APPLICATION OF EYE GAZE DETECTION FOR ALS PATIENTS

A PROJECT REPORT

Submitted by

BL.EN.U4AIE19070 Y. TARAKARAM

BL.EN.U4AIE19071 Y. LAKSHMI PRASANNA

BL.EN.U4AIE19072 Y. MOUNIKA

for the course

19AIE203- Data Structures and Algorithms – 2

Guided and Evaluated by

D. RADHA

Asst. Prof(SG),

Dept. of CSE,

AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDHYAPEETHAM

BANGALORE-560 035

December-2020

TABLE OF CONTENTS

TITLE

APPLICATION OF EYE GAZE DETECTION FOR ALS PATIENTS

ABSTRACT

Amyotrophic lateral sclerosis (ALS), which is also called “Lou Gehrig’s Disease,” is a neurodegenerative disease that attacks nerve cells in the brain and spinal cord. These motor neurons basically control the muscles throughout the body, and their progressive degeneration eventually leads to the death of ALS patients. In the first stages of ALS, patients begin to lose control of their muscle movements as their motor neurons die. In later stages of the disease, voluntary muscle action is affected, and patients may become totally paralyzed. Patients may experience slurred speech, which can become more severe with time, or even completely lose the ability to speak (dysarthria). ALS is a devastating disease that robs patients of their ability to experience life to the fullest.

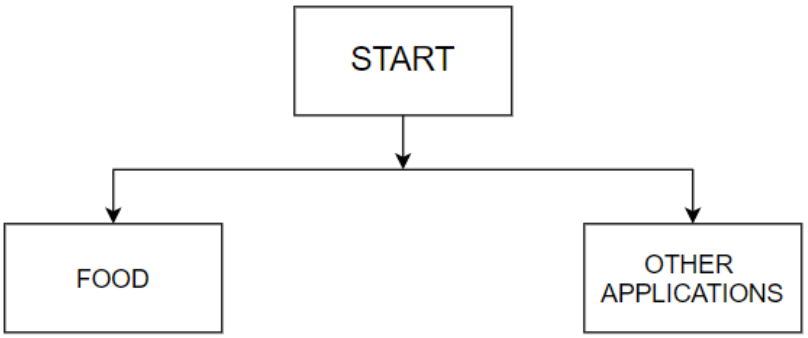
Since there is no technology which can heal the neurons, we can develop some applications for helping these patients to have a better life. So, we have done a small application for these patients. This application works based on the detection of the eye gaze of the patient.

INTRODUCTION

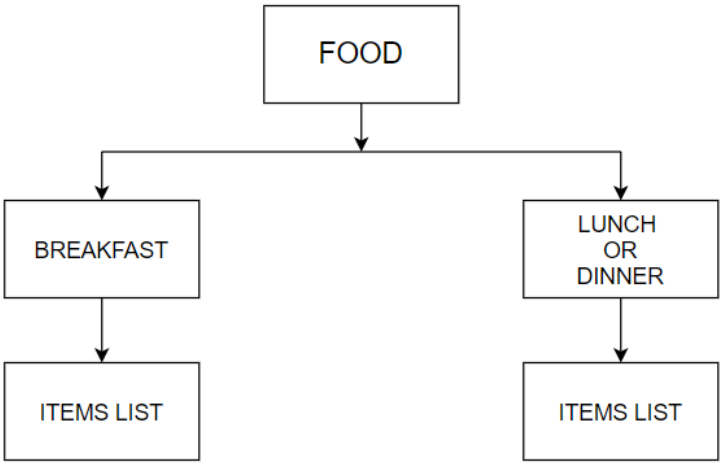
Amyotrophic lateral sclerosis (ALS) is a group of rare neurological diseases that mainly involve the nerve cells (neurons) responsible for controlling voluntary muscle movement. Voluntary muscles produce movements like chewing, walking, and talking. The disease is progressive, meaning the symptoms get worse over time. Currently, there is no cure for ALS and no effective treatment to halt, or reverse, the progression of the disease.Patients may experience slurred speech, which can become more severe with time, or even completely lose the ability to speak (dysarthria). These patients need to communicate with others. Since they cannot speak with their mouth, they can use their eyes for communication. So, based on their eye movement they can communicate with others.

There are applications being developed to make communication easy to these patients. We have built an application for them using gaze detection. The application which we have made makes use for the patients to order the food, listen to songs, watch videos and type something using a keyboard menu. This application mainly depends on the eye movement and also the blinking of the eye.

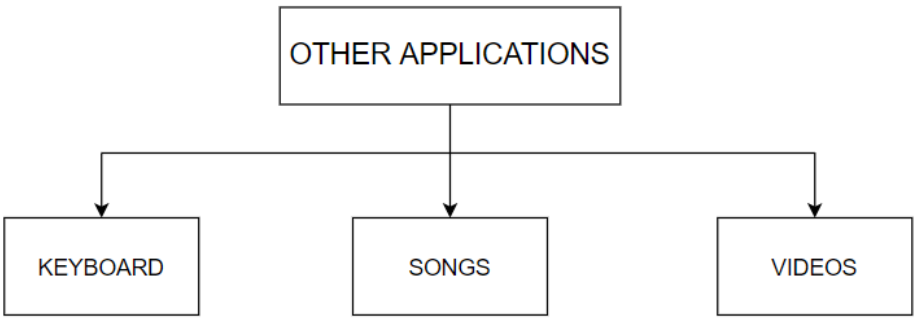
SYSTEM MODEL



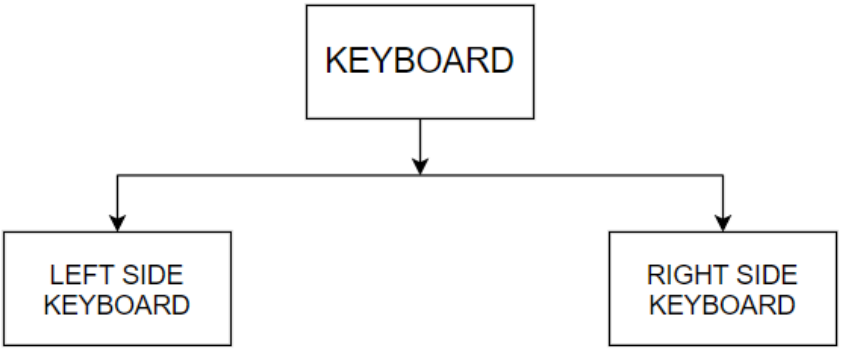
First we will be having two options: food and other applications which are selected based on gaze ratio.



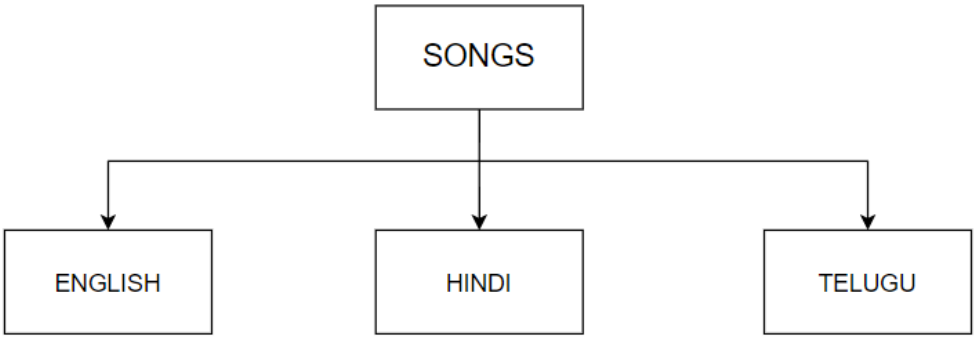
If the food option is selected then we get two other options: breakfast and lunch or dinner which are also selected using a gaze ratio. After selecting you get an items list corresponding to the option selected. Those items are selected using a blinking ratio.



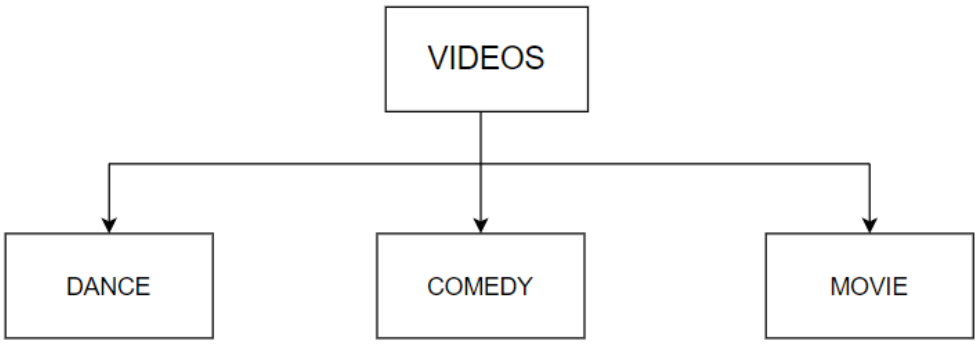
If other application option is selected then we get three options: keyboard, songs and videos. These are selected using a blinking ratio.



If the keyboard option is chosen then we get left and right which are nothing but the left side keyboard and right side keyboard to enter the text. Each letter in the keyboard is selected and typed on the board based on blinking ratio.



If the song option is chosen, we get three language options: English, Hindi and Telugu, which correspond to different songs in the respective language. These are selected using a blinking ratio. After selecting a language you will get different songs corresponding to the language selected. Each song can be selected and played using a blinking ratio.



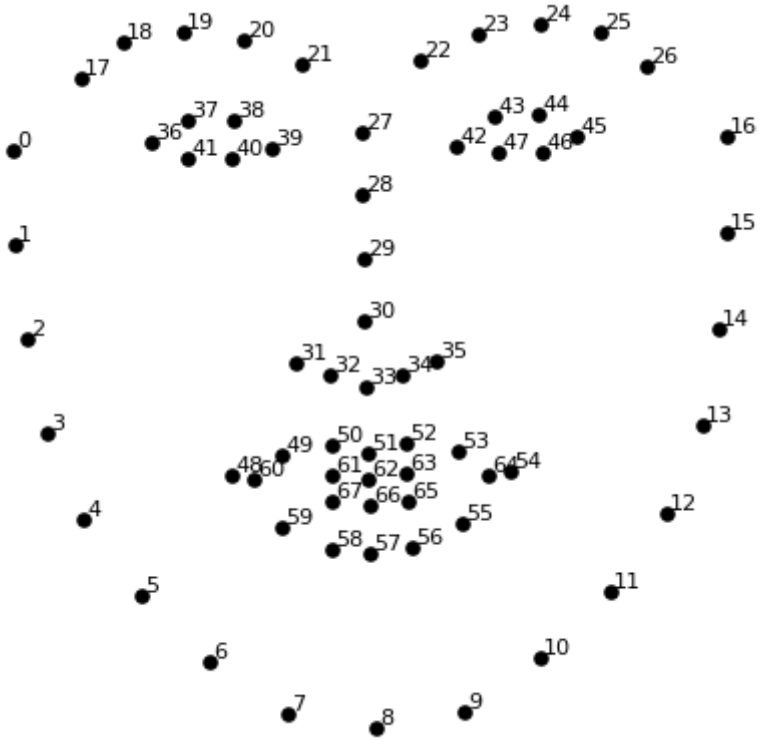
If the video option is chosen, we get three categories: Dance, Comedy and Movie. These options contain videos of respective categories. These are selected using a blinking ratio. After selecting the category, some videos of the selected category are displayed. These are selected and played using a blinking ratio.

THEORY

This application is built in two main steps.

- Detection of eye
- Virtual Menu

So, let's first focus on eye detection. For this, we have first detected the face and then the location of the eye. Detection of eyes is important because their movement and blinking are the main part of this project. To detect the face, some of the important libraries used are opencv and dlib. This will be later explained in detail. We are using the real time frames from the webcam and detecting the face. After detection of the face, we have to detect the eyes. Using the face landmarks detection approach we can find 68 specific landmarks of the face. To each point there is a specific index assigned.



We need to detect two eyes separately:

- Left eye points: (36, 37, 38, 39, 40, 41)
- Right eye points: (42, 43, 44, 45, 46, 47)

So we got the points of the eye. Now we have calculated the vertical midpoints of the eye. These midpoints can be used to detect the blinking of the eye. Blinking is used to select items from the menu. There are three ways where we can say that the eye is blinked. Eyelid is closed, we can't see the eyeball anymore and the other is when the both eyelashes connect together. We have to also count the amount of time the eye is closed. The time should be short enough to say it is blinking. If it is of more time, we can say that the eye

is closed, not blinking. So a notable time 0.3 or 0.4 seconds can be said for the blinking of eyes. So, for the detection of the blinking we have used opencv. We can note that if the eye is closed, the distance between the horizontal end points of the eye is almost the same. But the distance between the vertical points is very less compared to when it is open. So to detect the blinking, we use the horizontal distance as reference and calculate the ratio of the horizontal distance and vertical distance. So, we use this ratio and we can tell if the eye is blinking or not. This method we do for both the eyes and then we take the average of these ratios to detect the blinking of the eye. Now we have done with the blinking. Now, we have to detect the gaze of the eye.

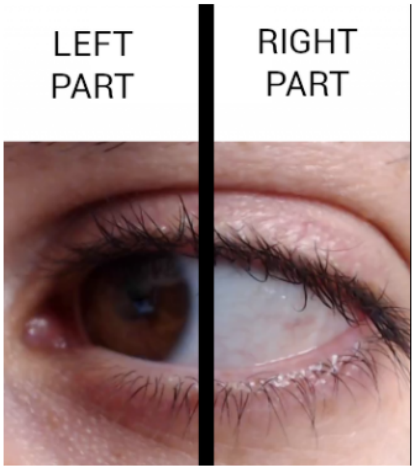
We have to know the importance of detection of the gaze in this project. The menu which we are creating has two parts. The parts are on the left and right side of the menu. To select a particular part the user should see on the particular side. So to detect which side the user is watching, we find the gaze ratio. For this, we first find the coordinates of the eye (landmark points). Once we have the coordinates of the eye, we can create the mask to extract exactly the inside of the eye and exclude all the surroundings.

So using this, we can extract the eye from the face and can be placed in another window. We have to remember one point that we can just cut or crop images as a rectangular shape. So, to crop the eye we take all the extreme points of the eye to get the rectangle. We also get the threshold to detect the gaze. Now the important thing is to detect or judge which side our eye is looking at. To find this, first we have to know how the eye appears when it is looking in different directions. Generally, the eye is composed of three parts.

- Pupil – the black circle in the middle
- Iris – the bigger circle that can have different color for different people
- Sclera – it's always white

Now let us know how the eye behaves when we look in a desired direction.

From the above image, we can easily infer that the sclera which is the white part of the eye covers the left part of the eye when we look at the right side and vice versa. When it is a centre , it is well balanced. So, using this we can detect the gaze of the eye. We divide the eye into two parts.



Here, we can say that the sclera(white part) is more visible on the right side since the eye is looking towards the left. So, we can convert the eye into a threshold and then count the white pixels on both the sides and then we can calculate the gaze ratio using the count of white pixels on the left and right. This we do for both eyes and then take the average of it as gaze ratio.

We have done with the detection of the eye. Now, we have to create the menu. The menu which has been created is used for ALS patients. The menu consists of two parts. The first part is the food and the other is other applications. In the other applications, there are three options: keyboard, songs and videos. Keyboard application is used to type anything, songs application is used to play songs of desired language and video application is used to play videos of desired category. In the food, there are two options: breakfast and lunch/dinner. In each of these categories the user can select the food he wants. After his selection, a confirmation menu is opened of yes/no to confirm his order.

IMPLEMENTATION

Blinking based option selection boards:

- **def draw_letters(letter_index, items, letter_light)**

This is a function to draw the keyboard of letters to type text.This keyboard consists of blocks where in each block we have letter options to type. Usually the blocks are black and with white text but whenever the block turns white and text turns black then we have the opportunity to choose the option in that particular block.

For this function we pass three arguments. One is letter_index which assigns the coordinates of the top left corner of the block in the keyboard. Next is items which points to a value in the dictionary where the value is the text we write in the respective block as option. Next is letter_light if it's given true, then the block turns white and text turns black and vice versa if it's given false. Next we used the getTextSize() function from the cv2 module to get the width and height of the item's text size. This function requires text, font type, font scale and font thickness as the arguments.

We used the rectangle() function from the cv2 module to get the block in the keyboard and we used the putText() function from the cv2 module to put the text in the block of the keyboard. The rectangle() function requires keyboard, coordinates of diagonals i.e., coordinates of top left and bottom right corner of the block, color of the block using bgr color code and thickness which is generally '-1' to fill the rectangle border by black color as the arguments. The putText() function requires keyboard, text, starting coordinates of the text, font style, font scale, color using rgb color and font thickness as parameters.

- **def draw_menu(item_index, items, letter_light)**

This is a function to draw a board of applications, languages of songs, categories of videos and videos.This keyboard consists of blocks where in each block we have options to choose. For this function we pass three arguments. One is item_index which assigns the coordinates of the top left corner of the block in the keyboard. Next is items which points to a value in the dictionary where the value is the text we put in the respective block as option. Next is letter_light if it's given true, then the block turns white and text turns black and vice versa if it's given false. Next we used the getTextSize() function from the cv2 module to get the width and height of the item's text size. We used the rectangle() function from the cv2 module to get the block in the keyboard and we used the putText() function from the cv2 module to put the text in the block of the keyboard.

- **def draw_items(item_index, items, letter_light)**

This is a function to draw a board of menu items in the breakfast section and lunch or dinner section. This keyboard consists of blocks where in each block we have food options to choose. For this function we pass three arguments. One is item_index which assigns the coordinates of the top left corner of the block in the keyboard. Next is items which points to a value in the dictionary where the value is the text we put in the respective block as option. Next is letter_light if it's given true, then the block turns white and text turns black and vice versa if it's given false. Next we used the getTextSize() function from the cv2 module to get the width and height of the items text size. We used the rectangle() function from the cv2

module to get the block in the keyboard and we used the putText() function from the cv2 module to put the text in the block of the keyboard.

- **def draw_songs(song_index, text, letter_light)**

This is a function to draw a board of songs playlist. This keyboard consists of blocks where in each block we have song options to choose.. For this function we pass three arguments. One is song_index which assigns the coordinates of the top left corner of the block in the keyboard. Next is text which points to a value in the dictionary where the value is the text we put in the respective block as option. Next is letter_light if it's given true, then the block turns white and text turns black and vice versa if it's given false. Next we used the getTextSize() function from the cv2 module to get the width and height of the items text size. We used the rectangle() function from the cv2 module to get the block in the keyboard and we used the putText() function from the cv2 module to put the text in the block of the keyboard.

Gaze based option selected keyboards:

- **def draw_keyboard()**

This function is used to draw board of left and right options in the virtual keyboard section. This board is divided into two parts left and right and the parts are separated by a line in the middle of the board. We draw this line with the help of line() function from the cv2 module. The line() function requires keyboard, starting and ending coordinates of the line, color using rgb code and thickness of the line as arguments. We put text in each of the two parts using the putText() function from the cv2 module.

- **def draw_keyboard_1()**

This function is used to draw board of breakfast and lunch or dinner options in the food section. This board is divided into two parts left and right and the parts are separated by a line in the middle of the board. We draw this line with the help of line() function from the cv2 module. We put text in each of the two parts using the putText() function from the cv2 module.

- **def draw_keyboard_2()**

This function is used to draw board of food and other applications options. This board is divided into two parts left and right and the parts are separated by a line in the middle of the board. We draw this line with the help of line() function from the cv2 module. We put text in each of the two parts using the putText() function from the cv2 module.

- **def draw_keyboard_3()**

This function is used to draw a board of yes and no options in the food section. This board is divided into two parts left and right and the parts are separated by a line in the middle of the board. We draw this line with the help of line() function from the cv2 module. We put text in each of the two parts using the putText() function from the cv2 module.

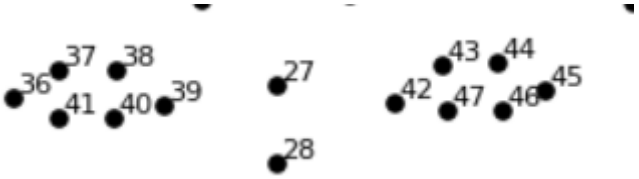
Functions for eye detection :

- **midpoint(p1 ,p2)**

This function is used to find the midpoint of two points. We pass two points as arguments for this function and it returns the midpoint coordinates of these two points. The midpoint which is returned is converted into int.

- **get_blinking_ratio(eye_points, facial_landmarks)**

This function is used to find the blinking ratio of the particular eye. For this function, we pass the arguments as the eye_points and the facial_landmarks. In this function, first we calculate the leftmost point, rightmost point, center top and center bottom. These points are calculated using the eye points we pass. If we take left eye as an example,



So, for the left eye the extreme left is indexed with 36. So now the leftmost point has the coordinates of the point which is indexed as 36. Similarly with the extreme right. To get the coordinates of center top, we calculate the midpoint of the points which are indexed as 37 and 38. To get the coordinates of center bottom, we calculate the midpoint of the points which are indexed as 41 and 40. So, now we got the coordinates of all the points which we require. Now, we have to calculate the horizontal line length and vertical line length. So, for this we use the hypot() function from the math library.

The math.hypot() method returns the Euclidean norm.The Euclidean norm is the distance from the origin to the coordinates given. We pass the difference of the left point and right point as parameters to get the horizontal distance(x-coordinate: difference of left_point x-coordinate and right_point x-coordinate and y-coordinate: difference of left_point y-coordinate and right_point y-coordinate). To get the vertical distance , we pass the difference of the center top point and center bottom point as parameters(x-coordinate: difference of center_top x-coordinate and center_bottom x-coordinate and y-coordinate: difference of center_top y-coordinate and center_bottom y-coordinate). Now, we got the horizontal and vertical distances of the eye. So, we have to now calculate the blinking ratio.The ratio of horizontal distance and vertical distance gives us the ratio. After the calculation of the ratio, it is returned.

- **eyes_contour_points(facial_landmarks)**

This function is used to get all the contour points of the both eyes.We pass the facial_landmarks as the parameter for this function. This function returns two arrays, one array contains the contour points of the left eye and other contains the contour points of the right eye.

First we initialize two lists named as right and left. First we append all the coordinates of the left eye into the list which is named as left. To append the coordinates of the left eye, we have to take the index range from 36 to 42(Since left eye indices lie between 36 and 42). Then we append coordinates of each and every point in the list as a list. Similarly for the right eye, but the index range is between 42 to 48.The coordinates of the right eye are appended into the list named as right as a list. Then we convert these left and right lists into arrays using numpy. Then we return both of these arrays

- **get_gaze_ratio(eye_points, facial_landmarks)**

This function is used to find the gaze ratio of the particular eye. For this function, we pass the arguments as the eye_points and the facial_landmarks.

First we create an array with all coordinates of the eye. If we take the left eye for example, we have 6 indices. So we have to create an array using the coordinates of all these indices. The first column of this array is the x coordinates of all the points and second column is the y coordinates of all the points. We now extract the eye from the face and we put it on a new window. Only we need to keep in mind that we can only cut out rectangular shapes from the image, so we take all the extreme points of the eye (top left and right bottom) to get the rectangle.

So to get the extreme points, we use the min() and max() in the numpy library. To get the leftmost point we use the min() function and we calculate the minimum of all the x-coordinates in the array which we have created initially. To get the rightmost point we use the max() function and calculate the maximum of all x coordinates from the same array. Then to get the topmost point we use the min() function and calculate the minimum of all y coordinates from the same array. To get the bottommost point we use the max() function and calculate the maximum of all y coordinates from the same array.

Then we calculate the height and width of the frame or the window from which the eye is extracted. Then we create a mask. We can create the mask to extract exactly the inside of the eye and exclude all the surroundings. Initially we create the mask array with all zeros of size height x width. Then we create a window with only the eye. Then we convert the gray_eye into a threshold. This we do using the inbuilt threshold() function in cv2 library.

Threshold images help us to calculate the actual values of what we want. After finding out the threshold, we calculate the number of white pixels in the image what we got. Since we have to detect the sclera. Technically to detect the sclera we convert the eye into grayscale, we find a threshold and we count the white pixels. We divide the white pixels of the left part and those of the right part and we get the gaze ratio. We calculate these white pixels using the inbuilt function cv2.countNonZero. We find the white pixels on both the sides and then find the gaze ratio. Gaze ratio is defined as white pixels on to the left by white pixels on to the right. So after the calculation of gaze ratio, the value is returned.

TOOLS/ SOFTWARE USED

- JupyterLab
- Anaconda

MODULES AND SHORT DESCRIPTION

The main package we have used is **opencv**. OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax.

All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

Another important package used is **dlib**. Dlib is a toolkit for making real world machine learning and data analysis applications in C++. While the library is originally written in C++, it has good, easy to use Python bindings. Other libraries used are:

- **numpy** - 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.
- **math** - The math module is a standard module in Python to use mathematical functions.
- **pyglet** - pyglet is a cross-platform windowing and multimedia library for Python, intended for developing games and other visually rich applications.
- **mutagen** - Mutagen is a Python module to handle audio metadata.
- **win32com** - It is an API which is used to convert text to speech.
- **ffpyplayer** - FFPyPlayer is a python binding for the FFmpeg library for playing and writing media files.
- **glob** - The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order.
- **time** - This module provides various time-related functions.

SAMPLE CODE:

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

```
# FUNCTION TO DRAW A BOARD OF APPLICATIONS, LANGUAGES OF SONGS, CATEGORIES OF VIDEOS AND VIDEOS
def draw_menu(item_index, items, letter_light):
    # Keys
    if item_index == 0:
        x = 0
        y = 0
    elif item_index == 1:
        x = 450
        y = 0
    elif item_index == 2:
        x = 0
        y = 435
    elif item_index == 3:
        x = 450
        y = 435

    width = 450
    height = 435
    th = 3 # thickness

    # items settings
    font_letter = cv2.FONT_HERSHEY_PLAIN
    font_scale = 3.5
    font_th = 4
    items_size = cv2.getTextSize(items, font_letter, font_scale, font_th)[0]
    width_items, height_items = items_size[0], items_size[1]
    items_x = int((width - width_items) / 2) + x
    items_y = int((height + height_items) / 2) + y

    if letter_light is True:
        cv2.rectangle(keyboard, (x + th, y + th), (x + width - th, y + height - th), (255, 255, 255), -1)
        cv2.putText(keyboard, items, (items_x, items_y), font_letter, font_scale, (51, 51, 51), font_th)
    else:
        cv2.rectangle(keyboard, (x + th, y + th), (x + width - th, y + height - th), (51, 51, 51), -1)
        cv2.putText(keyboard, items, (items_x, items_y), font_letter, font_scale, (255, 255, 255), font_th)
```

```
# FUNCTION TO DRAW BOARD OF LEFT AND RIGHT OPTIONS
def draw_keyboard():
    rows, cols, _ = keyboard.shape
    th_lines = 4 # thickness lines
    cv2.line(keyboard, (int(cols/2) - int(th_lines/2), 0),(int(cols/2) - int(th_lines/2), rows),
             (51, 51, 51), th_lines)
    cv2.putText(keyboard, "LEFT", (80, 300), font, 6, (255, 255, 255), 5)
    cv2.putText(keyboard, "RIGHT", (80 + int(cols/2), 300), font, 6, (255, 255, 255), 5)
```

```
# FUNCTION TO FIND BLINKING RATIO OF EYES
def get_blinking_ratio(eye_points, facial_landmarks):
    left_point = (facial_landmarks.part(eye_points[0]).x, facial_landmarks.part(eye_points[0]).y)
    right_point = (facial_landmarks.part(eye_points[3]).x, facial_landmarks.part(eye_points[3]).y)
    center_top = midpoint(facial_landmarks.part(eye_points[1]), facial_landmarks.part(eye_points[2]))
    center_bottom = midpoint(facial_landmarks.part(eye_points[5]), facial_landmarks.part(eye_points[4]))

    hor_line_lenght = hypot((left_point[0] - right_point[0]), (left_point[1] - right_point[1]))
    ver_line_lenght = hypot((center_top[0] - center_bottom[0]), (center_top[1] - center_bottom[1]))

    ratio = hor_line_lenght / ver_line_lenght
    return ratio
```

```python
# FUNCTION TO GET CONTOUR POINTS OF EYE
def eyes_contour_points(facial_landmarks):
    left_eye = []
    right_eye = []
    for n in range(36, 42):
        x = facial_landmarks.part(n).x
        y = facial_landmarks.part(n).y
        left_eye.append([x, y])
    for n in range(42, 48):
        x = facial_landmarks.part(n).x
        y = facial_landmarks.part(n).y
        right_eye.append([x, y])
    left_eye = np.array(left_eye, np.int32)
    right_eye = np.array(right_eye, np.int32)
    return left_eye, right_eye
```

```python
# FUNCTION TO FIND GAZE RATIO OF EYES
def get_gaze_ratio(eye_points, facial_landmarks):
    eye_region = np.array([(facial_landmarks.part(eye_points[0]).x, facial_landmarks.part(eye_points[0]).y),
                           (facial_landmarks.part(eye_points[1]).x, facial_landmarks.part(eye_points[1]).y),
                           (facial_landmarks.part(eye_points[2]).x, facial_landmarks.part(eye_points[2]).y),
                           (facial_landmarks.part(eye_points[3]).x, facial_landmarks.part(eye_points[3]).y),
                           (facial_landmarks.part(eye_points[4]).x, facial_landmarks.part(eye_points[4]).y),
                           (facial_landmarks.part(eye_points[5]).x, facial_landmarks.part(eye_points[5]).y)], np.int32)


    min_x = np.min(eye_region[:, 0])
    max_x = np.max(eye_region[:, 0])
    min_y = np.min(eye_region[:, 1])
    max_y = np.max(eye_region[:, 1])

    height, width, _ = frame.shape
    mask = np.zeros((height, width), np.uint8)
    cv2.polylines(mask, [eye_region], True, 255, 2)
    cv2.fillPoly(mask, [eye_region], 255)
    eye = cv2.bitwise_and(gray, gray, mask=mask)

    gray_eye = eye[min_y: max_y, min_x: max_x]
    _, threshold_eye = cv2.threshold(gray_eye, 70, 255, cv2.THRESH_BINARY)

    threshold_eye = cv2.resize(threshold_eye, None, fx=5, fy=5)
    eye = cv2.resize(gray_eye, None, fx=5, fy=5)
    cv2.imshow("Eye", eye)
    cv2.imshow("Threshold", threshold_eye)

    height, width=threshold_eye.shape
    left_side_threshold=threshold_eye[0:height,0:int(width/2)]
    left_side_white=cv2.countNonZero(left_side_threshold)

    right_side_threshold=threshold_eye[0:height,int(width/2):width]
    right_side_white=cv2.countNonZero(right_side_threshold)


    if left_side_white == 0:
        gaze_ratio=1
    elif right_side_white == 0:
        gaze_ratio=5
    else:
        gaze_ratio=left_side_white/right_side_white


    return gaze_ratio
```

```
# FOOD, OTHER APPLICATIONS
if selected_frame==0:
    time.sleep(0.01)
    # DETECTING GAZE TO SELECT LEFT OR RIGHT KEYBOARD
    gaze_ratio_left_eye = get_gaze_ratio([36, 37, 38, 39, 40, 41], landmarks)
    gaze_ratio_right_eye = get_gaze_ratio([42, 43, 44, 45, 46, 47], landmarks)
    gaze_ratio = (gaze_ratio_right_eye + gaze_ratio_left_eye) / 2

    # IF GAZE_RATIO IS LESS THAN 0.9 THEN THE PERSON IS LOOKING RIGHT
    # OPTION ON THE RIGHT IS SELCTED AND APPROPRIATE ACTION IS PERFORMED
    if gaze_ratio < 0.9:
        keyboard_selected = "OTHER APPLICATIONS"
        keyboard_selection_frames += 1
        # IF GAZE REMAINS SAME FOR MORE THAN 15 FRAMES THEN MOVE TO THE KEYBOARD
        if keyboard_selection_frames == 15:
            selected_frame=4
            t="other applications"
            speaker.Speak(t)
            # SET FRAMES COUNT TO 0 AFTER KEYBOARD IS SELECTED
            frames = 0
            keyboard_selection_frames = 0
        if keyboard_selected != last_keyboard_selected:
            last_keyboard_selected = keyboard_selected
            keyboard_selection_frames = 0

    # IF GAZE_RATIO IS GREATER THAN 0.9 THEN THE PERSON IS LOOKING LEFT
    # OPTION ON THE LEFT IS SELCTED AND APPROPRIATE ACTION IS PERFORMED
    else:
        keyboard_selected = "FOOD"
        keyboard_selection_frames += 1
        # IF GAZE REMAINS SAME FOR MORE THAN 15 FRAMES THEN MOVE TO THE KEYBOARD
        if keyboard_selection_frames == 15:
            selected_frame=1
            t="food"
            speaker.Speak(t)
            # SET FRAMES COUNT TO 0 AFTER KEYBOARD IS SELECTED
            frames = 0
        if keyboard_selected != last_keyboard_selected:
            last_keyboard_selected = keyboard_selected
            keyboard_selection_frames = 0
```

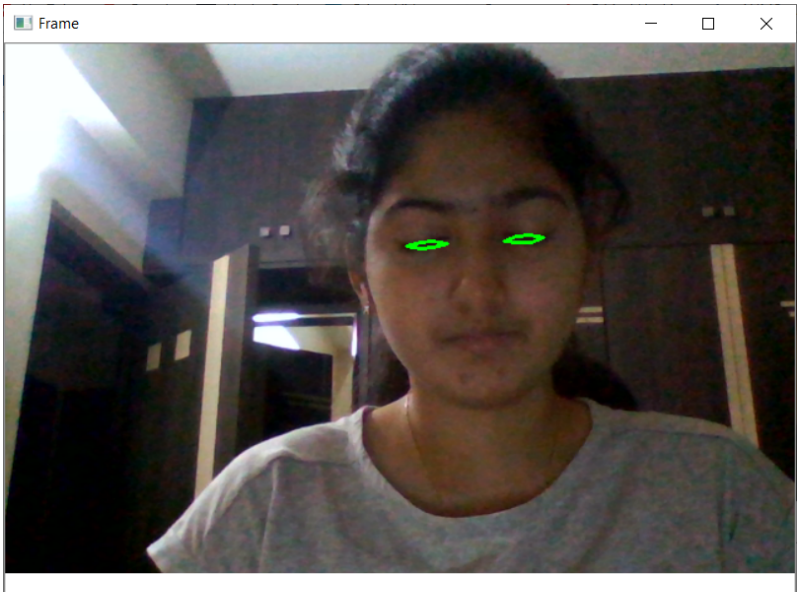
```
# BREAKFAST MENU
elif selected_frame==2:
    if blinking_ratio > 5:

        blinking_frames += 1
        frames -= 1

        # SHOW GREEN EYES WHEN CLOSED
        cv2.polylines(frame, [left_eye], True, (0, 255, 0), 2)
        cv2.polylines(frame, [right_eye], True, (0, 255, 0), 2)

        if blinking_frames == frames_to_blink:
            if active_letter != "<" and active_letter != "ENTER":
                items.append(active_letter)
                sound.play()
                selected_frame = 2
            elif active_letter == "ENTER":
                if len(items)==0:
                    t="Kindly place your order    "
                    speaker.Speak(t)
                    selected_frame=2
                else:
                    order=2
                    t="Your order is     "
                    speaker.Speak(t)
                    for i in items:
                        speaker.Speak(i)
                    selected_frame=8
            else:
                items.clear()
                board[:] = 255
                cv2.imshow("Board", board)
                selected_frame=0


    else:
        blinking_frames = 0
```

```
# DRAW THE BREAKFAST MENU
if selected_frame == 2:
    if frames == frames_active_letter:
        item_index += 1
        frames = 0
    if item_index == 9:
        item_index = 0
    for i in range(9):
        if i == item_index:
            light = True
        else:
            light = False
        draw_items(i, items_set[i], light)
```

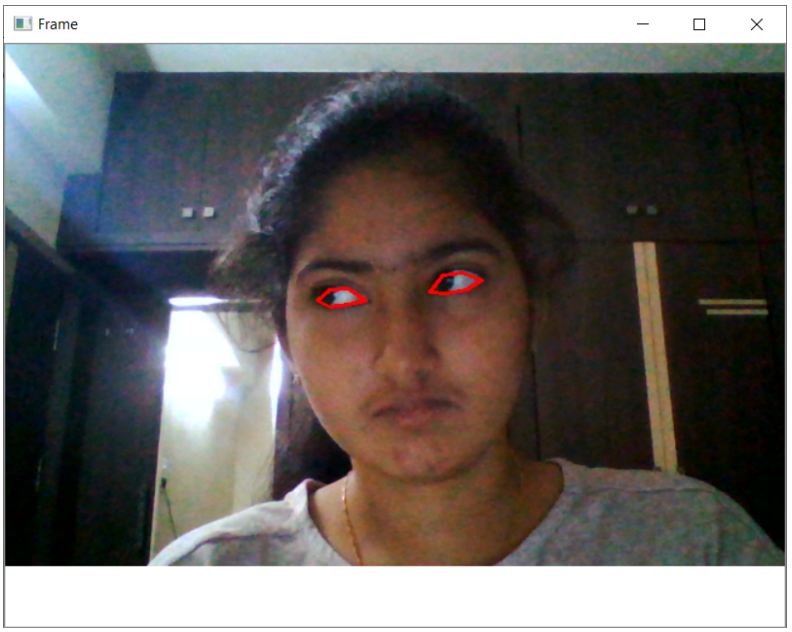
SAMPLE OUTPUT

Frame when we are not blinking:

Frame when we are blinking:



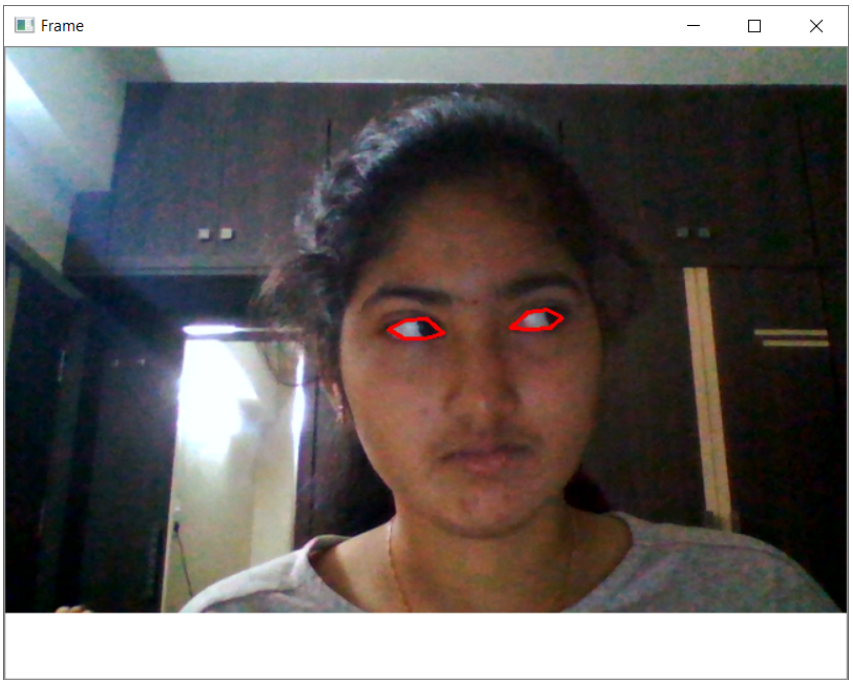
Frame when looking right side:



Eye when looking right side:

Threshold eye when looking right side:

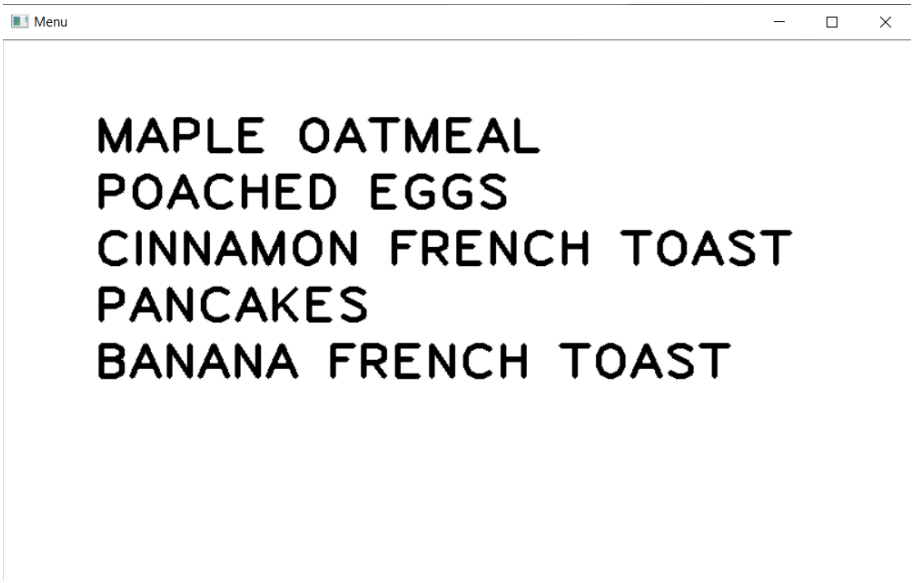
Frame when looking left side:



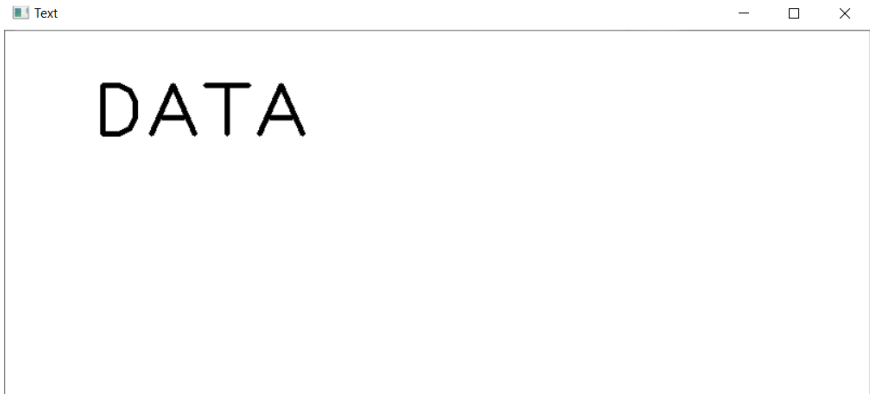
Eye when looking left side:

Threshold eye when looking left side:

Board after menu is selected:

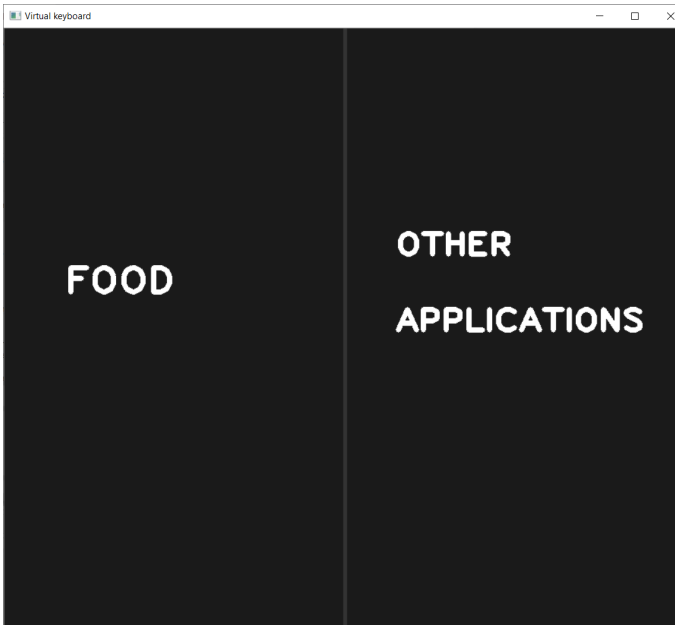


Board after text is typed:

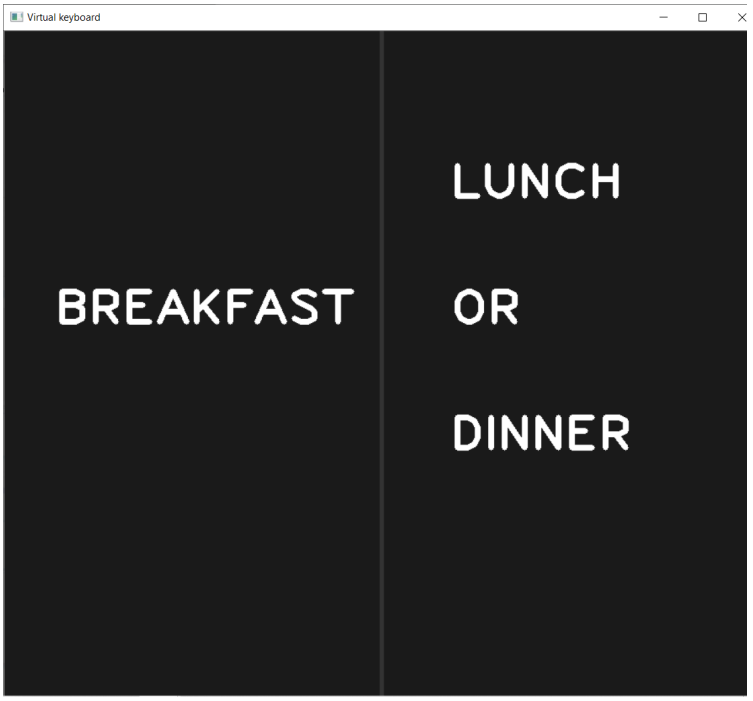


DIFFERENT FRAMES:

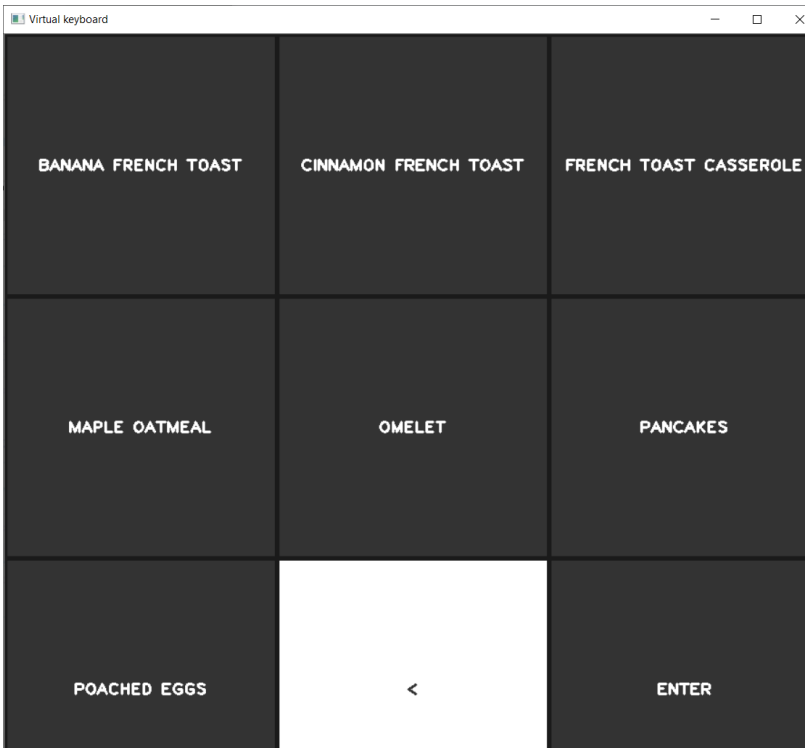
- **FOOD, OTHER APPLICATIONS**



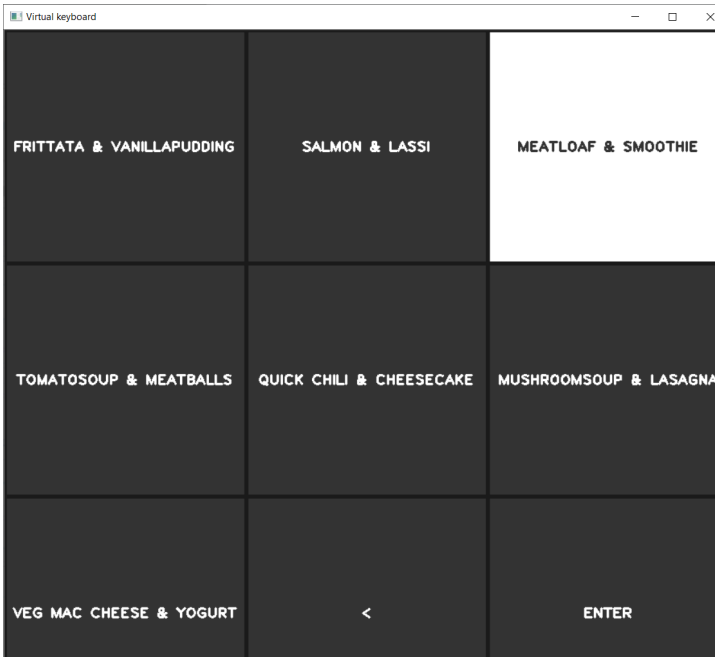
- **BREAKFAST, LUNCH OR DINNER**



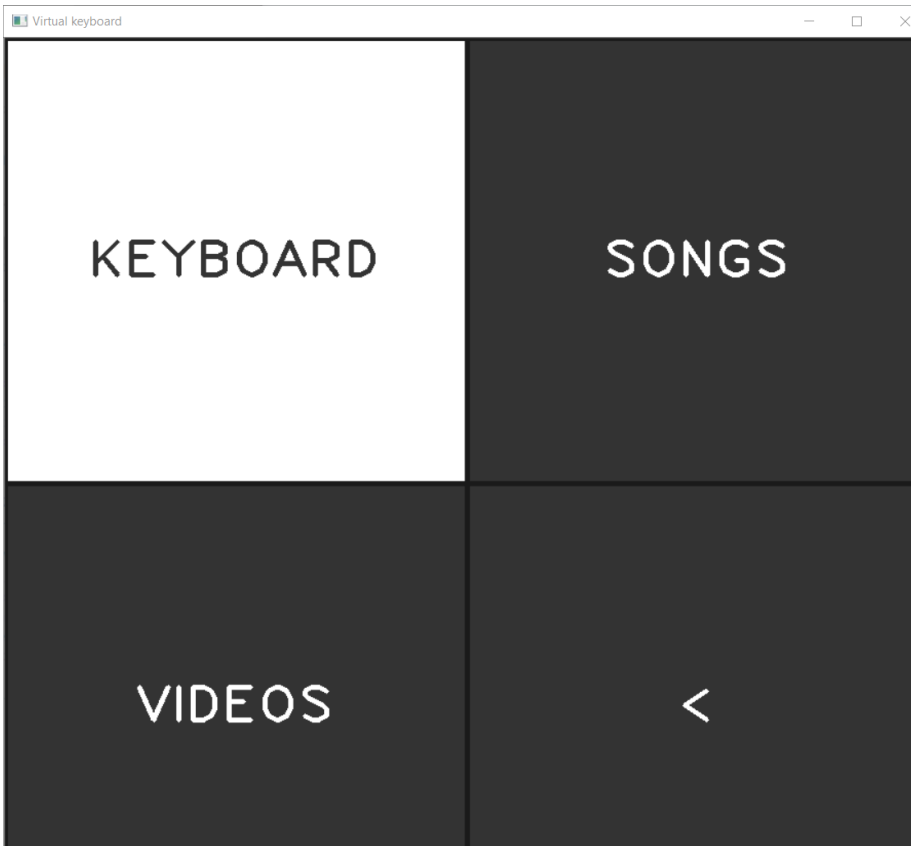
- **BREAKFAST MENU**



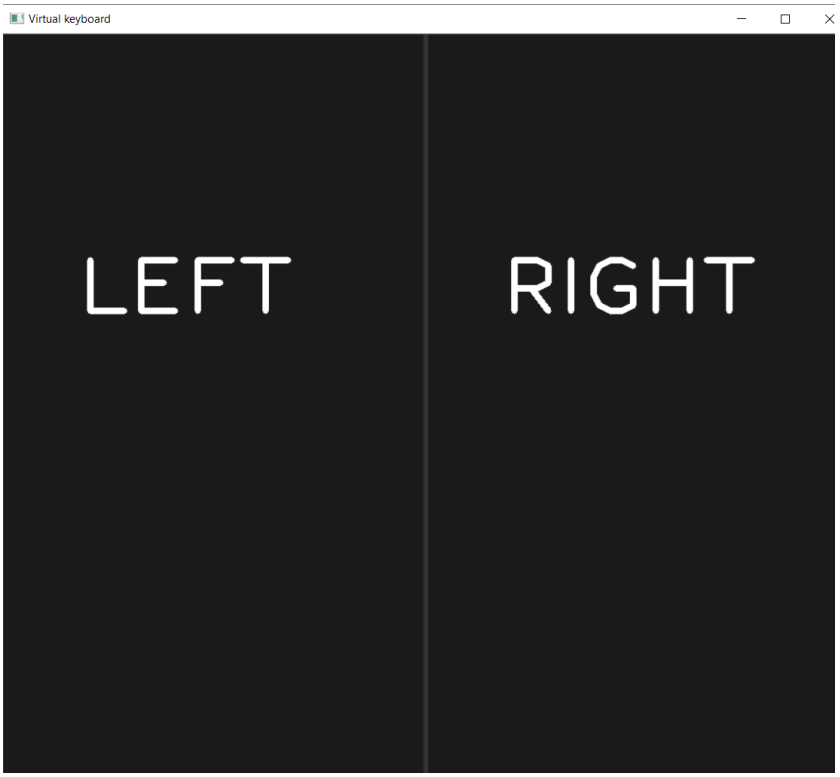
- **LUNCH OR DINNER MENU**



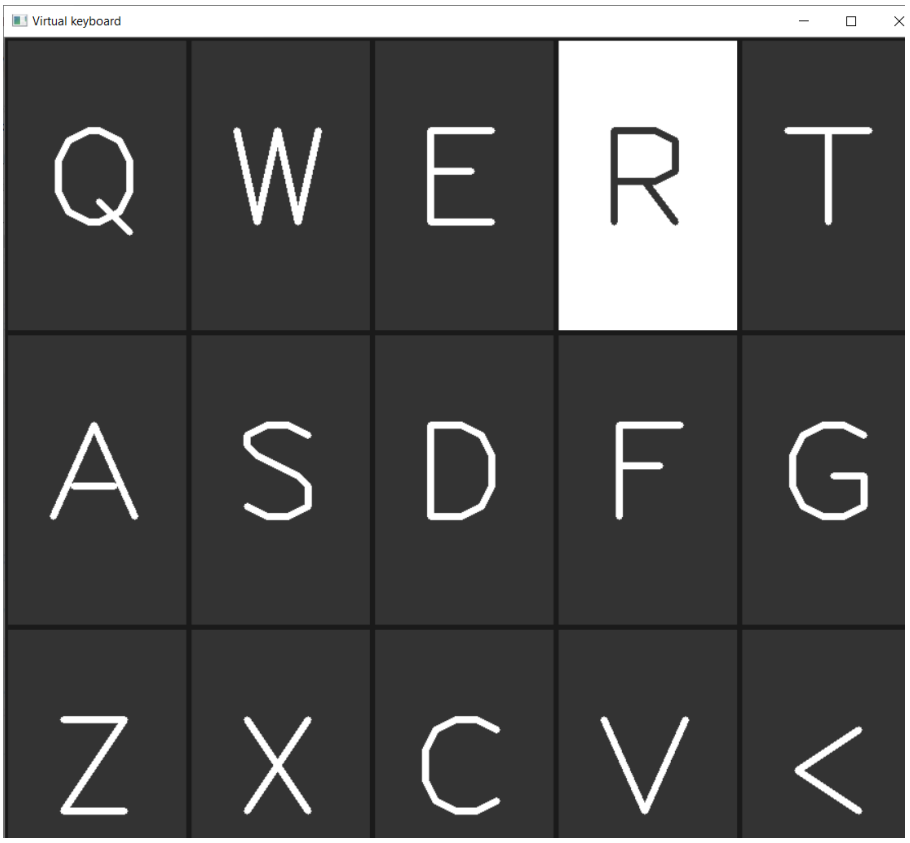
- **OTHER APPLICATIONS**



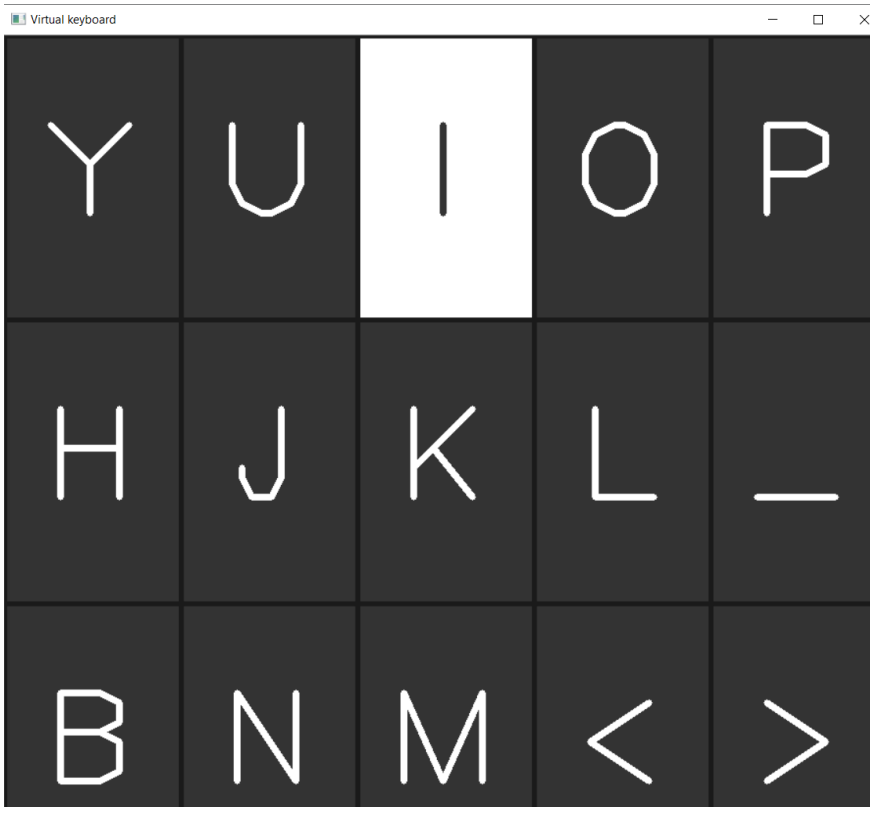
- **LEFT, RIGHT**



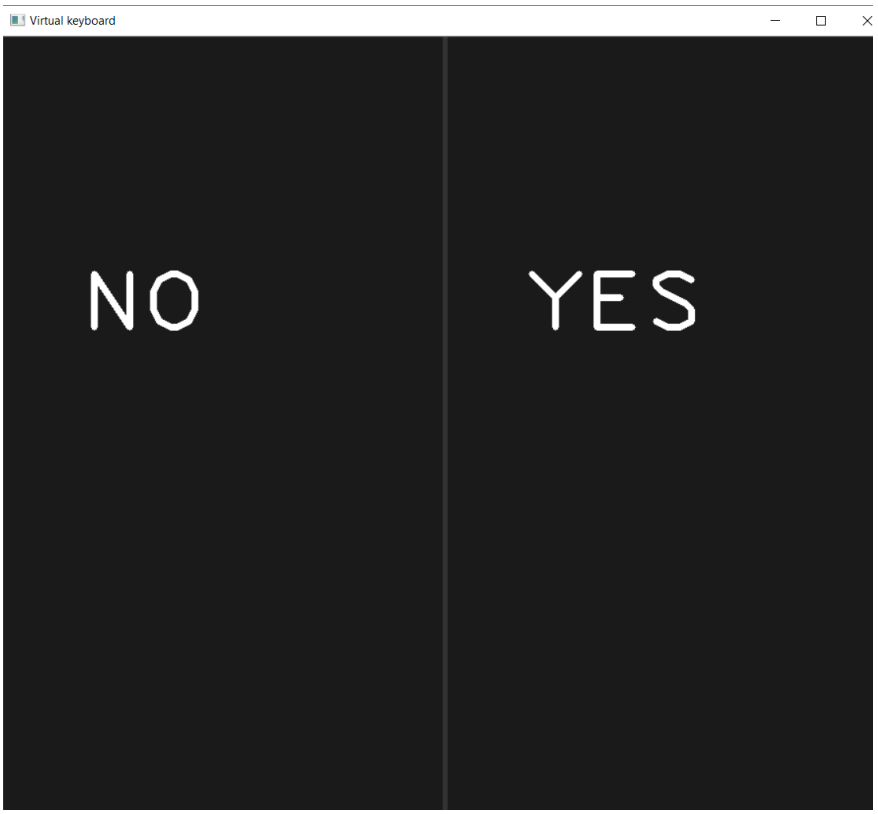
- **LEFT SIDE OF THE KEYBOARD**



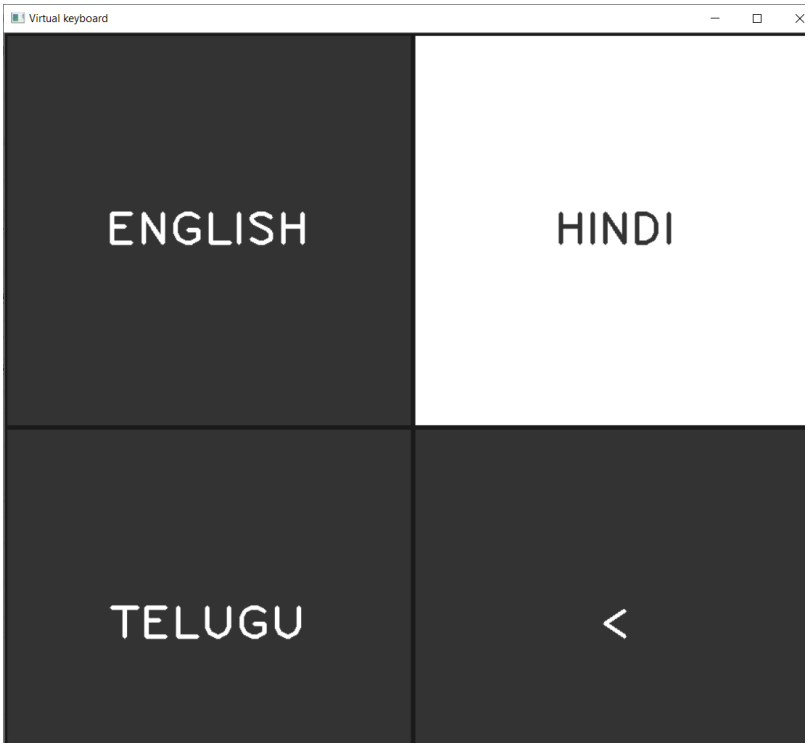
- **RIGHT SIDE OF THE KEYBOARD**



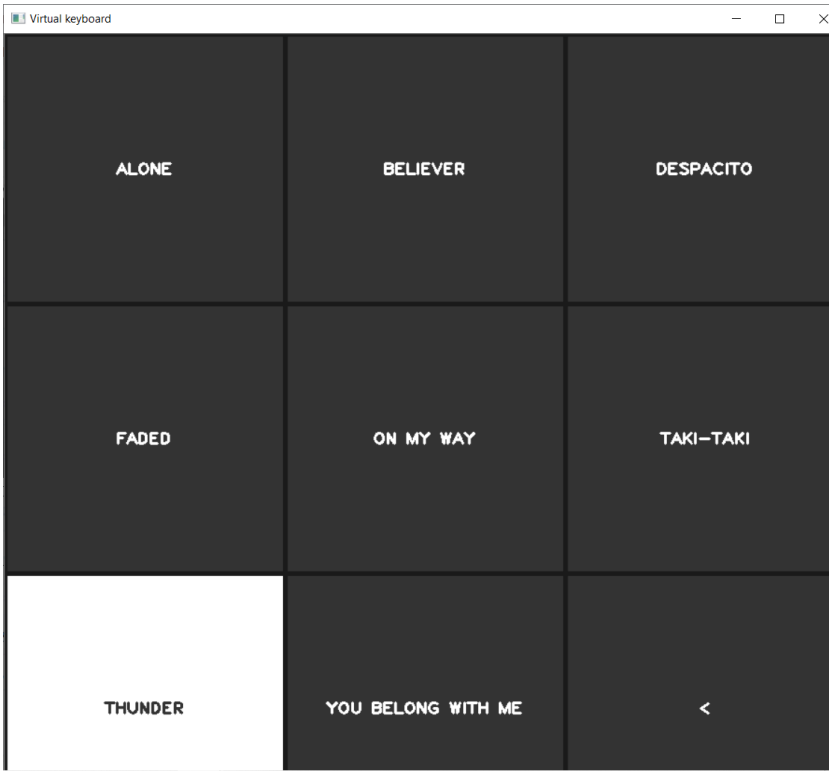
- **NO, YES**



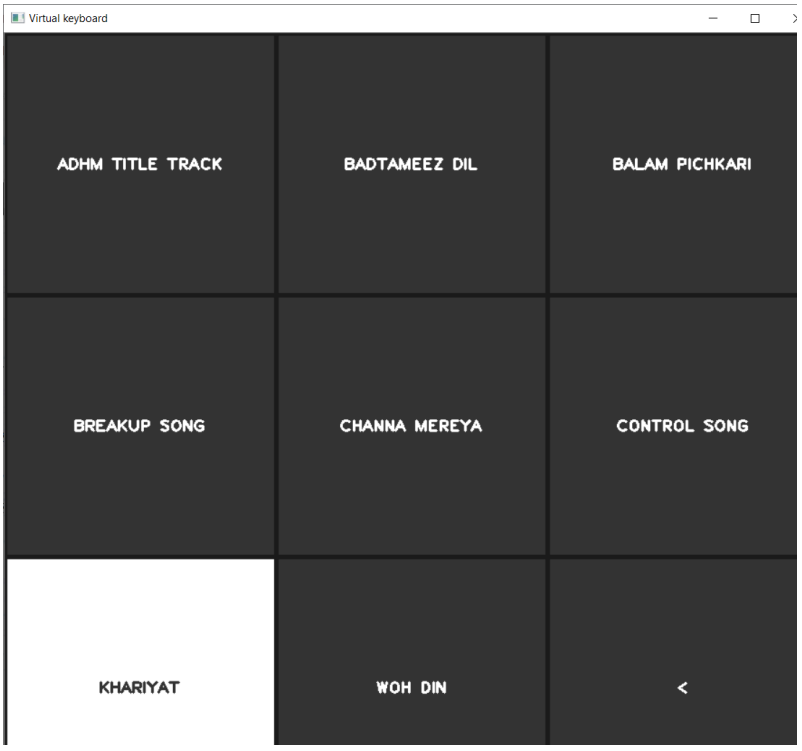
- **LANG_SELECTION**



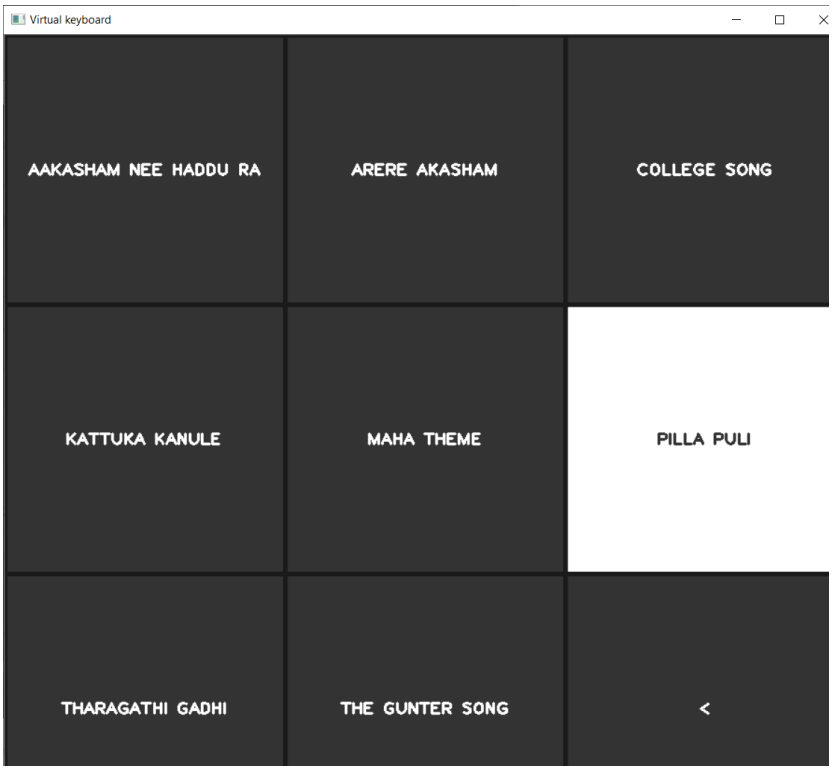
- **ENGLISH SONGS**



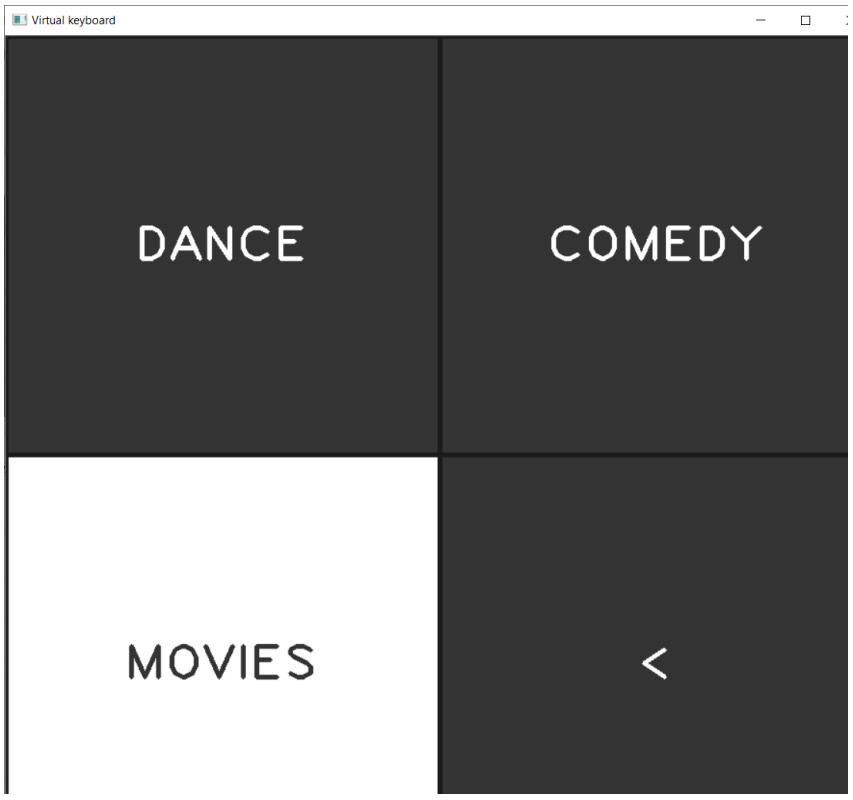
- **HINDI SONGS**



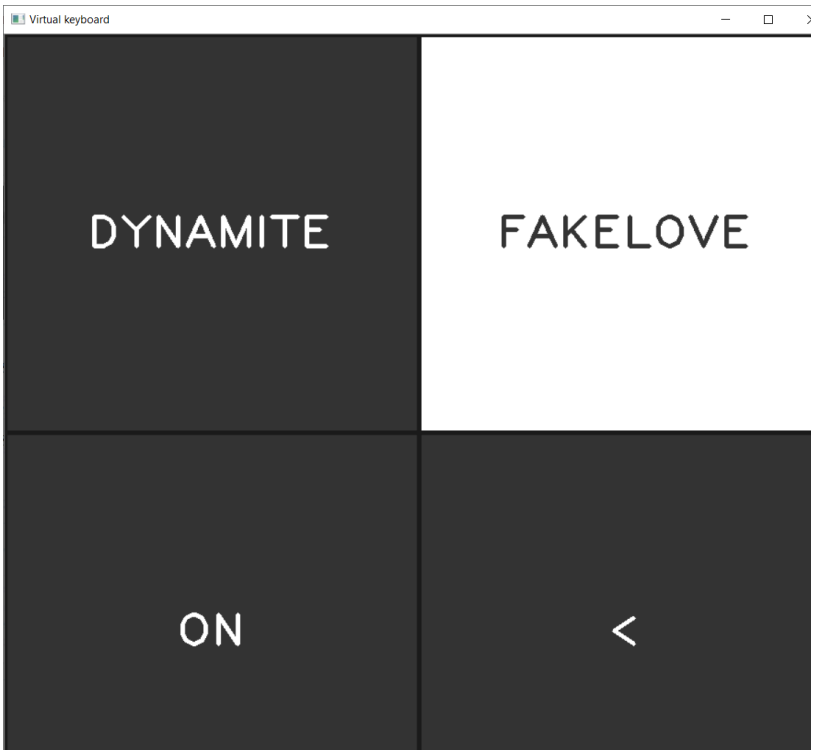
- **TELUGU SONGS**



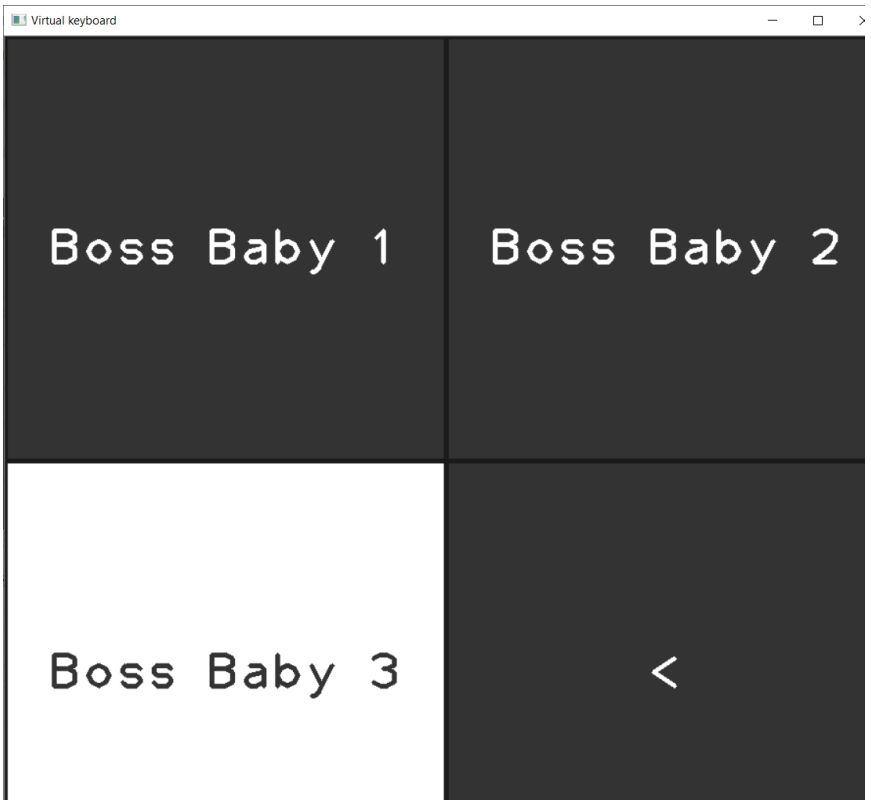
- **CATEGORIES**



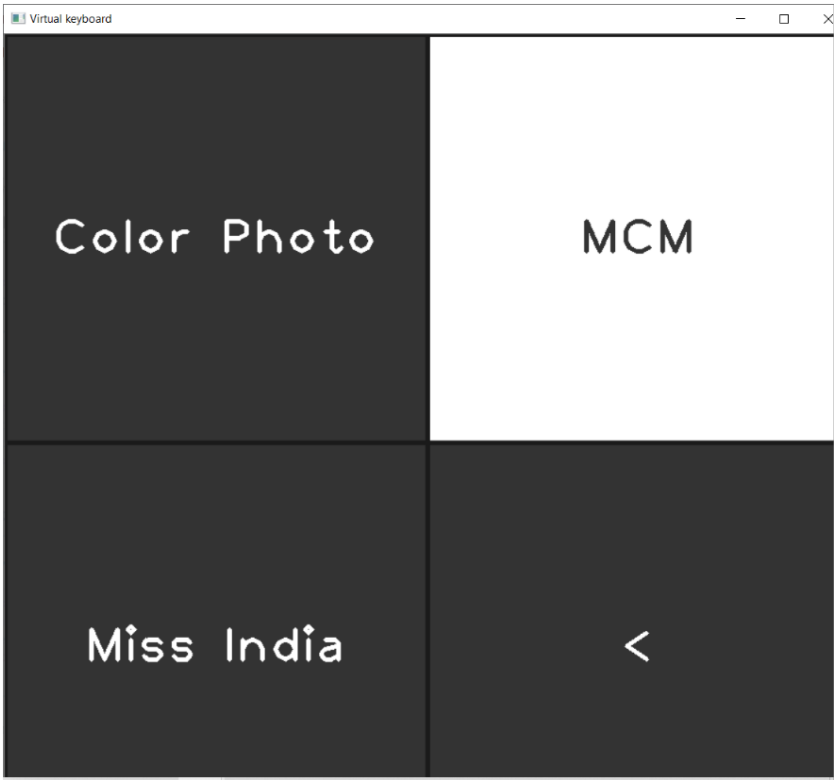
- **DANCE VIDEOS**



- **COMEDY VIDEOS**



- **MOVIE VIDEOS**



CONCLUSION

Eye tracking is an important technique that offers an objective way to see where in a scene a person's visual attention is located. Today, the human eye-gaze can be recorded by relatively unremarkable techniques. This project argues that it is possible to use eye-gaze of a computer user in the interface to aid the control of the application. Care must be taken, though, that eye-gaze tracking data is used in a sensible way, since the nature of human eye-movements is a combination of several voluntary and involuntary cognitive processes. There are some patients whose motor nerves do not work properly. So, this project helps in improving quality of living for ALS patients by helping them to tell their needs or entertain themselves using eye gaze.

REFERENCES

[1] The ALS Association, Oct 2008, "What is ALS, " http://www.alsa.org/als/what.cfm.

[2] Axistive, Jun 23, 2007, "ERICA Eye Tracking, " http://www.axistive.com/erica -eye-tracking.html.

[3] Eye Response Technologies (now DynaVox Mayer-Johnson), 21 July 2009, "Eye Tracker Erica," http://www.cogain.org/wiki/Eye_Tracker_Erica.

[4] Cihan Topal, Ömer Nezih Gerek and Atakan Doğan, "A head - mounted sensor-based eye tracking device: eye touch system," 2008.

[5] Dale Grover, "Progress on an eye tracking system using multiple near-infrared emitter/detector pairs with special application to efficient eye - gaze communication," Proceedings of COGAIN 2006: Gazing into the Future, pp. 21-25, 2006.

[6] Shuo Samuel Liu, Andrew Rawicz, Teng Ma, Cheng Zhang, Kyle Lin, Siavash Rezaei, Eion Wu Simon Fraser University, "AN EYE - GAZE TRACKING AND HUMAN COMPUTER INTERFACE SYSTEM FOR PEOPLE WITH ALS AND OTHER LOCKED - IN DISEASES", 2010.

.