**EXAM PROCTORING SYSTEM**



**Amrita School of Engineering, Coimbatore**

**Department of Computer Science and Engineering**

**BTech/III Year CSE/ VI Semester**

**15CSE336/ BIOMETRICS**

**FINAL CASE STUDY REVIEW**

**TECHNICAL REPORT**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **ROLL NUMBER** | **NAME** |
| 1. | CB.EN.U4CSE18302 | Achanta Ramya Sri |
| 2. | CB.EN.U4CSE18310 | Chandravadhana. A |
| 3. | CB.EN.U4CSE18314 | Dhivakar K |
| 4. | CB.EN.U4CSE18328 | Juvvala Hemantha Sai Sandhya |
| 5. | CB.EN.U4CSE18333 | K Akhila Kumari |

**ABSTRACT**

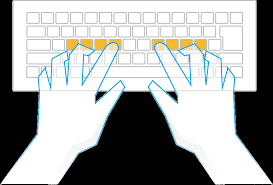
The ongoing pandemic has forced the education sector to plan continuity in learning and exams. Therefore, it is necessary to proctor and monitor the students.

The project aims to use face and eye movement to ensure protection from malpractices.

The monitoring system detects if the student moves the face away from the screen or sees anywhere other than the screen.

Traits Used:

1. Keystroke dynamics
2. Face and Eye movement





**KEYSTROKE DYNAMICS MODULE**

**INTRODUCTION**

Keystroke dynamics basically deals with the analysis of keystroke patterns of a user. It is classified as a **behavioural biometric.** In one of its many use cases, keystroke dynamics can enable proctored exams to provide ***non-intrusive real-time authentication of users.***

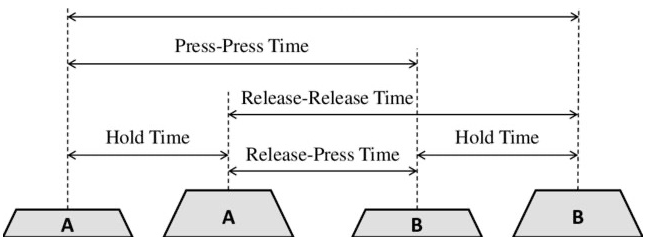
Capturing the duration between 2 keypresses, the duration of pressing down a key, and the duration between the current key release and the next key press can provide insights about the user.

The next time the user logs in, by comparing his/her current typing pattern with his/her previous typing patterns, the platform can authenticate whether the logged in user is legitimate or fraudulent.

It is a classification based problem, where different typing patterns have been matched and clustered under the same umbrella.

**ABOUT THE TECHNIQUE – GENERATE FEATURES**

* Usually a keystroke logger would just log the timestamps when the key is pressed and released.
* Now, by itself, these timestamps would mean nothing.
* However, insights can be gained on user typing patterns by creating features such as:
  1. Press-press duration (PPD)
  2. Hold duration (HD)
  3. Release-press duration (RPD)
* The image shows the duration calculations for pressing 2 keys — A and B.



* Here, the smaller keys represent the key press event and the larger ones represent the key release event. These durations can be used in understanding the user’s typing patterns.
* Here,
  1. The smaller keys: key press event.
  2. The larger keys: key release event.
     + These durations can be used in understanding the user’s typing patterns.
* Press-Press Duration(PPD) 🡪 pressB - pressA
* Release-Press Duration (RPD) 🡪 pressB - releaseA
* Hold Duration (HD) -> releaseA 🡪 pressA
* PPD and RPD are duration between keys whereas HD is for each individual key.

**PART 1: DESCRIPTION OF THE DATASET**

* The dataset: Keystroke dynamics challenge 1 | Kaggle has been used in the jupytr notebook.
* This dataset captures typing attempts of **110 users.**
* Each user has attempted **8 times** to type the string ‘United States’ and the corresponding **timestamps** of key press and release relative to the first key press have been captured.
* Shape of training dataset: 880 X 27.
* Number of users for which training data is present: 110.
* Shape of test dataset: 220 X 26.
* The target variable is: “User”.

**PART 2: DATA PRE-PROCESSING**

* **Primary Observation:** Checking for the row count and data types.
  + **Inference:** All features are numerical data type.
* **Distribution:** Using the .describe()
  + **Inference:** count, mean, standard deviation, minimum and maximum.
* **Checking for missing values**: Using .insull().sum()
  + **Inference:** There are no missing values.

**PART 3: DEMONSTRATION**

* This data can’t be used directly.
* **Feature generation** is done:
  + Press-press, hold, release-press durations from this dataset.
* Taking a sample of 5 users but this data can’t provide useful information directly. More useful insights can be drawn only from the **time durations** between these timestamps.
* 5 x 8 = 40 typing patterns.
* **PRESS – PRESS DURATION**
  + Time between 2 continues presses.
* **RELEASE PRESS DURATION**
  + Time between a release of 1 key and press of other.
* **HOLD DURATION**
  + Time between a press and release of a key.

**PART 4: EXPLORATORY DATA ANALYSIS (EDA)**

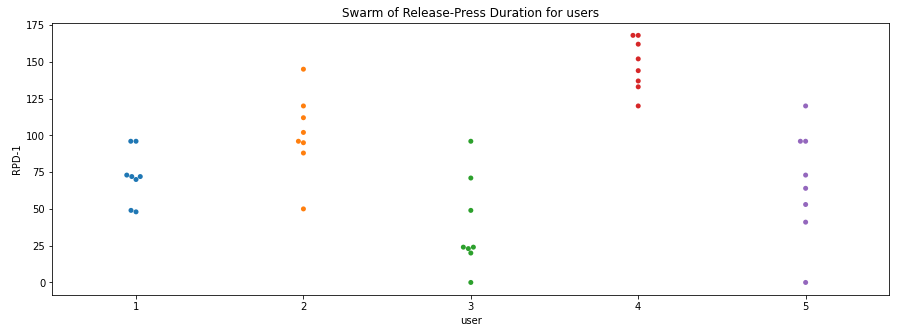
* Explored this generated data and tried to understand the data by its face value.
* Before further analysing the code, considered only 5 of the 110 users i.e. 5 x 8 = 40 typing patterns.

**WHAT IS SWARM PLOT?**

* Swarm plots depict all the data points. Swarm plots attempt to avoid obscuring points by calculating non-overlapping positions instead of adding random jitter.
* Using seaborn’s *swarm plot* function, we can generate swarm plots of release-press, press-press, and hold duration for these 5 users.

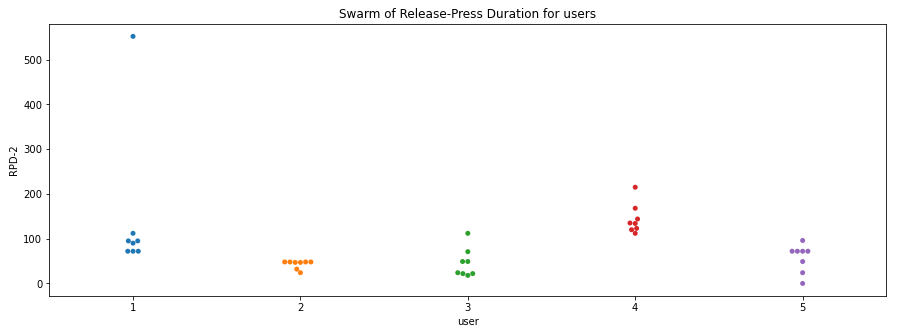
**PART 5: INFERENCE FROM SWARM PLOT**

**RPD time for pressing keys 'u' and 'n':**



* Analysing only two keys/letters ‘u’ and ‘n’. So duration between two key presses for 8 times each user taken for analysis. Therefore 8 data points for every user plotted on graph.
* Initially, only two keys/letters were analysed 🡪 ‘U’ and ‘N’. So two key presses were made 8 times for analysis i.e 2 X 8 = 16.

**RPD time for pressing keys 'n' and 'i':**

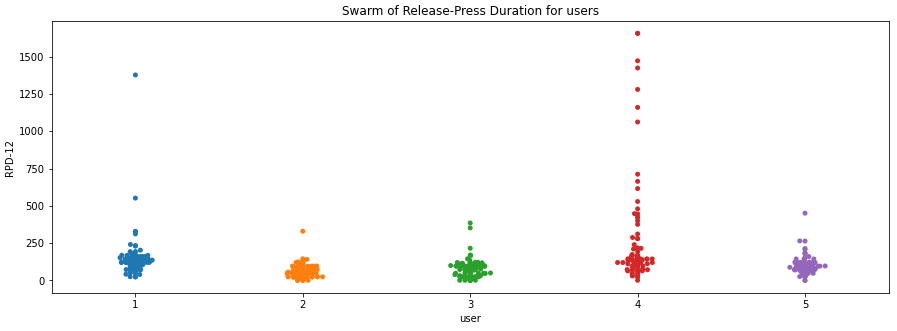


* Similarly, the second graph two keys/letters were analysed 🡪 ‘N’ and ‘I”.
* From the above two graphs, the overall typing pattern is not inferred because it is necessary to take his/her typing data for all the letters into consideration for coming to conclusions.

**Overall Plot for the full text over 8 repetitions:**

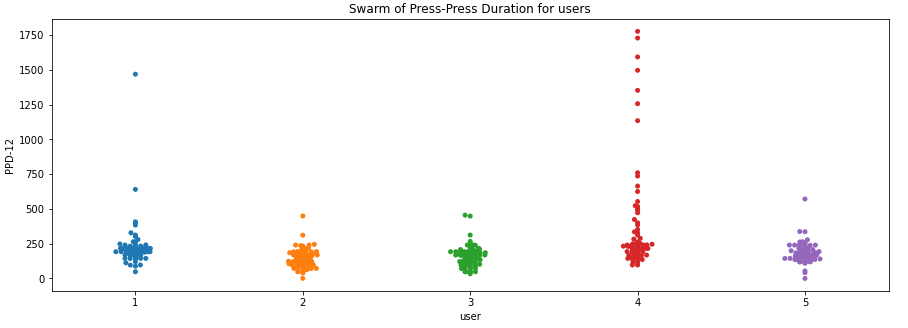
* Therefore, all 13 keys/letters were analysed 🡪 ‘United States” i.e 13 X 8 = 104 data points for each swarm plot.

***Release Press Duration:***



**Inference:**  
We can see RPD for some users is different but for some is similar so RDP cannot solely identify the user.

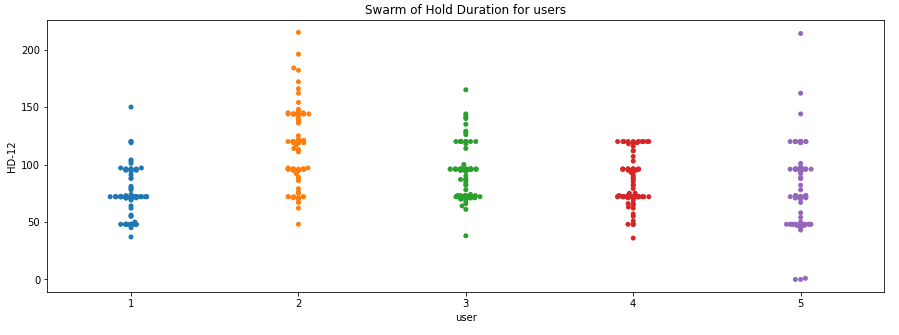
***Press Press Duration:***



**Inference:**  
Similarly PPD also fails to distinguish properly.

* As seen from the swarm plots, the press-press duration, release-press durations are roughly the same across all users. Therefore, cannot be a metric to distinguish.
* Thus, directly using an average duration will not be helpful.

***Hold Duration:***



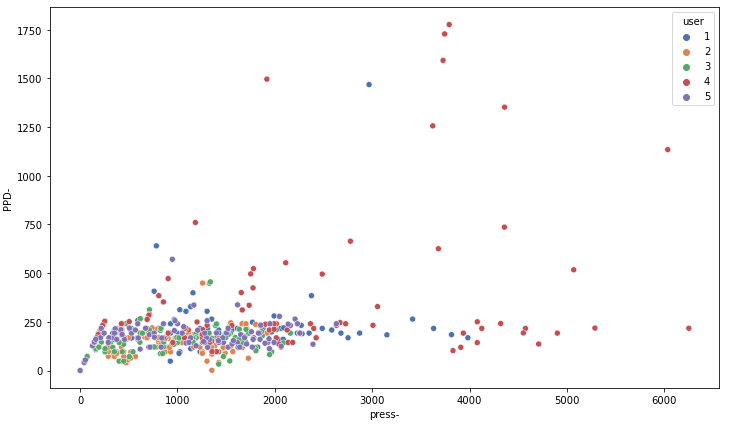
**Inference:**  
Now, Hold duration shows a different analysis. Can be inferred that people tend to differ in their hold duration which means people take different time duration to press a particular key.

* However, hold duration is roughly different for each user which is correct since each user has a different typing speed according to his/her familiarity with typing.
* Therefore, using histograms to check if any variations could be identified.
* Since, each typing pattern consists of 13 keystrokes, scatter plots and line plots can be used.

**PART 6: CONVERTING ROW FEATURES TO COLUMN FEATURES FOR PLOTTING**

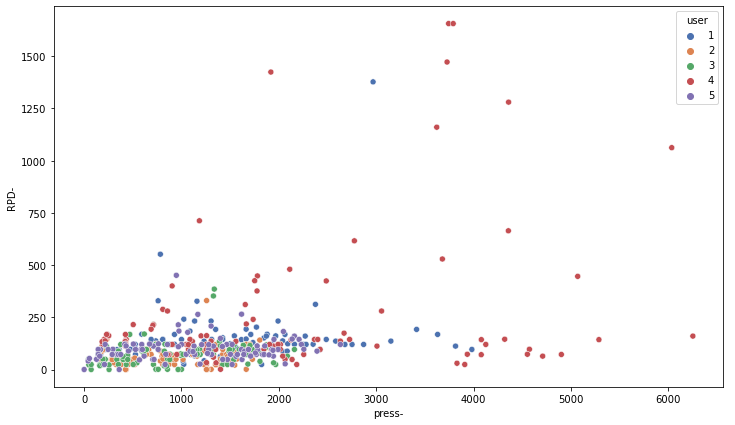
* Now, each row in the dataset is a typing pattern corresponding to a user, but if analysing the typing patterns across the users is the aim (timestamps connected by lines), there is a need to bring these row features into a single column.
* For this purpose, along with few other pandas functions, the *wide\_to\_long* feature of pandas is used.
* For a particular user, there are 12 X 8 times (=96), the typing analysis is done. Therefore, for 5 users: 96 X 5 times (=480). Hence, the shape is 480 X 3.
  + So, for user: one, the iloc values ranges from 0:15, where each row is the analysis between two consecutive keys/letters. Since there are 13 keys in “United states”, we get 12 comparisons.
  + Similarly for user: 5, the iloc values ranges from 32:29.

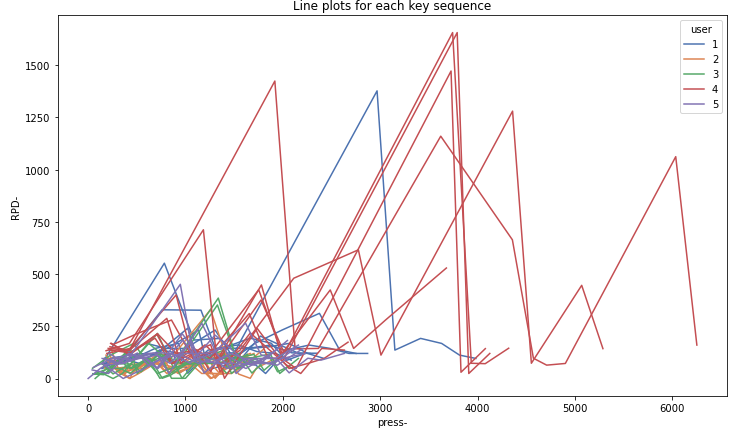
**PART 7: PLOTTING PRESS, PRESS DURATION VS PRESS TIMESTAMPS USING PLOTS AND LINES**



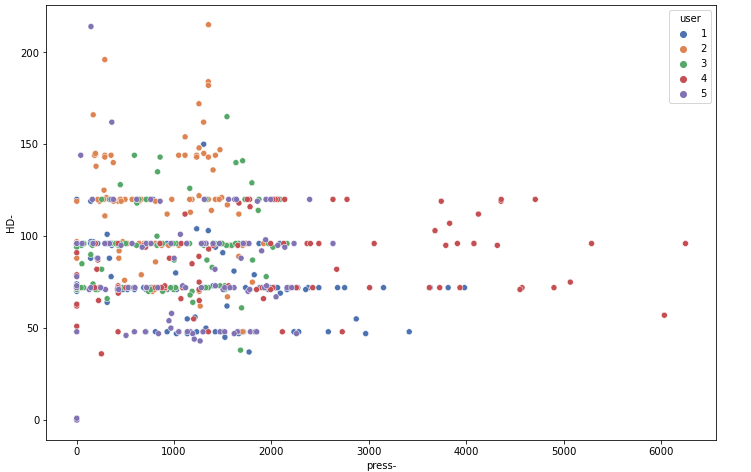


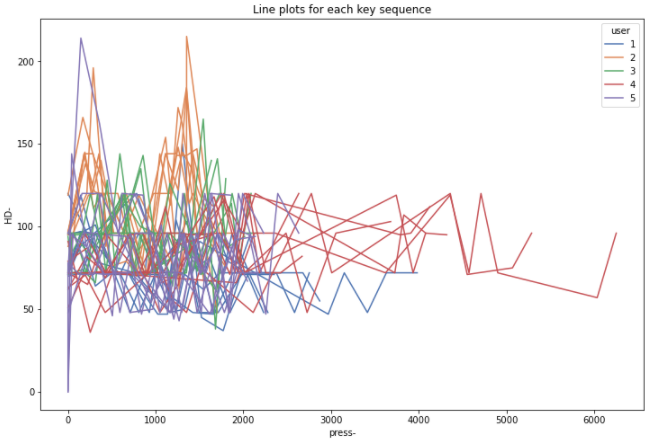
* Here there are 40 line plots i.e. 8 line plots per user and 5 users are being considered.
* Notice the jagged lines for user 4 (red lines).
  + The PPD suddenly increases and then becomes very low for the next key.
  + This means that this user waits for a relatively longer time before typing in 2 keys back to back.
  + So, it could be said that this user typically types in groups of 2 keys.
* Now, imagine the level of security if we use sophisticated algorithms to generate insights and devise authentication techniques.
* This would be something that the user wouldn’t even need to remember.





For RPD and PPD 🡺 Similar graphs





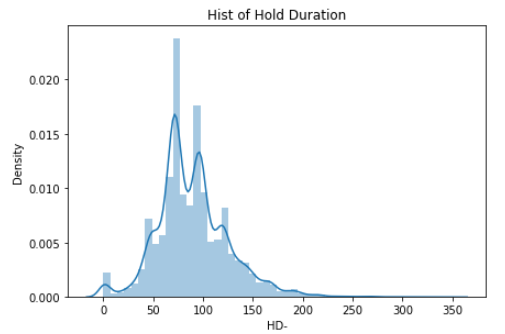
HD 🡪 Hold duration for each key does not have any common pattern.

**PART 8: FACTORS IMPACTING THE TYPING PATTERNS**

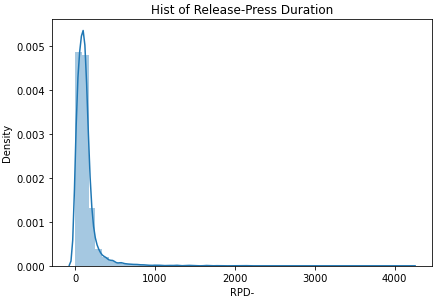
* Time of day for typing.
* Lighting condition of keyboard while typing (especially for non-touch typers).
* Level of familiarity with the keyboard layout.
* Location where typing is done.
* The arm, sitting position of the typer.
* Stress of the typer.

**PART 9: DATA PREPARATION – FOR COMPLETE TRAINING AND TESTING DATASET 🡪 BUCKETING USING HISTOGRAMS**

* *wide\_to\_long* function transforms the data frame by arranging all the row values of *press-\** into a single column.
* Instead of directly using these durations as inputs, we could group these durations into histograms which would represent groups of different typing speeds.







* For e.g. a slow typer would have his keystroke durations falling in the histogram bucket of larger durations.
* At the same time, a fast typer (or perhaps a touch typer) would have his keystroke durations falling in the histogram bucket of smaller durations.
* Here, we can see that we can easily classify the people in the **higher bucket of values** and **lower buckets of values**.
  + The people in the **Higher bucket of values** take **longer time to type**  🡪  **Slow typers**
  + The people in the **Lower buckets of values** take **shorter time to type** 🡪  **Fast typers**
* Rather than us distinguishing slow typers and fast typers by looking the distribution, we can instead classify them into a range of levels.
* For example, we can divide the above distribution into 10 slots ranging from 0-9, meaning that a person with 0 is the fastest typer of the lot, 5 is typer of an average person and 9 is the slowest typer of the lot.
  + This process is termed as **Binning**
* Thus, we will convert the entire training and test data by using this histogram technique.

**PART 10: VISUALIZATION**

# *Categorize typing style using Binning:*

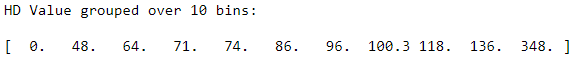
* Now, let’s view the resulting histograms to determine which buckets can be created so as to split the data across them uniformly.
* Instead of determining buckets, a pandas function called   
  “qcut” is used. There is a special function called **'qcut' in pandas** library that performs **binning** into **'q' cuts**.
* In this function we specify the column name over which we want to perform the binning/bucketing.
* This function checks the distribution of values in the specified column and divides the distribution into equal frequencies with q number of bins.
* This ensures that each bin has equal frequency and we can get more meaningful bins.
* In this function we **specify the column name** over which we want to perform the binning/bucketing. This **function checks the distribution of values** in the specified column and **divides the distribution** into **equal frequencies** with **q number of bins**.

**PART 10: BINNING**

* Grouping of categories is done. **Categories (10): [0 < 1 < 2 < 3 ... 6 < 7 < 8 < 9].**
* So, it has grouped the continuous values into bins.

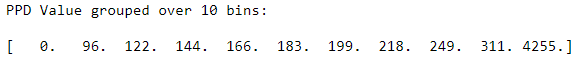
***Done to all Durations:***

* Encoded the values to labels 0–9 in HD\_Bin, PPD\_Bin, RPD\_Bin corresponding to the bins.



This means, our histogram bins for HD are [0,48), [48,64), [64,71) and so on.





* Setting retbins to True so that it can later use these bins over our test dataset.
* For test datasets, since these bins already exist, there is no need to perform qcut. Instead the old cut which directly assigns the values into the appropriate bin is used.
* Since, there are 12 RPD, 12 PPD, 13 HD values corresponding to the 13 key strokes in a typing pattern, the missing RPD, PPD values are filled by -1.

# *Similarly we need to group into buckets in Test Dataset:*

# SWARM PLOTS: Now, the swarm plots of the encodings formed by these buckets can be visualized. In swarm plots, plotted all the encoded values of durations on the graph for each user.

* This indicates the **Average Speed of every user**.
  + For example, from the RPD, PPD durations in the swarm plots can be seen.
* For instance, **User 4** has many encodings in the **higher range of bins (more HD, PPD, RPD)**.
  + At the same time, **User 2** has more encodings in the **lower range of bins**.
* This implies that **User 4 types relatively slowly than User 2**.

# PART 11: PROCESSING

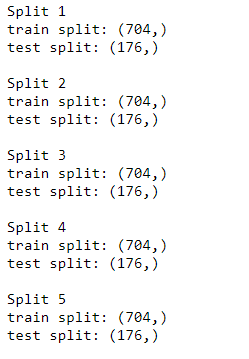
* There are three options based on which the decision can be made on which approach to use.
* Now, for further processing, we have these options:
  1. Aggregate typing patterns of **each user**, consider their **normal distribution** and analyze after putting into **10 bins**.
  2. Aggregate typing patterns of **each user** into **average bin value** for every keystroke.
  3. Retain **all typing patterns** to train model and validate accuracy.
* The 1st approach would be best suitable in cases where the **length of the text** sequences is **not fixed**.  
  For example, you are typing "sun" rather than “united states”, we can compare the time taken with the **rough probabilities for any 2 given keystrokes** (for example, Prob. speed for 'u' and 'n').
* The 2nd approach inherently **considers the slight variations** in **inter - keystroke durations**.  
  That is, when typing 'united states', the **relative position of consecutive pairs of keystroke** on the keyboard might lead to variations in duration. But since 2nd approach takes the **average bin value**, it solves this problem.
* For our dataset, if we use the average bin values for each user, we will have **only 1 value per user for training** which is **very less**. Thus, we will go ahead with the **3rd approach** to just **use the encoded values of all the duration data** for training.

**PART 12: TRAIN THE MODEL**

* Now, previously, in order to determine the bucket value, we had converted the typing patterns (row-wise) into grouped columns.
* But in order to use machine learning techniques like KNN, there is a need to convert these properties/features back into rows. This could help to identify the closest key dynamics signature for a user.

***There is a need for validation dataset***

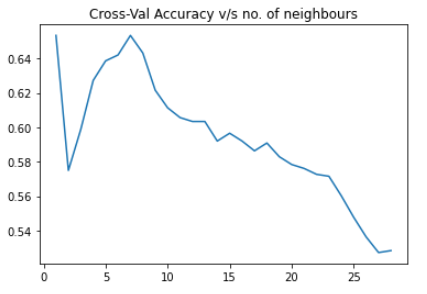
* Our test dataset has only the timestamps without the user id.
* So, in order to validate our models, we need to keep some validation dataset aside from the training dataset.
* We have 110 users but only 8 samples per user, so there is a need to ensure that enough samples per user are present in training set.
* Therefore, during test time, at least few samples must be present for each test user. For this, **StratifiedShuffleSplit** is performed.
* **StratifiedShuffleSplit:** Provides train/test indices to split data in train/test sets. This cross-validation object is a merge of StratifiedKFold and ShuffleSplit, which returns stratified randomized folds. The folds are made by preserving the percentage of samples for each class.

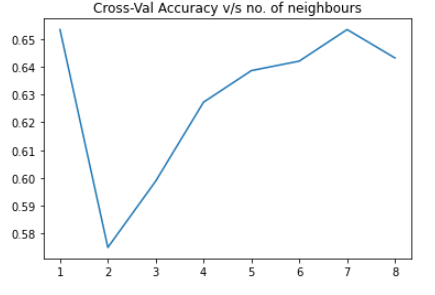


* Here, *n\_splits* is used: as 5 which means 5 different splits will be created (for 5-fold cross validation) and test\_size as 0.2 resulting in (8 x 0.2) = 1.6 test samples for (8 - 1.6) = 6.4 training samples.

**PART 13: KNN (K NEAREST NEIGHBOUR CLASSIFIER)**

* The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both **classification** and **regression** problems.
* It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.





* Finally, 65.3409090909091 % of Accuracy for KNN model with K = 7

**Iris Tracking Module**

The process of measuring either the point of gaze (where one is looking) or the motion of an eye relative to the head. An eye tracker is a device for measuring eye positions and eye movement.

**Packages Used:**

## OpenCv is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

## Dlib is a general purpose cross-platform software library written in the programming language C++, which contains software components for dealing with graphical user interfaces, data structures, linear algebra, machine learning, image processing, data mining, etc.

## Some of the most popular mathematical functions are defined in the math module. These include trigonometric functions, representation functions, logarithmic functions, angle conversion functions, etc. In addition, two mathematical constants are also defined in this module.

## Numpy is a Python library used for working with arrays.It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

**Dataset:**

Helen Dataset was on which the classifier was built upon.

In our effort of building a facial feature localization algorithm that can work dependably and precisely under a wide scope of appearance variation, including pose,lighting, expression, occlusion, and individual contrasts, we understand that it is vital that the raining set include high resolution examples so that, at test time, a high resolution test image can be fit precisely.

Although a number of face databases exist,we discovered none that meet our necessities, especially the resolution requirement.Thus, they developed another dataset using annotated Flickr images.

1. First, a large set of candidate photos was gathered using a variety of keyword searches on Flickr. In all cases the query included the keyword ``portrait'' and was augmented with different terms such as ``family'', ``outdoor'', ``studio'', ``boy'', ``wedding'', etc.
2. The subset was further filtered by hand to remove false positives, profile views, as well as low quality images.
3. For each accepted face, we generated a cropped version of the original image that includes the face and a proportional amount of background. In some cases, the face is very close or in contact with the edge of the original image and is consequently not centered in the cropped image. Also, the cropped image can contain other face instances since many photos contain more than one person in close proximity.
4. Finally, the images were hand-annotated using Amazon Mechanical Turk to precisely locate the eyes, nose, mouth, eyebrows, and jawline.



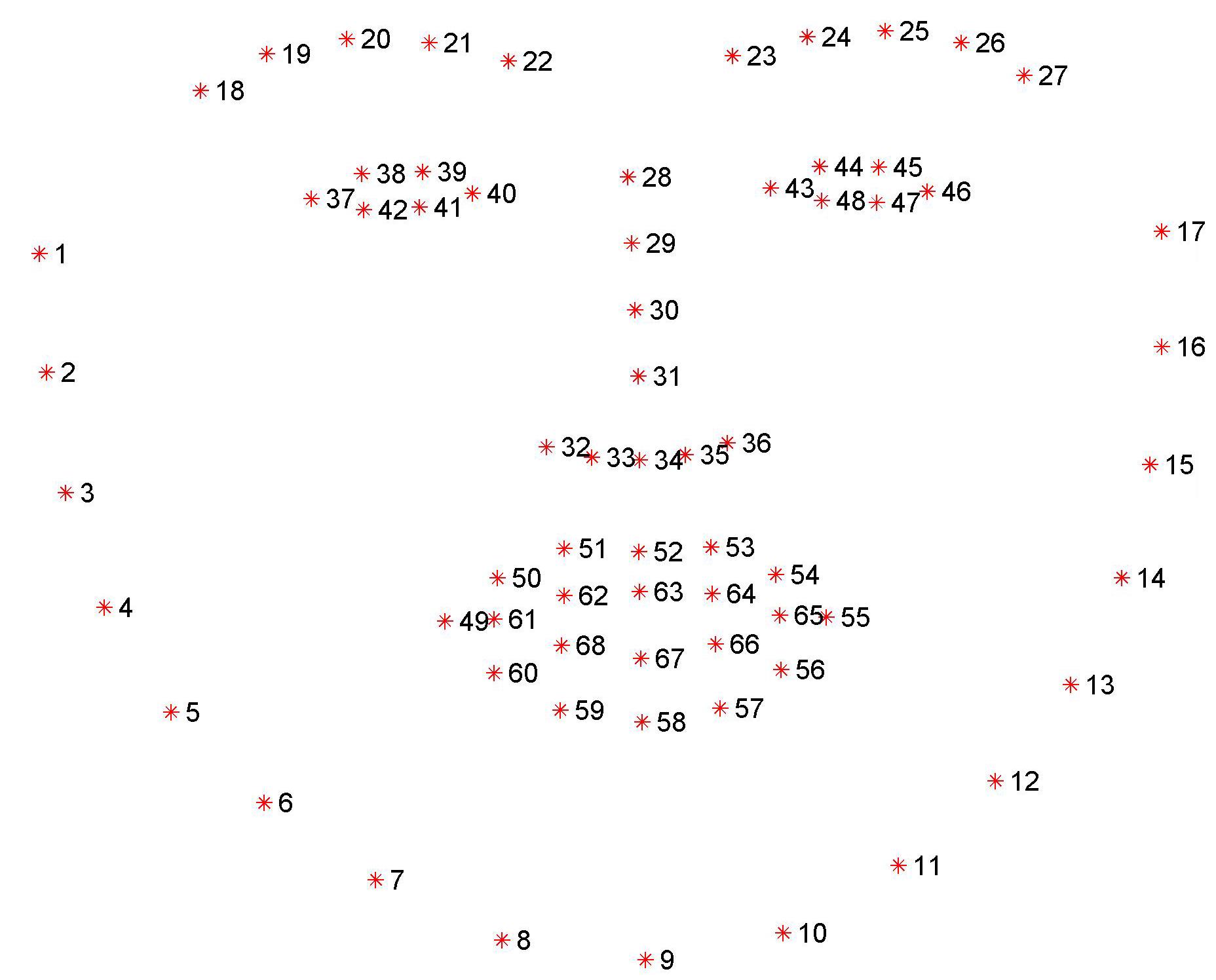


# Facial point annotations:

1. The majority of existing databases provide annotations for a relatively small subset of the overall images.
2. The accuracy of provided annotations in some cases is not so good (probably due to human fatigue).
3. The annotation model of each database consists of a different number of landmarks.

These problems make cross-database experiments and comparisons between different methods almost infeasible. To overcome these difficulties, we propose a semi-automatic annotation methodology for annotating massive face datasets. This is the first attempt to create a tool suitable for annotating massive facial databases.

Link : <https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/>



# Why was Dlib and a pretrained CNN classifier used?

### Dlib is an open source collaborative platform with impressive and efficient tools for image processing/computer vision. One of the reasons for choosing this over native python tools, is because it boosts efficiency and has a lot of prebuilt classifiers with legacy datasets. Its benefits are explained briefly in the following article.

<https://learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/>

### So from the above article, it’s clear that dlib classifiers run the best on CPU and give the best acceptable results. There was an even stronger reason to pick dlib, for its support of premade classifiers, such as the shape predictor we are using in our project.

### The first thing to do is to find eyes before we can move on to image processing and to find the eyes we need to find a face. The facial keypoint detector takes a rectangular object of the dlib module as input which is simply the coordinates of a face.

### As explained in the dataset we used, we clearly established how facial annotation points identify and are indicative of various features of one's face. Dlib’s shape predictor tries to match these facial annotation points onto a subject's face effectively identifying various regions of interest/facial features.

### Using these annotated points, we can clearly identify various regions in the subject's face and at the same time confine the region of interest only to the face of the said individual.

### By using a premade classifier, to archive these results, the following are the reasons:

* To successfully train the model, the minimum recommendation is a system with a cpu having 8 cores. Aside from the other limitations like ram, none of our systems met the minimum requirements for us to successfully train the model.
* The ram limitations posed a problem of higher computation time to train the models and at the same time may cause failures due to memory overhead.
* Aside from the above facts, the trained model would have its source code written and compiled in C++ (since Dlib is written and compiled in c++, however the original source code has been ensured to run in any language; which is something very specific to the libraries and programming constructs used; and dlib is an executable file rather than actual source code that is executed in real time).

**Code Explanation:**

# Class Iris has two main functions:

* Processing the image to effectively mask out the required information, which in this case is Iris.
* Detecting the region of sclera and iris.

## 

## Function pre\_processing:

### The pre-processing function first applies a 3x3 Kernel, Kernels in computer vision are matrices, used to perform some kind of convolution in our data.

### Convolutions are mathematical operations between two functions that create a third function. In image processing, it happens by going through each pixel to perform a calculation with the pixel and its neighbours.

### The kernels will define the size of the convolution, the weights applied to it, and an anchor point usually positioned at the center. So in a 3x3 matrix, each pixel is affected only by the pixels around it.

### So what can kernels do?

* We can filter and modify images by interacting with their pixels;
* That interaction may occur with convolutions;
* Those convolutions use kernels to describe how the pixels will be affected;
* Kernels represent the area for each operation, the values/weights, and the anchor point.

### After that bilateral filter is applied to the frame(video is made up of images, a frame is one of such images). A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images. It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels(this neighbourhood is defined by the kernel). This weight can be based on a Gaussian distribution. Crucially, the weights depend not only on Euclidean distance of pixels, but also on the radiometric differences (e.g., range differences, such as color intensity, depth distance, etc.). This preserves sharp edges. It is very important to preserve these sharp edges because only then the color intensity difference between the sclera and iris becomes evident.

## The bilateral filter is defined as

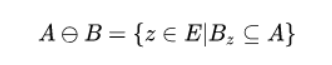
image.png

### Following the bilateral filter, we perform a morphological process called erode. Erosion is one of two fundamental operations (the other being dilation) in morphological image processing from which all other morphological operations are based. It erodes the boundaries of the foreground object (Always try to keep the foreground in white).

### So what does it do? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero). So what happens is that all the pixels near the boundary will be discarded depending upon the size of the kernel.

### So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in the colorspace chapter), detaching two connected objects etc.

## The erosion of the binary image A by the structuring element B is defined by:



### Finally the image is thresholded, to cleary distinguish the iris from the sclera. In digital image processing, thresholding is the simplest method of segmenting images. From a grayscale image, thresholding can be used to create binary images. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is cv2.threshold. First argument is the source image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value.

## Function detect\_iris:

### This function first calls the processing function to receive the image of an eye alone.

### It's followed by finding contours to distinguish the iris from the sclera. Contours are defined as the line joining all the points along the boundary of an image that are having the same intensity. Contours come handy in shape analysis, finding the size of the object of interest, and object detection. The intensity difference is between the sclera(white part of the eye) and iris(which is a grey/black color, since the image is in greyscale).

### OpenCV has a findContour() function that helps in extracting the contours from the image. It works best on binary images(which is solved by thresholding the image in the previous stage), so we should first apply thresholding techniques, Sobel edges, etc.

### Then these contours are sorted w.r.t. to area to better represent this in array form. By arranging the contours w.r.t area, the largest contour indicates the entire iris. Thats what is done in the try block to extract this region of the largest non-white region within an eye. This largest non-white region is actually the iris.

### By performing the above functions an image of an eye is processed and the iris within it is detected.

# Class Eye has three main functions:

* \_middle\_point which returns the center of the eye
* \_isolate has one primary function, i.e. isolate the portion of the eye from the rest of the image.
* \_analyse on the other hand which eye it is, whether its left or right. and then with this information drives the above two codes and also calls the Threshold to identify the lighting situation and return an optimal threshold value. No further explanation is required for this function.

## Function \_middle\_point:

### It is the most straightforward function with only one objective, given the least x coordinate and largest x coordinate, it returns the mean or effectively the center of the iris's x coordinate. The same is repeated for the y coordinates as well.

## Function \_isolate:

### By far this is the most important function in this project. It tries to identify the region of the eye, from the entire frame. This means that given an eye(left or right), it detects the eye within the face, by matching the facial annotation points mapped to the subject's face with that of regular facial annotation points. These regular points are given by LEFT\_CORDINATES and RIGHT\_CORDINATES, which are part of the 68 landmarks given by the shape predictor.

### So the first thing it does is using list comprehension, reads all the points in the assigned annotation points within LEFT\_CORDINATES or RIGHT\_CORDINATES, So basically an np array representation of these annotation points(X,Y coordinates of these points which are mapped onto the subjects face).

### After that with height and width of the frame, a complete black frame is created using np.zeros. Since images are just matrices filled with 0,1 representing the intensities of image, by filling the entire image with 0, we get a image with RGB #000000(Black). This black frame is used for the computation of masks to perform tasks on specific roi(Regions of interest).

### Then a polygon mask is created using fillpoly, with the points computed earlier. So this effectively is indicative of our regions of interest and these coordinates the corresponding pixels have a value of 1(basically a white region).

### The function cv::fillPoly fills an area bounded by several polygonal contours. The function can fill complex areas, for example, areas with holes, contours with self-intersections (some of their parts), and so forth.

### The next step is the step for which the previous computations have been done. Basically what it does takes a copy of the frame and masks it out using the mask computed in the previous step. So a not operation is done with black frame on top of the frame.copy. Wherever the mask(each pixel is a value in an np.array) has a value of 1, the corresponding pixel in our frame will be visible and will be exempt from the operation of not. This masked frame is then saved on the eye.

### In simpler terms assume you have a printed image and a piece of black paper with the same dimensions of the printout. So let's say you cut out portions of the black paper corresponding to that of the masks. And then when align these two papers, you can only see through this cut out region, and this exactly is what is happening on performing a bitwise not operation on these images. So basically it is an arithmetic operation performed on an image.

### The next lines of code assign the instance variables with rectangular cropped images of only the eye, by figuring out the min and max coordinates in the facial annotation points for x axis and y axis. The origin of the frame is the min coordinates of both the images followed by the center of image computed by the center of height and width of the image.

# Class Threshold has the following functions:

* is\_complete is a function which checks that the length of left and right thresholds are greater than a predetermined constant. No further explanation is required.
* threshold on the other hand computes the mean of left or right threshold values(indicative of left side and right side of the face, ensures both parts of the face are visible). No further explanation is required.
* iris\_size on the other hand computes how many times the black pixels in your iris image are greater than that of the pixels that have data within(which is the iris). This is indicative of the darkest ratio between the darkest point(shadows) from that of brightest point in the image(highlights). No further explanation is required.
* find\_best\_threshold on the other hand iteratively finds out the above ratio within the iris frame, and assigns a threshold value capable of distinguishing the iris from that of the sclera, which happens to be the minimum of above ratios. No further explanation is required.
* evaluate is just a driver function which drives the above functions based on left or right side

# Class IrisTracking is basically like the driver function for the above code for each frame of the image.

## Function pupils\_located:

### The function just validates if the coordinates of respective pupils are integers, effectively also checking if they have some value assigned to them.

## Function \_analyse:

### This function converts the image into grayscale, an unsaid rule in image processing and in any other computation is to not have extra unnecessary information or in better words dimensionality reduction. So until and unless the processing requires colour for its computation, the best way is to avoid the use of colour in general. Apart from that certain functions in the above code are in need of a grayscale image input.

### Then using our illustrious shape predictor of 68 landmarks we detect the face within a frame. It then in the try block, checks if landmarks can be extracted, which are the facial annotation points, and then passes them to detect the left and right eye.

## Function refresh:

### The simplest function so far, it takes the next frame from the video feed to compute the movement of iris within that frame.

## Function pupil\_left\_coords:

### Returns the x and y coordinates of the left eye.

## Function pupil\_right\_coords:

### Returns the x and y coordinates of the right eye.

## Function x\_mean\_displacement:

### Returns the x axis displacement of the left and right iris from its origin

## Function y\_mean\_displacement:

### Returns the y axis displacement of the left and right iris from its origin.

## Function shape\_to\_np:

### Converts the landmark points into its numpy array representation for future computation.

## Function annotated\_frame:

### Returns the frame with points, lines and rectangles printed on top of our input feed to indicate the output of previous functions.

## Function direction:

### Based on the x\_mean\_displacement and y\_mean\_displacement it calculates if the given subject is looking up,down,left,right,center or away(face not detected)

### Followed by the above function we set up the previous IrisTracking class and a cv2.VideoCapture object to read the feed from our webcams. In a while loop each frame is passed onto the IrisTracking Class by calling its umpteen number of functions. And then proceeds to give output based on the outputs of the previous classes.

**Advantages of a Iris Tracking System over the alternatives:**

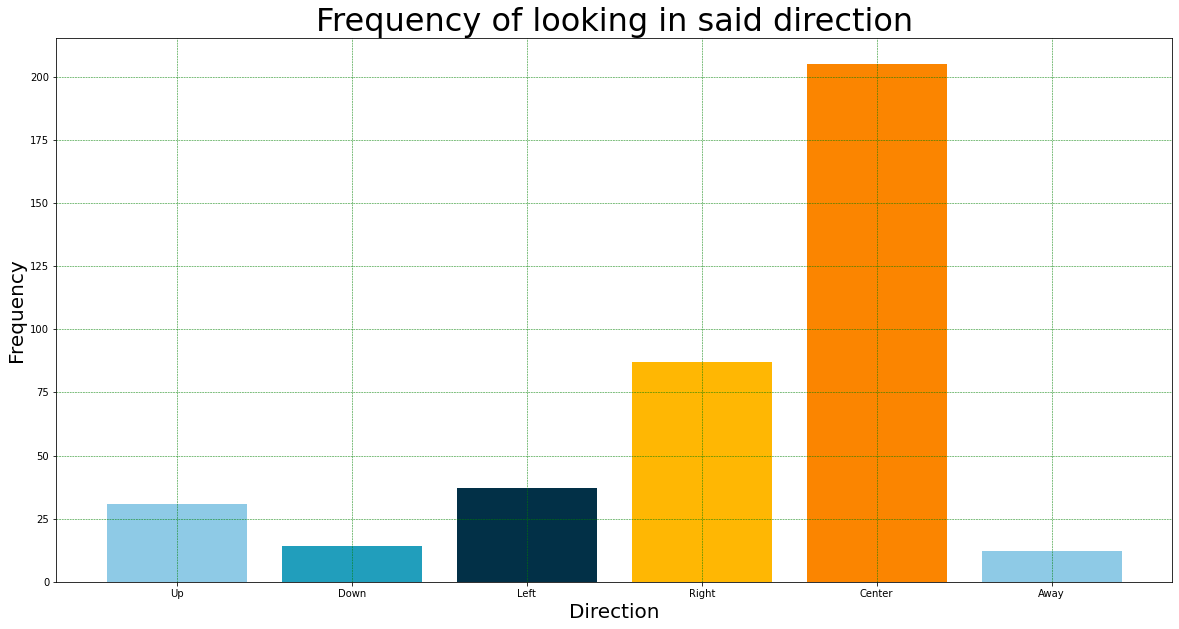
* More Accuracy compared to the alternatives. Since direction and movement are easier to detect with iris. Since, iris can be identified much easier in a 2d plane. This also allows us to easily make a threshold for its x axis and y axis movement and figure out the number of times the subject is looking offscreen.
* More freedom to set rules and regulations. With iris, we can even calculate the time the subject is looking offscreen.
* Also, Possibility of identifying if the person is having an aide within the screen as well, with window snapping.
* Also, can flags users who take a quick glance offscreen, which is often not noticed when the system primarily tracks faces.

**Disadvantages of a Iris Tracking System over the alternatives:**

* Lessprecision, due to the fact that most of the time iris is a very small part of the webcam feed. Unlike that of face, the fraction of iris within a frame is significantly less for iris.
* Quiet taxing on the device it runs on. We figured out even the most efficient algorithms and classifiers required most of the resources of the device. So we assume when running on the web, the device side performance should be good.
* It requires a very stable and fast internet connection. Since iris is very erratic and calculates the results within a frame rather than interframe, it means that it requires a good o=ping or latency for the best results.
* As of now Iris Tracking systems don't have an efficient algorithm for gaze estimation. If gaze estimation is included in iris tracking, the system accuracy may improve.
* Requires a higher fault tolerance than its alternatives.

**Future Scope:**

The best iteration of this device is one that is a multimodal with various other features such as face being tracked. Also it necessitates a way for inter frame algorithms and an effective gaze estimation algorithm. The current estimation technique is a bounding box technique, which is indicative of movement only, and flags the user once the gaze crosses this bounding box..

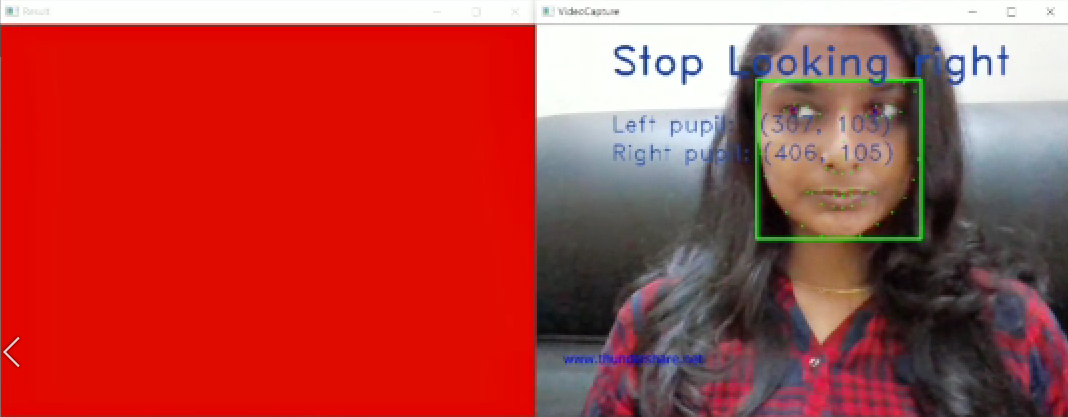


**Demonstration:**

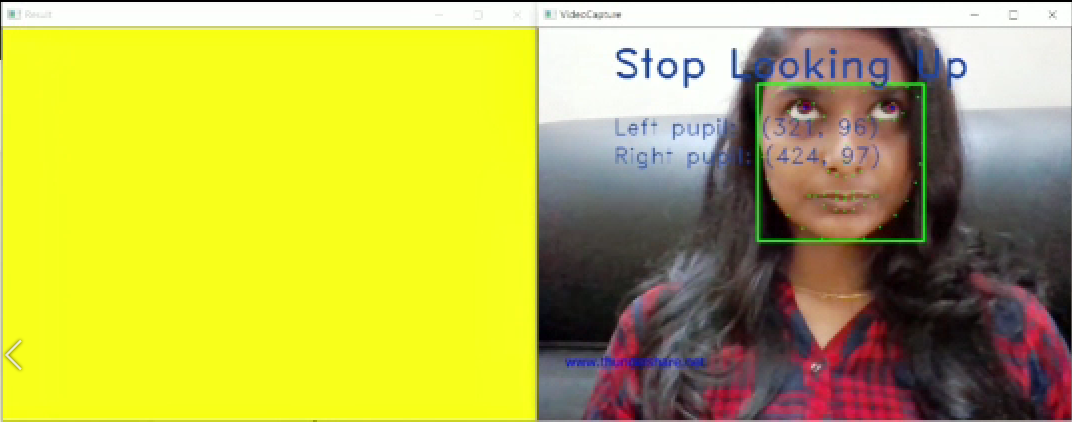
**BLUE-LEFT**



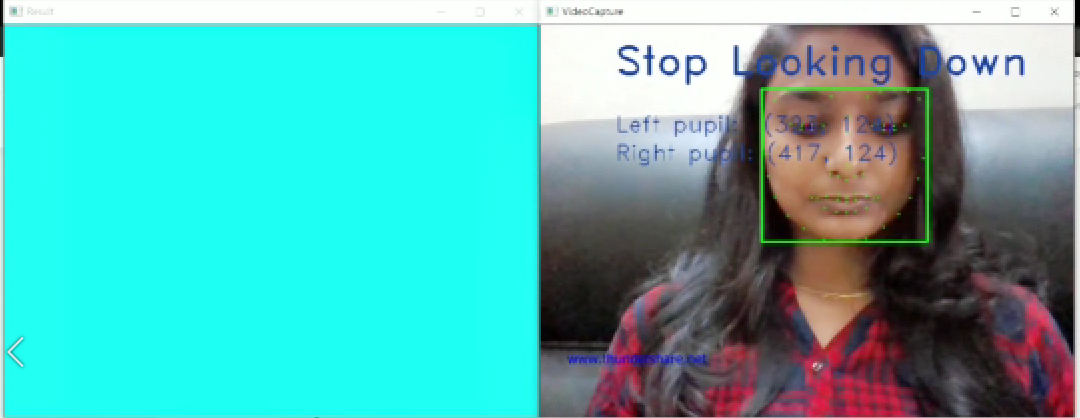
**RED-RIGHT**



**YELLOW- UP**



**LIGHT BLUE- DOWN**



**GREEN-CENTRE**

