***PROBLEM STATEMENT***

**DROWSINESS DETECTION AND ALERT SYSTEM**

**Abstract:**

Drowsiness and Fatigue of drivers are amongst the significant causes of road accidents. Every year, they increase the amounts of deaths and fatalities injuries globally. In this paper, a module for Advanced Driver Assistance System (ADAS) is presented to reduce the number of accidents due to drivers fatigue and hence increase the transportation safety; this system deals with automatic driver drowsiness detection based on visual information and Artificial Intelligence. We propose an algorithm to locate, track, and analyze both the drivers face and eyes to measure PERCLOS, a scientifically supported measure of drowsiness associated with slow eye closure.

Nowadays, more and more professions require long-term concentration. Drivers must keep a close eye on the road, so they can react to sudden events immediately. Driver fatigue often becomes a direct cause of many traffic accidents. Therefore, there is a need to develop the systems that will detect and notify a driver of her/him bad psychophysical condition, which could significantly reduce the number of fatigue-related car accidents. However, the development of such systems encounters many difficulties related to fast and proper recognition of a driver’s fatigue symptoms. One of the technical possibilities to implement driver drowsiness detection systems is to use the vision-based approach. This article presents the currently used driver drowsiness detection systems. The technical aspects of using the vision system to detect a driver drowsiness are also discussed.

**INTRODUCTION**

The development of technology allows introducing more advanced solutions in everyday life. This makes work less exhausting for employees, and also increases the work safety. Vision-based systems are becoming more popular and are more widely used in different applications. These systems can be used in industry (e.g. sorting systems), transportation (e.g. traffic monitoring), airport security (e.g. suspect detection systems), and in the end-user complex products such as cars (car parking camera). Such complex systems could also be used to detect vehicle operator fatigue using vision-based solutions. Fatigue is such a psychophysical condition of a man, which does not allow for a full concentration. It influences the human response time, because the tired person reacts much slower, compared to the rested one. Appearance of the first signs of a fatigue can become very dangerous, especially for such professions like drivers. Nowadays, more and more professions require long-term concentration. People, who work for transportation business (car and truck drivers, steersmen, airplane pilots), must keep a close eye on the road, so they can react to sudden events (e.g. road accidents, animals on the road, etc.) immediately. Long hours of driving causes the driver fatigue and, consequently, reduces her/him response time . According to the results of the study presented at the International Symposium on Sleep Disorders, fatigue of drivers is responsible for 30% of road accidents . The British journal “What Car?” presented results of the experiment conducted with the driving simulator and they concluded that a tired driver is much worse dangerous than a person whose alcohol in blood level is 25% above the allowed limit . Driver fatigue can cause a microsleep (e.g. loss of concentration, a short sleep lasting from 1 to 30 seconds), and falling asleep behind the wheel. Therefore, there is a need to develop a system that will detect and notify a driver of her/him bad psychophysical condition, which could significantly reduce the number of fatigue-related car accidents. However, the biggest difficulties in development of such a system are related to fast and proper recognition of a driver’s fatigue symptoms. Due to the increasing amount of vehicles on the road, which translates into the road accidents directly, equipping a car with the fatigue detection system is a must. One of the technical possibilities to implement such a system is to use vision-based approach. With the rapid development of image analysis techniques and methods, and a number of ready Component-on-the-Shelf solutions (e.g. high resolution cameras, embedded systems, sensors), it can be envisaged, that introducing such systems into widespread use should be easy. Car drivers, truck drivers, taxi drivers, etc. should be allowed to use this solution to increase the safety of the passengers, other road users and the goods they carry.

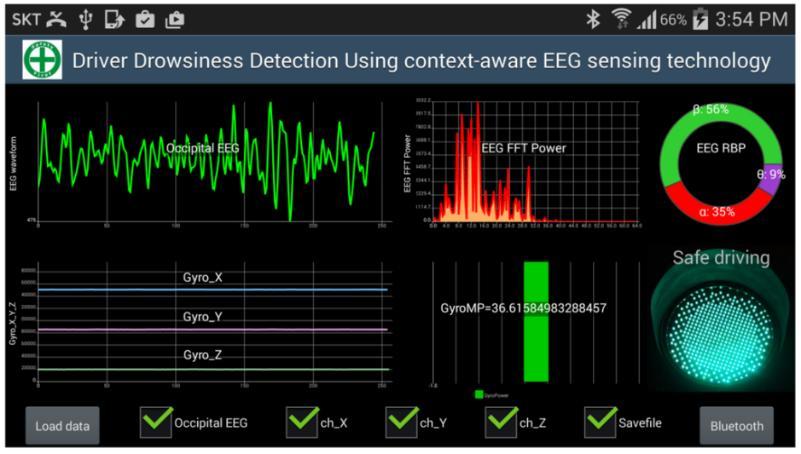
**Currently used driver fatigue detection systems**

One of the examples of a system detecting a driver’s fatigue is the system implemented into the Driver Assistant in Ford cars . It analyzes rapid steering movements, driving onto lines separating lanes, irregular and rapid braking or acceleration. The system collects and processes these data, assigns the driver using one of the 5-degree concentration levels (5 – the driver is concentrated, drives properly, 1 – the driver is very tired, should immediately stop driving and rest). When the rating falls to level 1, the driver is notified by beeps and warnings on the instrument panel's middle screen. The system can be reset and the warnings will disappear, only when the driver stops and opens the door. Skoda cars use a similar system. It analyzes the steering movements and compares them to the movements in normal driving. The system begins to analyze how the vehicle performs 15 minutes after starting the engine and at the speeds of more than 65 km/h . When the system detects that driving is abnormal, the driver's fatigue status is displayed on the screen, followed by a beep, informing the driver to take a break . Volkswagen uses the Bosch Driver Drowsiness Detection system . It also analyses how a car behaves on the road. Based on the information from the power assisted steering sensor and the steering angle sensor, the system detects sudden changes in the trajectory of the vehicle, which translates into driver’s fatigue . Some driver fatigue detection methods use the heart rate analysis . The psychophysical state is determined by the HRV (heart rate variability). DENSO (manufacturer of car parts and systems) at the Detroit Auto Show presented a system that relies on a driver's heart rate analysis and the use of the cameras to observe a driver’s eyes. Such a solution allows detecting a fatigue at the operator of the vehicle. There are also ideas for the use of electroencephalogram (EEG) to detect the driver's brain wave changes, which may indicate the first symptom of fatigue. The panel view implemented in Android is shown below.



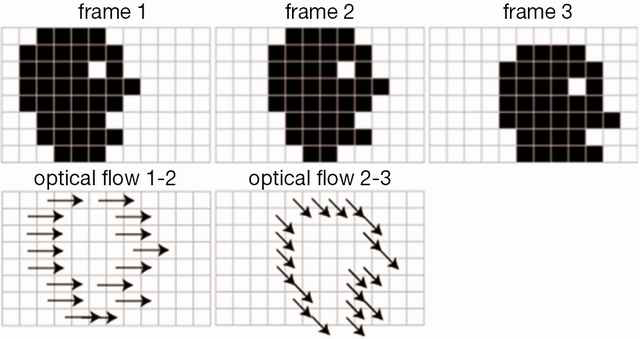
*View at a driver fatigue information (Ford Driver Assistant)*

The PSA Group (formerly PSA Peugeot Citroën), in collaboration with the Lausanne University of Technology, are working on a camera-based system to analyze the facial expressions of a driver. It is interesting to note that the very early aim of this system was a detection of emotions of a driver, but they decided to develop it into the fatigue detection system). It is based on the analysis of eye movement, the closing and opening of the eyelids as well as the movement of the mouth. It allows detecting the first symptoms of a fatigue. Information provided by this system will inform a driver on the state of her/his psychophysical state.

*Driver Drowsiness Detection System using EEG*

**Using a vision-based system to detect a driver fatigue**

Fatigue detection is not an easy task. It requires taking into account many factors. Using a video system for this purpose can be a good solution. This system would allow for precise detection of a fatigue in real time. The speed of such a system is very important because even slight delays in the operation of such a system could be fatal (excessive reaction of the system while traveling along the highway). An important issue in the design of the vision-based driver fatigue detection system is the right choice of the analyzed symptoms of fatigue. In a situation, where it is not possible to monitor all potential symptoms of a fatigue, it should be limited to the detection of the most important ones such as: closing the eyelids, slow the eye movements, yawning and drooping a head. The basis of the fatigue detection system are the algorithms responsible for detecting facial features and their motion. There are many methods that allow detecting individual facial elements. They are based both on the vector operations and the pattern classification. Particular methods are based on an image filtering in complex space or an image processing in spatial-frequency domain. Some methods are very effective in detecting characteristic facial features, but sensitive to changing lighting conditions. There are also methods that rely on analyzing a 3D image. The most popular methods are Main Component Analysis, Neural Networks, Gabor filters, and frequency-spatial methods. Principal Components Analysis (PCA) is based on its own vectors. It is often used during preprocessing (to get rid of noise from the image – they correspond to a small variance that is correlated with the corresponding own vectors) . The method based on neural networks is used for processing input data. Neural networks are used for identification and classification of pattern data, and therefore they are also used in face detection and recognition systems . Gabor filters are one of the most commonly used methods for representing facial features, using complex functions. Frequency-spatial methods are based on frequency analysis of the image in conjunction with the methods based on a geometric model. Frequency-spatial methods allow for the proper isolation of 44 the characteristic facial features and minimizing the influence of lighting conditions during the acquisition . In vision-based systems it is important to correctly identify the specific elements as well as to analyze their movement. Common methods used to detect a motion in video systems are differential and gradient methods. Differential methods determine the difference between the subsequent image frames. This allows determining the brightness level in the grayscale or the color intensity of the pixel during the frame changes. So, the movement of the object can be detected (this is related to the change in the brightness of the pixels that appear next to each other in the image). This is a simple way to detect a movement, however, its implementation can be tedious. One of the limitations of this method is to have the stationary background, the lighting should be constant, and the noise in the film should be reduced to a minimum (otherwise, the algorithm may work not properly). Additionally, in order to improve a movement detection, the moving object should contrast with the background . Gradient methods rely on the optical flow. They use spatial and temporal derivatives of the consecutive video frames. In order to make an effective use of this group of the methods, the following conditions should be met: invariability of the light, a small displacement of moving objects in one sequence and spatial coherence of the contiguous dots. Two most popular gradient algorithms are Lucas-Kanade and Horn-Schunk algorithm. The principle of operation of the first algorithm is a characteristic assumption: the brightness of the dots in the image is unchanged over the time, the movement of the frames is constant. This method is intended for the methodological purposes, that is, the section of traffic in the area devoted to the area (not exceeding the registration process). Its performance can be improved by solving this algorithm into the form of a pyramid (image analysis of a small improvement and then its gradual change). The Horn-Schunk algorithm is based on the use of the optical flow equation, taking into account two conditions: the brightness of the dots (the pixel brightness of the moving object in the image is constant) and the speed of the dots (the speed of the pixels belonging to the moving object are close to each other, the motion field changes smoothly). This method belongs to the global methods. Thanks to this, we get a high density of the flow vector, which results in a more accurate information about the movement of the object, including information about the area under investigation in which the object is moving. The disadvantage of this algorithm is that it is more susceptible to interference compared to the local methods (e.g. Lucas-Kanade). The example of the application of an optical flow is shown below.

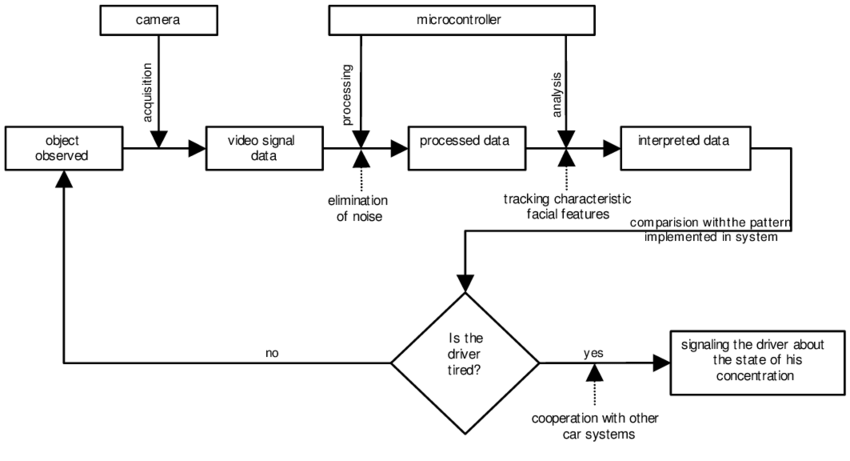


*Example of using optical flow*

When designing a video system that records moving objects, one should choose algorithms that are resistant to interference. Interference occurrence may disturb the processing and the analysis of the data, which may lead to misinterpretation by the system. If the selected methods are susceptible to interference, then the system will not analyze the movement of the elements. It may lead to a kind of dynamic "jumps" of the system between the observed objects. The next procedure is to register and analyze the movement of the classified features. It should be remembered that real-time systems require a rapid response to the changes in observed objects. For example, if the eyelid is closed for a long time, the system response: "eyelids close" should be immediate. Any delay in the identification of a fatigue can have catastrophic consequences. If the system does not respond in time to the driver's microsleep, an accident may occur. Problems related to design vision-based fatigue detection system operating in realtime also apply to the hardware used for the video signal acquisition and processing. One of the issues is the number of cameras recording the object of the interest. We can install three cameras in front of a driver, apply algorithms that generate a 3D image, and then implement the appropriate data analysis methods. Information will be the most accurate, but assiocated costs of the final system may disqualify it for the industrial scale. Limiting the number of the cameras to one will allow you to cut costs considerably. The quality of the recording equipment is also of great importance. Using equipment of poor quality, recorded video signal may become noisy, affecting the performance of the algorithms for video processing and analyzing, which ultimately can lead to misinterpretation of the results by the system. In addition, since capturing details of the face is of the great importance to the system, the vision system should use cameras high resolution image.

**BLOCK DIAGRAM**

In order to reduce the number of road accidents resulting from a driver fatigue, it is of great importance to introduce to the automotive industry a system that would effectively detect the first signs of a fatigue and notify the driver. A system based on real-time face analysis can be one of the most effective approaches for detecting fatigue symptoms. There are many problems associated with its design such as uneven illumination of a driver’s face or the selection of effective real-time data processing algorithms to name a few. Current technological advances in video recording and processing help reduce and even eliminate such problems. It is envisaged that by integrating such a system with other on-board car driving system would increase road safety definitely. The block diagram of the hypothetical system, and principle of its operation is presented in Figure 4. The investigations of the proposed drowsiness detection vision system will be continued and the results of the research will be delivered.



*Block diagram of driver drowsiness detection vision system*

**How Driver Drowsiness Detection System Works? ?**



During long journeys, it’s possible that the driver may lose attention because of drowsiness, which may be a potential reason for fatal accidents. With technologies like **Driver Drowsiness Detection** getting it is possible to detect driver’s driving behavior that may prove fatal to the vehicle as well as the people boarding it.

Having such **sleep detection system in cars** embedded in vehicles could protect precious lives and property worth billion dollars. The outcome would be positive – it would be suitable for fleet owners as well as individual vehicle users. In either case, the objective is identical by sleep detection while driving.

In this article, we’ll discuss How Driver Drowsiness Detection works in Vehicles.

Driving a vehicle involves coordination of the locomotor system along with the healthy function of the brain. When the driver feels drowsy, it may unsettle the balance and may lead to erratic driving causing potential accidents.

While driving, you may feel drowsy when you’re under driving fatigue because of continuous driving for several hours. It’s here that the driver drowsiness detection plays a significant role in preventing accidents that could otherwise cause massive loss of life and property.

## How does Driver Drowsiness Detection System work?

The driver drowsiness detection system uses Image Processing to analyze the driver’s eye blink pattern by sitting on the vehicle’s dashboard

If the eye lid movements are abnormal than usual then the detection system triggers the alarm thus alerting the driver about the condition.

According to National Highway Traffic Safety Administration around 72K crashes, 44K injuries & 800+ deaths are reportedly caused due to Drowsiness in the year 2013.

## Importance of sleep detection – Who needs to get it?

Life is precious and no number of words suffice to evaluate it. It’s, therefore, imperative to protect it from fatal consequences while driving a vehicle. And, if you’re an owner of a fleet/s of vehicles.

Further, your vehicles comprise heavy capital assets, which you need to protect from potential losses because of fatal accidents.

There are diverse products that protect and ensure the safety of vehicles but most of them don’t come with a built-in **drowsiness detection sensor**.

If you’re looking for the feature such as sleep detection in vehicles and happen to be a Fleet owner. Then you’ve to deal with multiple devices & multiple apps associated with them.

Practically this is not an ideal option for both fleet owners and even for individual car owners to deal with multiple devices.

**Drowsiness detection with OpenCV**

### Rigging with a drowsiness detector

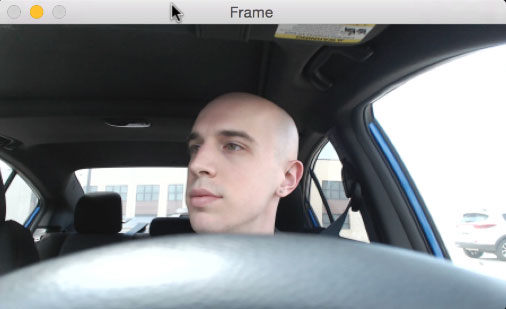


  I had intended on using my Raspberry Pi 3 due to (1) form factor and the real-world implications of building a driver drowsiness detector using very affordable hardware.The Raspberry Pi isn’t quite fast enough for real-time facial landmark detection.

**The drowsiness detector algorithm**

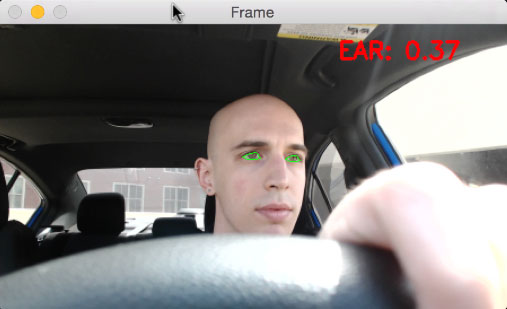
The general flow of our drowsiness detection algorithm is fairly straightforward.

First, we’ll setup a camera that monitors a stream for faces:



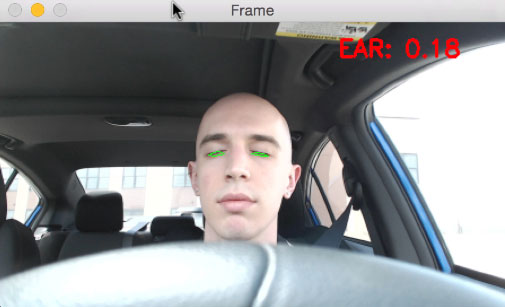
Look for faces in the input video stream.

If a face is found, we apply facial landmark detection and extract the eye regions:



Apply facial landmark localization to extract the eye regions from the face.

Now that we have the eye regions, we can compute the eye aspect ratio (detailed [here](https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/)) to determine if the eyes are closed:



**Figure 5:** Step #3 — Compute the eye aspect ratio to determine if the eyes are closed.

If the eye aspect ratio indicates that the eyes have been closed for a sufficiently long enough amount of time, we’ll sound an alarm to wake up the driver:



**Figure 6:** Step #4 — Sound an alarm if the eyes have been closed for a sufficiently long enough time.

In the next section, we’ll implement the drowsiness detection algorithm detailed above using OpenCV, dlib, and Python.

Building the drowsiness detector with OpenCV

To start our implementation, open up a new file, name it detect\_drowsiness.py , and insert the following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **# import the necessary packages**  **from scipy.spatial import distance as dist**  **from imutils.video import VideoStream**  **from imutils import face\_utils**  **from threading import Thread**  **import numpy as np**  **import playsound**  **import argparse**  **import imutils**  **import time**  **import dlib**  **import cv2** |

*import our required Python packages.*

the [SciPy](https://www.scipy.org/" \t "_blank) package is used to compute the Euclidean distance between facial landmarks points in the eye aspect ratio calculation (not strictly a requirement, but you should have SciPy installed if you intend on doing any work in the computer vision, image processing, or machine learning space).

The [imutils package](https://github.com/jrosebr1/imutils" \t "_blank), my series of computer vision and image processing functions to make working with OpenCV easier.

If you don’t already have imutils  installed on your system, you can install/upgrade imutils  via:

***pip install --upgrade imutils***

 the Thread  class so we can play our alarm in a separate thread from the main thread to ensure our script doesn’t pause execution while the alarm sounds.

In order to actually play our WAV/MP3 alarm, we need the [playsound library](https://pypi.python.org/pypi/playsound/1.2.1" \t "_blank), a pure Python, cross-platform implementation for playing simple sounds.

The playsound  library is conveniently installable via pip :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ***pip install playsound***  To detect and localize facial landmarks we’ll need the [dlib library](http://dlib.net/" \t "_blank) which is imported on **Line 11**.  Next, we need to define our sound\_alarm  function which accepts a path  to an audio file residing on disk and then plays the file:   |  |  | | --- | --- | | **14**  **15**  **16** | **def sound\_alarm(path):**  **# play an alarm sound**  **playsound.playsound(path)** |   To define the eye\_aspect\_ratio  function which is used to compute the ratio of distances between the vertical eye landmarks and the distances between the horizontal eye landmarks:   |  |  | | --- | --- | | **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32** | **def eye\_aspect\_ratio(eye):**  **# compute the euclidean distances between the two sets of**  **# vertical eye landmarks (x, y)-coordinates**  **A = dist.euclidean(eye[1], eye[5])**  **B = dist.euclidean(eye[2], eye[4])**    **# compute the euclidean distance between the horizontal**  **# eye landmark (x, y)-coordinates**  **C = dist.euclidean(eye[0], eye[3])**    **# compute the eye aspect ratio**  **ear = (A + B) / (2.0 \* C)**    **# return the eye aspect ratio**  **return ear** |   The return value of the eye aspect ratio will be approximately constant when the eye is open. The value will then rapid decrease towards zero during a blink.  If the eye is closed, the eye aspect ratio will again remain approximately constant, but will be *much smaller* than the ratio when the eye is open.  https://www.pyimagesearch.com/wp-content/uploads/2017/04/blink_detection_plot.jpg  Top-left: A visualization of eye landmarks when then the eye is open.  Top-right: Eye landmarks when the eye is closed. Bottom: Plotting the eye aspect ratio over time. The dip in the eye aspect ratio indicates a blink  On the *top-left* we have an eye that is fully open with the eye facial landmarks plotted. Then on the *top-right* we have an eye that is closed. The *bottom* then plots the eye aspect ratio over time.  As we can see, the eye aspect ratio is constant (indicating the eye is open), then rapidly drops to zero, then increases again, indicating a blink has taken place.  In our drowsiness detector case, we’ll be monitoring the eye aspect ratio to see if the value *falls* but *does not increase again*, thus implying that the person has closed their eyes.   |  |  | | --- | --- | | ***34***  ***35***  ***36***  ***37***  ***38***  ***39***  ***40***  ***41***  ***42*** | ***# construct the argument parse and parse the arguments***  ***ap = argparse.ArgumentParser()***  ***ap.add\_argument("-p", "--shape-predictor", required=True,***  ***help="path to facial landmark predictor")***  ***ap.add\_argument("-a", "--alarm", type=str, default="",***  ***help="path alarm .WAV file")***  ***ap.add\_argument("-w", "--webcam", type=int, default=0,***  ***help="index of webcam on system")***  ***args = vars(ap.parse\_args())*** | |

Our drowsiness detector requires one command line argument followed by two optional ones, each of which is detailed below:

* --shape-predictor : This is the path to dlib’s pre-trained facial landmark detector.
* --alarm : Here you can optionally specify the path to an input audio file to be used as an alarm.
* --webcam : This integer controls the index of your built-in webcam/USB camera.

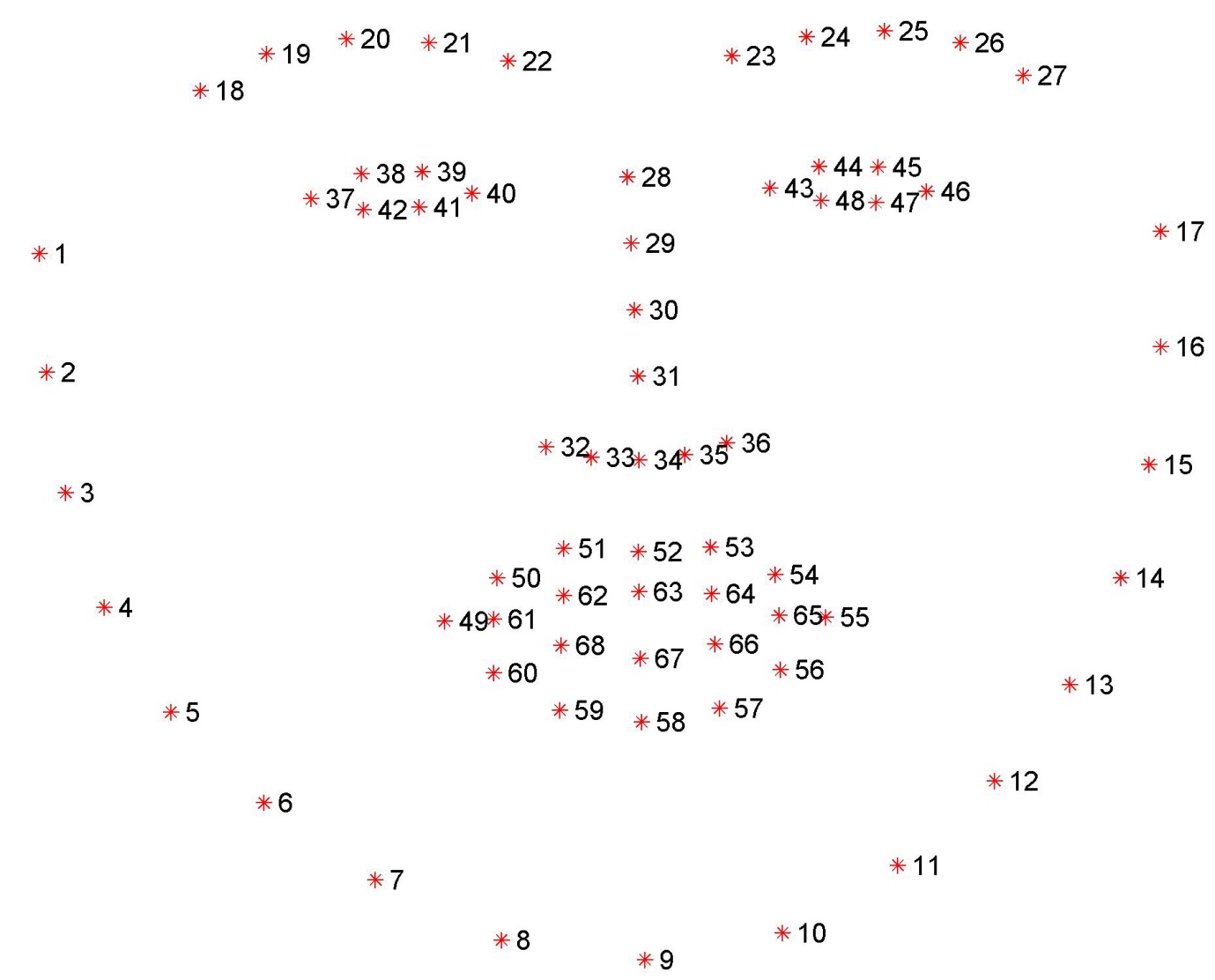
Now that our command line arguments have been parsed, we need to define a few important variables:

|  |  |
| --- | --- |
| **44**  **45**  **46**  **47**  **48**  **49**  **50**  **51**  **52**  **53**  **54** | **# define two constants, one for the eye aspect ratio to indicate**  **# blink and then a second constant for the number of consecutive**  **# frames the eye must be below the threshold for to set off the**  **# alarm**  **EYE\_AR\_THRESH = 0.3**  **EYE\_AR\_CONSEC\_FRAMES = 48**    **# initialize the frame counter as well as a boolean used to**  **# indicate if the alarm is going off**  **COUNTER = 0**  **ALARM\_ON = False** |
|  | **Line 48** defines the EYE\_AR\_THRESH . If the eye aspect ratio falls *below* this threshold, we’ll start counting the number of frames the person has closed their eyes for.  If the number of frames the person has closed their eyes in exceeds EYE\_AR\_CONSEC\_FRAMES  (**Line 49**), we’ll sound an alarm.  Experimentally, I’ve found that an EYE\_AR\_THRESH  of 0.3  works well in a variety of situations (although you may need to tune it yourself for your own applications).  I’ve also set the EYE\_AR\_CONSEC\_FRAMES  to be 48 , meaning that if a person has closed their eyes for 48 consecutive frames, we’ll play the alarm sound.  You can make the drowsiness detector *more* sensitive by decreasing the EYE\_AR\_CONSEC\_FRAMES  — similarly, you can make the drowsiness detector *less sensitive* by increasing it.  **Line 53** defines  COUNTER , the total number of *consecutive frames* where the eye aspect ratio is below EYE\_AR\_THRESH .  If COUNTER  exceeds  EYE\_AR\_CONSEC\_FRAMES , then we’ll update the Boolean  ALARM\_ON  (**Line 54**). |

The dlib library ships with a [Histogram of Oriented Gradients-based](https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/) face detector along with a [facial landmark predictor](https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/) — we instantiate both of these in the following code block:

|  |  |
| --- | --- |
| **56**  **57**  **58**  **59**  **60** | **# initialize dlib's face detector (HOG-based) and then create**  **# the facial landmark predictor**  **print("[INFO] loading facial landmark predictor...")**  **detector = dlib.get\_frontal\_face\_detector()**  **predictor = dlib.shape\_predictor(args["shape\_predictor"])** |

The facial landmarks produced by dlib are an indexable list, as I describe [here](https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/):

*Visualizing the 68 facial landmark coordinates from the iBUG 300-W dataset (*[*larger resolution*](https://www.pyimagesearch.com/wp-content/uploads/2017/04/facial_landmarks_68markup.jpg)*).*

Therefore, to extract the eye regions from a set of facial landmarks, we simply need to know the correct array slice indexes:

|  |  |
| --- | --- |
| **62**  **63**  **64**  **65** | **# grab the indexes of the facial landmarks for the left and**  **# right eye, respectively**  **(lStart, lEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["left\_eye"]**  **(rStart, rEnd) = face\_utils.FACIAL\_LANDMARKS\_IDXS["right\_eye"]** |

Using these indexes, we’ll easily be able to extract the eye regions via an array slice.

We are now ready to start the core of our drowsiness detector:

|  |  |
| --- | --- |
|  |  |
| **67**  **68**  **69**  **70**  **71**  **72**  **73**  **74**  **75**  **76**  **77**  **78**  **79**  **80**  **81**  **82** | **# start the video stream thread**  **print("[INFO] starting video stream thread...")**  **vs = VideoStream(src=args["webcam"]).start()**  **time.sleep(1.0)**    **# loop over frames from the video stream**  **while True:**  **# grab the frame from the threaded video file stream, resize**  **# it, and convert it to grayscale**  **# channels)**  **frame = vs.read()**  **frame = imutils.resize(frame, width=450)**  **gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)**    **# detect faces in the grayscale frame**  **rects = detector(gray, 0)** |

On **Line 69** we instantiate our VideoStream  using the supplied --webcam  index.

We then pause for a second to allow the camera sensor to warm up (**Line 70**).

On **Line 73** we start looping over frames in our video stream.

**Line 77** reads the next frame , which we then preprocess by resizing it to have a width of 450 pixels and converting it to grayscale (**Lines 78 and 79**).

**Line 82** applies dlib’s face detector to find and locate the face(s) in the image.

The next step is to apply facial landmark detection to localize each of the important regions of the face:

|  |  |
| --- | --- |
| **84**  **85**  **86**  **87**  **88**  **89**  **90**  **91**  **92**  **93**  **94**  **95**  **96**  **97**  **98**  **99**  **100** | **# loop over the face detections**  **for rect in rects:**  **# determine the facial landmarks for the face region, then**  **# convert the facial landmark (x, y)-coordinates to a NumPy**  **# array**  **shape = predictor(gray, rect)**  **shape = face\_utils.shape\_to\_np(shape)**    **# extract the left and right eye coordinates, then use the**  **# coordinates to compute the eye aspect ratio for both eyes**  **leftEye = shape[lStart:lEnd]**  **rightEye = shape[rStart:rEnd]**  **leftEAR = eye\_aspect\_ratio(leftEye)**  **rightEAR = eye\_aspect\_ratio(rightEye)**    **# average the eye aspect ratio together for both eyes**  **ear = (leftEAR + rightEAR) / 2.0** |

We loop over each of the detected faces on **Line 85** — in our implementation (specifically related to driver drowsiness), we assume there is only *one* face — the driver — but I left this for  loop in here just in case you want to apply the technique to videos with *more than one*face.

For each of the detected faces, we apply dlib’s facial landmark detector (**Line 89**) and convert the result to a NumPy array (**Line 90**).

Using NumPy array slicing we can extract the *(x, y)*-coordinates of the left and right eye, respectively (**Lines 94 and 95**).

Given the *(x, y)*-coordinates for both eyes, we then compute their eye aspect ratios on **Line 96 and 97**.

averaging both eye aspect ratios together to obtain a better estimation (**Line 100**).

We can then *visualize* each of the eye regions on our frame  by using the cv2.drawContours  function below — this is often helpful when we are trying to debug our script and want to ensure that the eyes are being correctly detected and localized

|  |  |
| --- | --- |
| **102**  **103**  **104**  **105**  **106**  **107** | **# compute the convex hull for the left and right eye, then**  **# visualize each of the eyes**  **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)** |

we are now ready to check to see if the person in our video stream is starting to show symptoms of drowsiness:

|  |  |
| --- | --- |
| **109**  **110**  **111**  **112**  **113**  **114**  **115**  **116**  **117**  **118**  **119**  **120**  **121**  **122**  **123**  **124**  **125**  **126**  **127**  **128**  **129**  **130**  **131**  **132**  **133**  **134**  **135**  **136**  **137**  **138** | **# check to see if the eye aspect ratio is below the blink**  **# threshold, and if so, increment the blink frame counter**  **if ear < EYE\_AR\_THRESH:**  **COUNTER += 1**    **# if the eyes were closed for a sufficient number of**  **# then sound the alarm**  **if COUNTER >= EYE\_AR\_CONSEC\_FRAMES:**  **# if the alarm is not on, turn it on**  **if not ALARM\_ON:**  **ALARM\_ON = True**    **# check to see if an alarm file was supplied,**  **# and if so, start a thread to have the alarm**  **# sound played in the background**  **if args["alarm"] != "":**  **t = Thread(target=sound\_alarm,**  **args=(args["alarm"],))**  **t.deamon = True**  **t.start()**    **# draw an alarm on the frame**  **cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),**  **cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)**    **# otherwise, the eye aspect ratio is not below the blink**  **# threshold, so reset the counter and alarm**  **else:**  **COUNTER = 0**  **ALARM\_ON = False** |

On **Line 111** we make a check to see if the eye aspect ratio is below the “blink/closed” eye threshold, EYE\_AR\_THRESH .

If it is, we increment COUNTER , the total number of *consecutive frames* where the person has had their eyes closed.

If COUNTER exceeds EYE\_AR\_CONSEC\_FRAMES  (**Line 116**), then we assume the person is starting to doze off.

Another check is made, this time on **Line 118 and 119** to see if the alarm is on — if it’s not, we turn it on.

**Lines 124-128** handle playing the alarm sound, provided an --alarm  path was supplied when the script was executed. We take special care to create a *separate thread* responsible for calling sound\_alarm  to ensure that our main program isn’t blocked until the sound finishes playing.

**Lines 131 and 132** draw the text DROWSINESS ALERT!  on our frame  — again, this is often helpful for debugging, especially if you are not using the playsound  library.

Finally, **Lines 136-138** handle the case where the eye aspect ratio is *larger* thanEYE\_AR\_THRESH , indicating the eyes are *open*. If the eyes are open, we reset COUNTER  and ensure the alarm is off.

The final code block in our drowsiness detector handles displaying the output frame  to our screen:

|  |  |
| --- | --- |
| **140**  **141**  **142**  **143**  **144**  **145**  **146**  **147**  **148**  **149**  **150**  **151**  **152**  **153**  **154**  **155**  **156** | **# draw the computed eye aspect ratio on the frame to help**  **# with debugging and setting the correct eye aspect ratio**  **# thresholds and frame counters**  **cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),**  **cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)**    **# show the frame**  **cv2.imshow("Frame", frame)**  **key = cv2.waitKey(1) & 0xFF**    **# if the `q` key was pressed, break from the loop**  **if key == ord("q"):**  **break**    **# do a bit of cleanup**  **cv2.destroyAllWindows()**  **vs.stop()** |

### Testing the OpenCV drowsiness detector

|  |  |
| --- | --- |
|  | **$ python detect\_drowsiness.py \**  **--shape-predictor shape\_predictor\_68\_face\_landmarks.dat**  **--alarm alarm.wav** |

Two important computer vision techniques:

* Facial landmark detection
* Eye aspect ratio

[Facial landmark prediction](https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/) is the process of localizing key facial structures on a face, including the eyes, eyebrows, nose, mouth, and jawline.

Specifically, in the context of drowsiness detection, we only needed the eye regions.

*The*eye aspect ratio  to determine if the eyes are closed. If the eyes have been closed for a sufficiently long enough period of time, we can assume the user is at risk of falling asleep and sound an alarm to grab their attention. More details on the eye aspect ratio

## Eye blink detection with OpenCV, Python, and dlib

Our blink detection is divided into four parts.

In the first part- the eye aspect ratio and how it can be used to determine if a person is blinking or not in a given video frame.

Then write Python, OpenCV, and dlib code to

(1) perform facial landmark detection

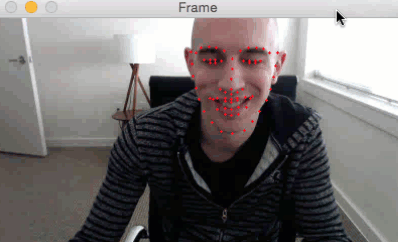
(2) detect blinks in video streams.

Based on this implementation we’ll apply our method to detecting blinks in example webcam streams along with video files.

Finally, methods to improve our blink detector.

### Understanding the “eye aspect ratio” (EAR)

facial landmark detection to localize important regions of the face, including eyes, eyebrows, nose, ears, and mouth:



*Detecting facial landmarks in an video stream in real-time.*

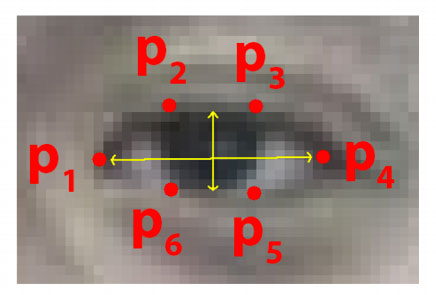
This also implies that we can extract [*specific facial structures*](https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/) by knowing the indexes of the particular face parts:



*Applying facial landmarks to localize various regions of the face, including eyes, eyebrows, nose, mouth, and jawline.*

**In terms of blink detection, we are only interested in two sets of facial structures — the eyes.**

Each eye is represented by 6 (x, y)-coordinates, starting at the left-corner of the eye (as if you were looking at the person), and then working clockwise around the remainder of the region:

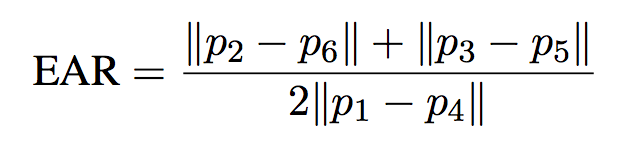


*The 6 facial landmarks associated with the eye.*

Based on this image, we should take away on key point:

**There is a relation between the width and the height of these coordinates.**

[*Real-Time Eye Blink Detection using Facial Landmarks*](http://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf)- we can then derive an equation that reflects this relation called the eye aspect ratio (EAR):



*The eye aspect ratio equation.*

Where p1, …, p6 are 2D facial landmark locations.

The numerator of this equation computes the distance between the vertical eye landmarks while the denominator computes the distance between horizontal eye landmarks, weighting the denominator appropriately since there is only one set of horizontal points but two sets of vertical points.

**What Do You Learn ?**

* Computer Vision(open CV)
* Integration of Camera and capturing live video stream with RPI
* Capturing images using python Programming
* Integrating Speaker with RPI for alarm

**Project Highlights**

* Using Eye- aspect ratio algorithm for drowsiness detection
* Sending Message to the respective authorities to inform his drowsiness condition in the form of speech
* Continuous live video monitoring of driver to prevent accidents.
* Making use of Mobile App to detect Driver’s alertness condition

**Hardware**

* Raspberry pi
* Camera –Logitech

Or

PC or Laptop with inbuilt camera(to update the RAM speed)

**Softwares**

* Python IDE
* Android studio or msg91.com to send messages

**HARDWARES**

**RASPBERRY PI**

**The Raspberry Pi Components**

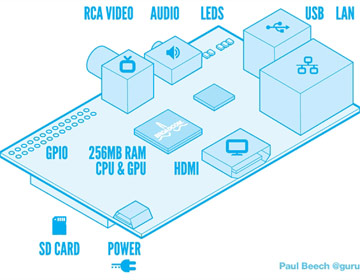


Diagram of the Raspberry Pi

The Raspberry Pi device looks like a [motherboard](https://computer.howstuffworks.com/motherboard.htm), with the mounted chips and ports exposed (something you'd expect to see only if you opened up your computer and looked at its internal boards), but it has all the components you need to connect input, output, and storage devices and start computing.

You'll encounter two models of the device: **Model A** and **Model B**. The only real differences are the addition of [Ethernet](https://computer.howstuffworks.com/ethernet.htm) and an extra USB port on the more expensive Model B.

Here are the various components on the Raspberry Pi board:

* **ARM CPU/GPU** -- This is a Broadcom BCM2835 System on a Chip (SoC) that's made up of an ARM central processing unit (CPU) and a Videocore 4 graphics processing unit (GPU). The CPU handles all the computations that make a computer work (taking input, doing calculations and producing output), and the GPU handles graphics output.
* **GPIO** -- These are exposed general-purpose input/output connection points that will allow the real hardware hobbyists the opportunity to tinker.
* **RCA** -- An RCA jack allows connection of analog TVs and other similar output devices.
* **Audio out** -- This is a standard 3.55-millimeter jack for connection of audio output devices such as headphones or speakers. There is no audio in.
* **LEDs** -- Light-emitting diodes, for all of your indicator light needs.
* **USB** -- This is a common connection port for peripheral devices of all types (including your mouse and keyboard). Model A has one, and Model B has two. You can use a USB hub to expand the number of ports or plug your mouse into your keyboard if it has its own USB port.
* **HDMI** -- This connector allows you to hook up a high-definition television or other compatible device using an HDMI cable.
* **Power** -- This is a 5v Micro USB power connector into which you can plug your compatible power supply.
* **SD cardslot** -- This is a full-sized SD card slot. An SD card with an operating system (OS) installed is required for booting the device. They are available for purchase from the manufacturers, but you can also download an OS and save it to the card yourself if you have a Linux machine and the wherewithal.
* **Ethernet** -- This connector allows for wired network access and is only available on the Model B.

Many of the features that are missing, such as [WiFi](https://computer.howstuffworks.com/wireless-network.htm) and audio in, can be added using the USB port(s) or a USB hub as needed. Next: More details on the device itself and its compatible operating systems.

**Raspberry Pi 3 Model B**

The Raspberry Pi 3 Model B  is a tiny credit card size computer. Just add a keyboard, mouse, display, power supply, micro SD card with installed Linux Distribution and you'll have a fully fledged computer that can run applications from word processors and spreadsheets to games.

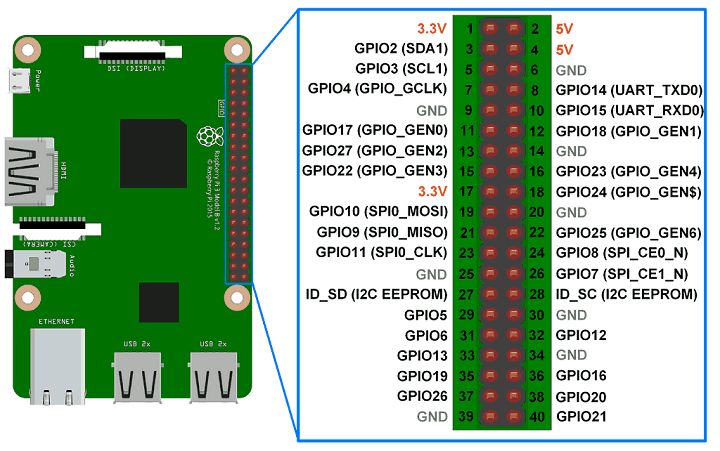
As the Raspberry Pi 3 supports HD video, you can even create a media centre with it. The Raspberry Pi 3 Model B is the first Raspberry Pi to be open-source from the get-go, expect it to be the defacto embedded Linux board in all the forums.

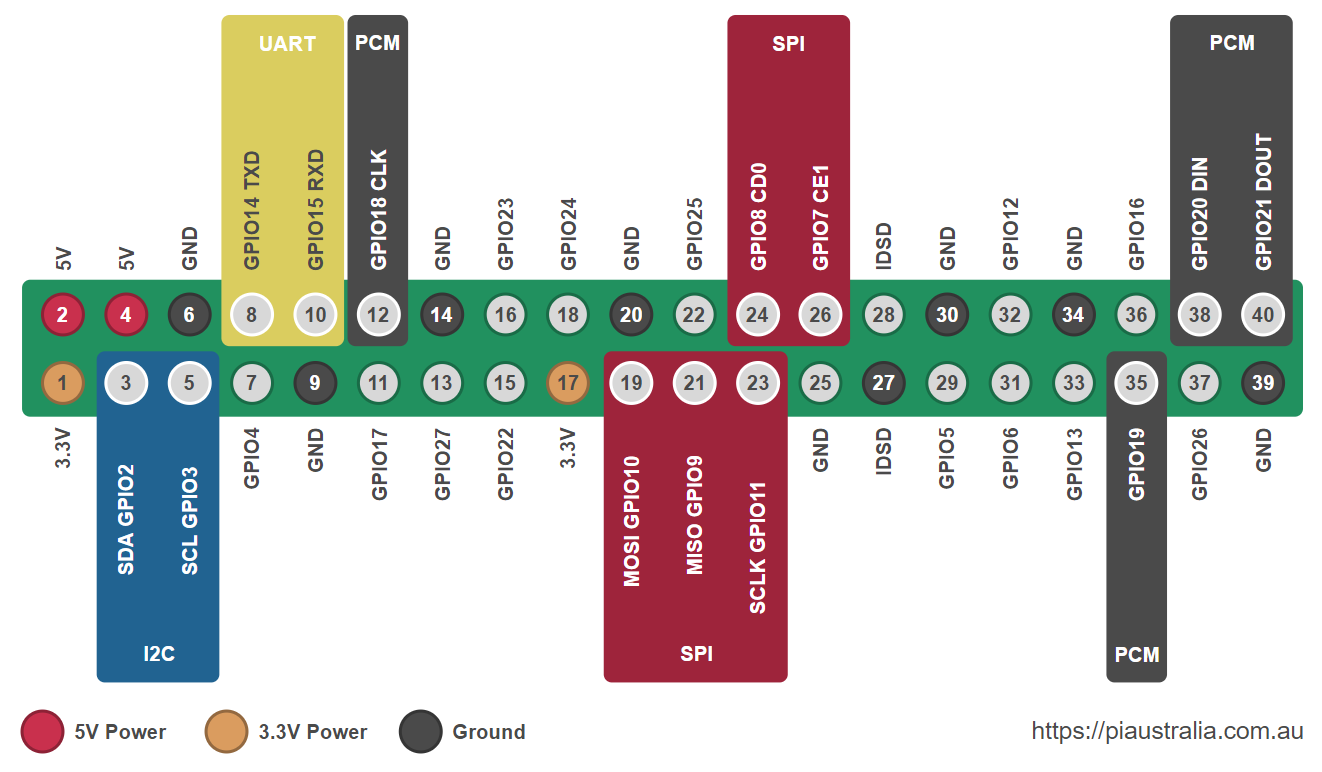
## Tech Specs

|  |  |
| --- | --- |
| Chip | * Broadcom BCM2837 * 64bit * ARMv8 * Quad Core Cortex A53 * 1.2 GHz |
| Storage | microSD Card |
| Memory | 1 GB |
| Graphics | * 400 MHz Dual Core VideoCore IV GPU * OpenGL ES 2.0 * Hardware-Accelerated OpenVG * 1080p30 H.264 high-profile decode |
| Weight | 42g |
| Audio | * HDMI port supports multichannel audio output * Audio line out/3.5-mm headphone jack (analog) |

|  |  |
| --- | --- |
| Connections and Expansions | C:\Users\Allam Sri Sowmya\Pictures\Screenshots\Screenshot (113).png   * Four USB 2.0 ports (up to 480 megabits per second) * HDMI port * 3.5mm 4-pole Composite Video and Audio jack * MicroUSB Power Input * DSI Display Port * CSI Camera Port * MicroSD card Sold * 40-pin GPIO (Male headers) |
| Communications | **Wi-Fi** 802.11n WiFi wireless Networking;IEEE 802.11a/g/b/n compatible  **Bluetooth** Bluetooth 4.1 wireless technology  **Ethernet** 10/100BASE-T Ethernet (RJ-45 connector) |
| Electrical and Operating Requirements | **Input voltage:** 5V DC **Current Requirement:** 2.5 Amps |
| Operating System | **Raspberry Pi Foundation's Offical supported Operating Systems**   * NOOBS * Raspbian   **Third Party Operating Systems**   * Libreelec * Open Elec * OSMC * Pinet * RISC OS * Snappy Ubuntu Core * Ubuntu Mate * Weather Station * Windows IOT Core * XBian |

## GPIO Pin-Out



****

**CAMERA –LOGITECH**

**QuickCam** is a line of **webcam** video **camera** products by **Logitech**. ... The software that originally shipped with the **camera** included QuickMovie for recording motion pictures and QuickPICT for capturing still images.

# LOGITECH HD WEBCAM C270 TECHNICAL SPECIFICATIONS

Top of Form

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | General Product Information: | [[Compliance Certification (CE) Link]](http://www.logitech.com/compliance/) | | | | | | Warranty / Self Help | See [product support page](http://www.logitech.com/support/7079) for warranty duration and frequently asked questions | | | | | |  | **Basic Requirement** | **HD Requirement** |  |  |  | | **System Requirements** | 1GHz = CPU Minimum  1.6GHz = CPU Recommended  1gb = RAM Minimum  2gb = RAM Recommended | 2.4 GHz Intel Core 2 Duo = CPU Recommended 2 GB RAM |  |  |  | | **Software Support (at release)** | Logitech Webcam Software 2.0 (LWS) (NOTE: Check website for the latest software release) | |  |  |  | | **OS Support (at release)** | Windows XP, Windows Vista, Windows 7 | |  |  |  |      |  |  | | --- | --- | | Camera Specifications: |  | | **Available Image(s)** | [[Left Side Image]](https://secure.logitech.com/assets/31706/c260-left-side-image.jpg) | | **Connection Type** | Corded USB | | **USB Type** | High Speed USB 2.0 | | **USB VID\_PID** | VID\_046D&PID\_081A | | **Microphone** | Built-in, Noise Supression | | **Lens and Sensor Type** | Plastic | | **Focus Type** | Fixed | | **Field of View (FOV)** | 60° | | **Focal Length** | 4.0 mm | | **Optical Resolution (True)** | 1280 x 960 1.2MP | | **Image Capture (4:3 SD)** | 320x240, 640x480 1.2 MP, 3.0 MP | | **Image Capture (16:9 W)** | 360p, 480p, 720p | | **Video Capture (4:3 SD)** | 320x240, 640x480, 800x600 | | **Video Capture (16:9 W)** | 360p, 480p, 720p, | | **Frame Rate (max)** | 30fps @ 640x480 | | **Video Effects (VFX)** | N/A | | **Right Light** | Right Light 2 | | **Buttons** | Other NA | | **Indicator Lights (LED)** | Activity/Power | | **Privacy Shade** | No | | **Clip Size (max)** | 0 to infinity | | **Cable Length** | 5 Feet or 1.5 Meters |      |  |  | | --- | --- | | Additional Hardware in Package: |  | | **Headset** | Not Included | | **Desktop Stand** | Not Included | |

**SOFTWARES**

**PYTHON IDE**

**IDLE** (short for integrated development environment or integrated development and learning environment) is an integrated development environment for **Python**, which has been bundled with the default implementation of the language since 1.5.2b1. ... **Python** shell with syntax highlighting.

**Python IDEs** For Data Science. ... **IDE** stands for Integrated Development Environment. It's a coding tool which allows you to write, test, and debug your code in an easier way, as they typically offer code completion or code insight by highlighting, resource management, debugging tools,…

Writing Python using IDLE or the Python Shell is great for simple things, but those tools quickly turn larger programming projects into frustrating pits of despair. Using an IDE, or even just a good dedicated code editor, makes coding fun—but which one is best for you?

Fear not, Gentle Reader! We are here to help explain and demystify the myriad of choices available to you. We can’t pick what works best for you and your process, but we can explain the pros and cons of each and help you make an informed decision.

To make things easier, we’ll break our list into two broad categories of tools: the ones built exclusively for Python development and the ones built for general development that you can use for Python. We’ll call out some Whys and Why Nots for each. Lastly, none of these options are mutually exclusive, so you can try them out on your own with very little penalty.

## What Are IDEs and Code Editors?

An IDE (or Integrated Development Environment) is a program dedicated to software development. As the name implies, IDEs integrate several tools specifically designed for software development. These tools usually include:

* An editor designed to handle code (with, for example, syntax highlighting and auto-completion)
* Build, execution, and debugging tools
* Some form of source control

Most IDEs support many different programming languages and contain many more features. They can, therefore, be large and take time to download and install. You may also need advanced knowledge to use them properly.

In contrast, a dedicated code editor can be as simple as a text editor with syntax highlighting and code formatting capabilities. Most good code editors can execute code and control a [debugger](https://realpython.com/python-debugging-pdb/). The very best ones interact with source control systems as well. Compared to an IDE, a good dedicated code editor is usually smaller and quicker, but often less feature rich.

## Requirements for a Good Python Coding Environment

So what things do we really need in a coding environment? Feature lists vary from app to app, but there are a core set of features that makes coding easier:

* **Save and reload code files**  
  If an IDE or editor won’t let you save your work and reopen everything later, in the same state it was in when you left, it’s not much of an IDE.
* **Run code from within the environment**  
  Similarly, if you have to drop out of the editor to run your Python code, then it’s not much more than a simple text editor.
* **Debugging support**  
  Being able to step through your code as it runs is a core feature of all IDEs and most good code editors.
* **Syntax highlighting**  
  Being able to quickly spot keywords, variables, and symbols in your code makes reading and understanding code much easier.
* **Automatic code formatting**  
  Any editor or IDE worth it’s salt will recognize the colon at the end of a while or forstatement, and know the next line should be indented.

Of course, there are lots of other features you might want, like source code control, an extension model, build and test tools, language help, and so on. But the above list is what I’d see as “core features” that a good editing environment should support.

With these features in mind, let’s take a look at some general purpose tools we can use for Python development.

**ANDROID STUDIO**

**Android studio** is the official IDE (Integrated Development Environment) or tool (layman terms) for developing application exclusively for **Android** platform. It has a strong editor tool for developing creative UI and emulators for different versions to test and simulate sensors without having actual **Android** devices.

Android Studio is Google’s Official Software OR **Ide(Integrated Development Environment)**to Develope android Apps.

Android studio gives us a proper environment for developing and for coding of android apps

right now their are two language in which android native apps are developed

1. Java
2. Kotlin

but most of the apps available in play store is developed in java only, because kotlin is a new language comes in android development.

Bottom of Form

Top of Form

Bottom of Form

**Msg91.com**

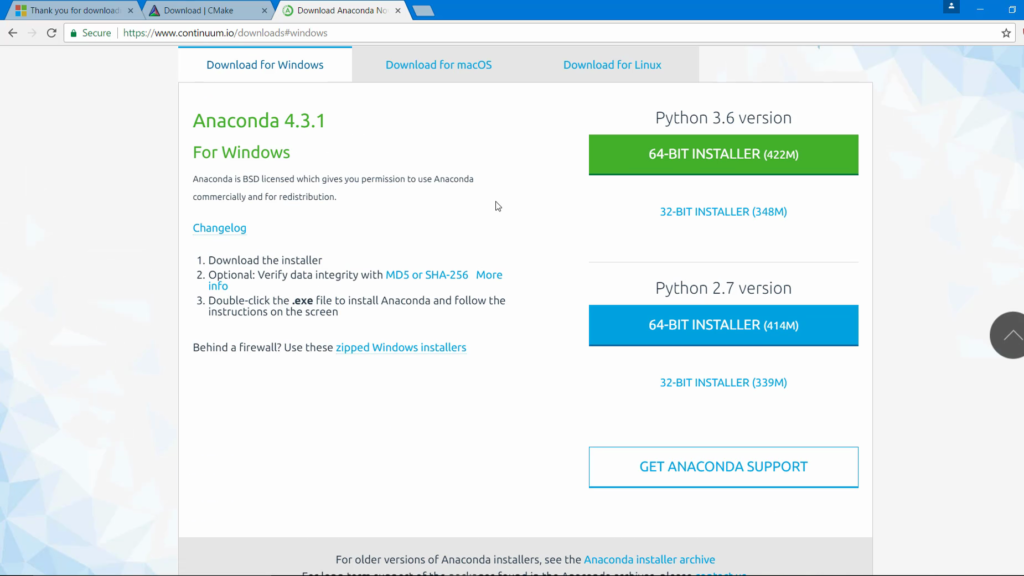
MSG91 is an enterprise SMS solutions provider that is on a mission to elevate the standard of express SMS services through the world's best panel and a robust mechanism. MSG91 gives you the power to boost the way your business communicates.

**INSTALLATIONS , INSTALLATION PROCESS**

## Install Anaconda (a python distribution)

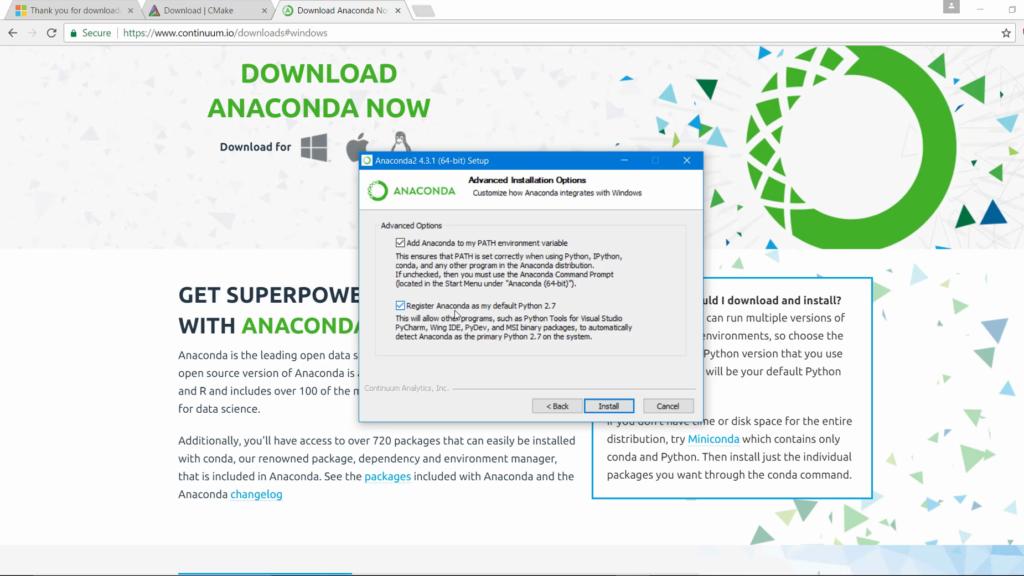
Download and install Anaconda 64-bit version from <https://www.continuum.io/downloads>.

It is advised to install Anaconda for Python 3.

[](https://www.learnopencv.com/wp-content/uploads/2017/05/opencv_0010.png)

While installing Anaconda make sure that you check both options:

1. Add Anaconda to my PATH environment variable
2. Register Anaconda as my default Python

[](https://www.learnopencv.com/wp-content/uploads/2017/05/opencv_0016.png)

# OPENCV 3 AND DLIB INSTALLATIONS

# Open anaconda powershell prompt(administrator)

# 1.conda

# 2.pip install opencv-python

# 3.python

# -import cv2

# 4.conda install –c anaconda pip

# 5.pip install imutils playsound numpy scipy matplotlib scikit-learn jupyter

**6. pip install opencv-contrib-python**

**7.pip install dlib**

**8.conda install –c conda-forge dlib=19.4**