclidean, which requires the use of a distance metric. The distance between two points is commonly referred to as the Euclidean distance.

Step 6: Let's define the Eye Aspect Ratio (EAR) function, which is used to calculate the ratio of distances between vertical and horizontal eye landmarks.

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
16 def eye_aspect_ratio(eye):
17     # compute the euclidean distances between the two sets of
18     # vertical eye landmarks (x, y)-coordinates
19     A = euclidean_dist(eye[1], eye[5])
20     B = euclidean_dist(eye[2], eye[4])
21
22     # compute the euclidean distance between the horizontal
23     # eye landmark (x, y)-coordinates
24     C = euclidean_dist(eye[0], eye[3])
25
26     # compute the eye aspect ratio
27     ear = (A + B) / (2.0 * C)
28
29     # return the eye aspect ratio
30     return ear
```

When the eye is closed, the return value is nearly constant, but when the eye blinks, it increases to a pre-defined value. The eye aspect ratio will remain constant at a considerably smaller value or 0 if the eye is closed.

Step 7: We must then parse our command line inputs.

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
32 # construct the argument parse and parse the arguments
33 ap = argparse.ArgumentParser()
34
35
36 ap.add_argument("-p", "--shape-predictor", required=True,
37     help="path to facial landmark predictor")
38 ap.add_argument("-a", "--alarm", type=int, default=0,
39     help="boolean used to indicate if TrafficHat should be used")
40 args = vars(ap.parse_args())
```

On Lines 36-40, we've established two necessary arguments and one optional argument: The path to the dlib face landmark predictor file is shape-predictor.

alarm: When an eye blink is detected, this Boolean indicates whether the piezoelectric buzzer should be employed.

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
42 # check to see if we are using GPIO/TrafficHat as an alarm
43 if args["alarm"] > 0:
44     from gpiozero import TrafficHat
45     th = TrafficHat()
46     print("[INFO] using TrafficHat alarm...")
```

As shown in Lines 43-46 if the argument supplied is greater than 0, we'll import the raspberry pi function to handle our buzzer alarm.

Let's also define a set of important configuration variables:

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
48 # define two constants, one for the eye aspect ratio to indicate
49 # blink and then a second constant for the number of consecutive
50 # frames the eye must be below the threshold for to set off the
51 # alarm
52 EYE_AR_THRESH = 0.3
53 EYE_AR_CONSEC_FRAMES = 16
54
55 # initialize the frame counter as well as a boolean used to
56 # indicate if the alarm is going off
57 COUNTER = 0
58 ALARM_ON = False
```

The EAR threshold is defined by the two constants on Lines 52 and 53.

The frame counter and a Boolean for the alarm are then initialised (Lines 57 and 58). In order to extract the eye regions from the set of face landmarks, we provide array slice indexes. Now is the time to begin our video feed thread:

Step 8: From there we'll load our Haar cascade

and facial landmark predictor files:

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
60 # load OpenCV's Haar cascade for face detection (which is faster than
61 # dlib's built-in HOG detector, but less accurate), then create the
62 # facial landmark predictor
63 print("[INFO] loading facial landmark predictor...")
64 detector = cv2.CascadeClassifier(args["cascade"])
65 predictor = dlib.shape_predictor(args["shape_predictor"])
```

Line 64 is different from the prior face detector initialization. Because Haar cascades are faster than dlib's face detector, it's an excellent choice for the Raspberry Pi.

Line 65, where we load dlib's shape predictor while specifying the file path, remains unchanged.

Step 9: For each eye, we'll now establish the indices of the facial landmarks:

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
67 # grab the indexes of the facial landmarks for the left and
68 # right eye, respectively
69 (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
70 (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
72 # start the video stream thread
73 print("[INFO] starting video stream thread...")
74 vs = VideoStream(src=0).start()
75 # vs = VideoStream(usePiCamera=True).start()
76 time.sleep(1.0)
```

If you are using the PiCamera module, be sure to *comment out* **Line 74** and *uncomment* **Line 75** to switch the video stream to the Raspberry Pi camera. Otherwise if you are using a USB camera, you can leave this unchanged.

We have one second sleep so the camera sensor can warm up.

From there let's loop over the frames from the video stream:

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
78 # loop over frames from the video stream
79 while True:
80     # grab the frame from the threaded video file stream, resize
81     # it, and convert it to grayscale
82     # channels)
83     frame = vs.read()
84     frame = imutils.resize(frame, width=450)
85     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
86
87     # detect faces in the grayscale frame
88     rects = detector.detectMultiScale(gray, scaleFactor=1.1,
89         minNeighbors=5, minSize=(30, 30),
90         flags=cv2.CASCADE_SCALE_IMAGE)
```

We take a frame and read it, resize it (to save time), and convert it to grayscale (Lines 83-85). Then, using our detector on Lines 88-90, we detect faces in the grayscale image. Let's go over the detections again.

```
Raspberry Pi: Facial landmarks + Eye blink detection with Op
92     # loop over the face detections
93     for (x, y, w, h) in rects:
94         # construct a dlib rectangle object from the Haar cascade
95         # bounding box
96         rect = dlib.rectangle(int(x), int(y), int(x + w),
97             int(y + h))
98
99         # determine the facial landmarks for the face region, then
100        # convert the facial landmark (x, y)-coordinates to a NumPy
101        # array
102        shape = predictor(gray, rect)
103        shape = face_utils.shape_to_np(shape)
```

Line 93 begins a lengthy for-loop which is broken down into several code blocks here. First we extract the coordinates and width + height of the `rects` detections. Then, on **Lines 96 and 97** we construct a dlib `rectangle` object using the information extracted from the Haar cascade bounding box.

From there, we determine the facial landmarks for the face region (**Line 102**) and convert the facial landmark *(x, y)*-coordinates to a NumPy array.

Given our NumPy array, `shape` , we can extract each eye's coordinates and compute the EAR:

```
Raspberry Pi: Facial landmarks + Eye blink detection with Ope
105         # extract the left and right eye coordinates, then use the
106         # coordinates to compute the eye aspect ratio for both eyes
107         leftEye = shape[lStart:lEnd]
108         rightEye = shape[rStart:rEnd]
109         leftEAR = eye_aspect_ratio(leftEye)
110         rightEAR = eye_aspect_ratio(rightEye)
111
112         # average the eye aspect ratio together for both eyes
113         ear = (leftEAR + rightEAR) / 2.0
```

We can slice the shape array using the indexes of the eye landmarks to get the (x, y)-coordinates for each eye (Lines 107 and 108).

On Lines 109 and 110, we calculate the EAR for each eye.

To achieve a better estimate, consider averaging both eye aspect ratios together (Line 113).

Step 10: This next block is strictly for visualization purposes:

```
Raspberry Pi: Facial landmarks + Eye blink detection with Ope
115         # compute the convex hull for the left and right eye, then
116         # visualize each of the eyes
117         leftEyeHull = cv2.convexHull(leftEye)
118         rightEyeHull = cv2.convexHull(rightEye)
119         cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
120         cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

Using cv2.drawContours and the cv2.convexHull computation of each eye, we can visualise each of the eye areas on our frame (Lines 117-120). These few lines are useful for debugging our script, but they aren't required if you're creating an embedded product without a display.

From there, we will check our Eye Aspect Ratio (EAR) to see if the eyes are opened while sounding the alarm to warn the eye blink the patient if needed

```
Raspberry Pi: Facial landmarks + Eye blink detection with Ope
122         # check to see if the eye aspect ratio is below the blink
123         # threshold, and if so, increment the blink frame counter
124         if ear < EYE_AR_THRESH:
125             COUNTER += 1
126
127             # if the eyes were closed for a sufficient number of
128             # frames, then sound the alarm
129             if COUNTER >= EYE_AR_CONSEC_FRAMES:
130                 # if the alarm is not on, turn it on
131                 if not ALARM_ON:
132                     ALARM_ON = True
133
134                     # check to see if the TrafficHat buzzer should
135                     # be sounded
136                     if args["alarm"] > 0:
137                         th.buzzer.blink(0.1, 0.1, 10,
138                             background=True)
139
140                 # draw an alarm on the frame
141                 cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
142                     cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
143
144         # otherwise, the eye aspect ratio is not below the blink
145         # threshold, so reset the counter and alarm
146         else:
147             COUNTER = 0
148             ALARM_ON = False
```

On Line 124, we compare the EAR to the EYE AR THRESH and sound the alarm if it is larger than zero (eyes are open) (Line 129).

As seen on Lines 141 and 142, we can draw the alarm on the frame.

That gets us to the situation where the EAR is less than the EYE AR THRESH, and we need to make sure our alarm is turned off (Lines 146-148).

We're almost done; in the final code block, we'll draw the EAR on the frame, show it, and clean up.

```
Raspberry Pi: Facial landmarks + Eye blink detection with Ope
150         # draw the computed eye aspect ratio on the frame to help
151         # with debugging and setting the correct eye aspect ratio
152         # thresholds and frame counters
153         cv2.putText(frame, "EAR: {:.3f}".format(ear), (300, 30),
154             cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
155
156     # show the frame
157     cv2.imshow("Frame", frame)
158     key = cv2.waitKey(1) & 0xFF
159
160     # if the `q` key was pressed, break from the loop
161     if key == ord("q"):
162         break
163
164 # do a bit of cleanup
165 cv2.destroyAllWindows()
166 vs.stop()
```

Step 11: If you're integrating with a screen or debugging, you might want to show the computed eye aspect ratio on the frame, like I did on Lines 153 and 154. On Lines 157 and 158, the frame is displayed on the actual screen.

When the ctrl + c key on a keyboard is pushed (Lines 157 and 158), the programme is terminated. Finally, we tidy up by closing any open windows and turning off the video (Lines 165 and 166).

Step 12: The results of the eye blink detection. Face detection Hog and dlib facial landmark detector are required to run this programme on your Raspberry Pi.

An image of my setup can be found below.

Figure 8: Desk set up for coding, testing, and debugging the eye blink Detector

Step 13: To run the program, simply execute the following command:

```
Raspberry Pi: Facial landmarks + c Eye blink detection with Op
$ python pi_detect_  eyeblink.py --cascade haarcascade_frontalface_default.xml \
    --shape-predictor shape_predictor_68_face_landmarks.dat --alarm 1
```

Our Raspberry Pi 3 is able to accurately determine when there is an “eye blink”.

Our Raspberry Pi 3 is able to accurately determine when there is an “eye blink”

V. RESULTS

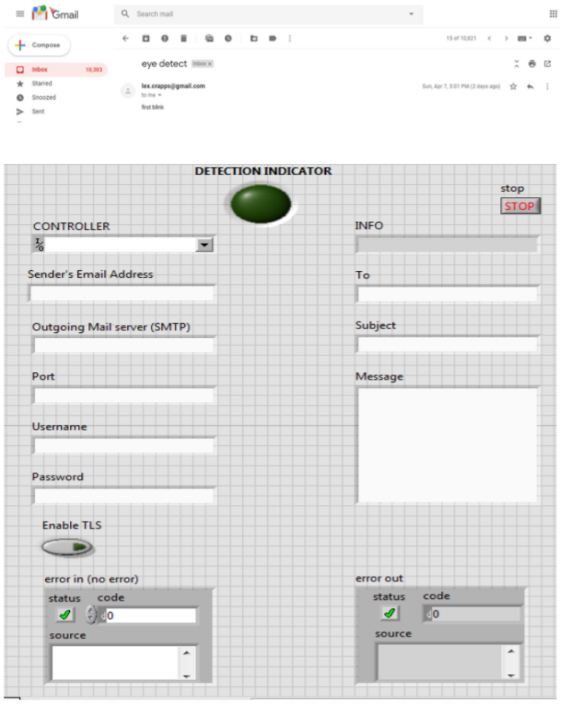
Many technologies that rely on eye movement are available on the market.

The most exact technique employed among all the gadgets is based on human physiological measurements. Projects that rely on eye movement tracking include IR-based eye blink detection and an eye movement-controlled wheelchair. This method is carried out by observing changes in physiological signals such as eye flashing, or the open/closed state of the eyes. A cursor control system based on MATLAB is also available.

Despite the fact that these processes are the most exact, they are not appropriate for some projects because detecting electrodes or other sensors would have to be placed directly on the patient's body, which would be inconvenient. Long periods of inactivity cause sweat to form on the sensors, limiting their ability to screen precisely. As a result, our approach will primarily focus on the amount of eye closure, also known as (PERCLOS) percentage of closure, because it offers the most accurate information on eye blink and leverages computer vision via Python programming, making it superior to other devices.

Figure 9: Showing alert when patient blinked

The labview mail alert when the patient blinked is shown below:



LabVIEW front panel block diagram for Email

Figure 10: LabVIEW graphical user interface

This project is similar to the others in that it is designed using computer vision and python programming, but it is superior to the others in that it is designed using computer vision and python programming. A better night vision camera helps to make the device a better one by using image processing. This device not only detects eye blinks, but it can also be used to detect driver sleep by slightly altering the code.

A better night vision camera helps to make the device a better one by using image processing. This device not only detects eye blinks, but it can also be used to detect driver sleep by slightly altering the code. It is also nonintrusive in nature, therefore it has no effect on the patient's condition, and the patient is completely at ease with it. Face recognition and tracking, human eye detection and location, human eye tracking, and eye state detection are all part of the system's development. The detection framework's main components combined the detection and localization of human eyes.

VI. CONCLUSIONS

This paper proposes a system that is both small and delivers real-time processing. The suggested system's goal is to create an easily accessible design that quickly communicates the patient's vital information to the doctor. The created model results in a better and more effective health-care service for checking comatose levels, and the data obtained is networked

with the use of the internet and communication, allowing for a faster response.

The Internet of Medical Things (IoMT) market includes a wide range of smart devices in the home and hospital, such as wearables and medical/vital monitors, as well as linked real-time location, telehealth, and other services. As a result, a doctor can readily assess his patient at any moment using these equipment. The suggested technology employs an eye blink sensor to detect movement of the comatose's eyes. Patients can also afford the system because it is well-designed.

VII. REFERENCE

[1] Naveen Kansal, Hardeep Singh Dhillon, "Advanced Coma Patient Monitoring System", International Journal of Scientific & Engineering Research Volume 2, Issue 6, June-2011 ISSN 2229-5518.

[2] Malika, Charu Rana, "An Indoor Wireless Zigbee based Patient Monitoring system for Hospitals", International Journal of Engineering Sciences Research-IJESR, Vol 04, Issue 02; March-April 2013.

[3] Josphine Leela, "Body Movement and Heart Beat Monitoring For Coma Patient Using IoT", International Journal of Innovative Research in Science, Engineering and Technology-IJIRSET, Vol.7, Special issue 2, March 2018.

[4] K. Navya, Dr M. B. R. Murthy, "A Zigbee Based Patient Health Monitoring System", Int. Journal of Engineering Research and Applications, www.ijera.com, Vol. 3, Issue 5, pp.483-486, Sep-Oct 2013.

[5] Dhaval Pimplaskar, "Real Time Eye Blinking Detection and Tracking Using OpenCV", Int. Journal of Engineering Research and Application, www.ijera.com, Vol. 3,Issue 5, Sep-Oct 2013, pp.1780-1787.

[6] Amruta Chopade, Prof. Nitin Raut, "Remote Patients Health Monitoring by Using Zigbee Protocol", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 4, Issue 8, August 2014.

[7] Xi Chen, "Human Motion Analysis with Wearable Inertial Sensors", University of Tennessee, Knoxville, 8- 2013.

[8] Khalifa AlSharqi, Abdelrahim Abdelbari, Ali AbouElnour, and Mohammed Tarique, "Zigbee based wearable remote healthcare monitoring system for elderly patients", International Journal of Wireless & Mobile Networks (IJWMN) Vol. 6, No. 3, June 2014