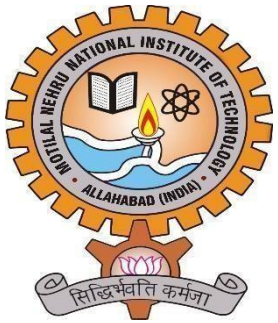
Face Detection and Recognition System using MATLAB

BACHELOR OF TECHNOLOGY in
department of Electronics and Communication Engineering
BY

Deep Shikha (20215018)
Gaurav Kumar Gupta (20215082)
Deepak Kumar Singh (20215141)
Isha Vishwakarma (20215057)

UNDER THE SUPERVISION OF

Dr. Dharmendra Dixit Assistant Professor ECE Department



DEPARTMENT OF Electronics and Communication Engineering
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY, ALLAHABAD
PRAYAGRAJ-211004 April 2024

ACKNOWLEDGEMENT

We, as a team are very grateful and thankful to Prof Dr. Dharmendra Dixit of Electronics and Communication Department, MNNIT Allahabad for providing the opportunity to make this wonderful project in pre-final year.

We also express my heartfelt gratitude to the Department of Electronics and Communication Engineering MNNIT Allahabad for giving us this opportunity, which has enriched our knowledge and experience immensely.

Last but not the least my head bows with reverse before Almighty GOD, who has given us strength, wisdom and will to complete the work.

Deep Shikha Reg. No.: 20215018

Gaurav Kumar Gupta Reg. No.: 20215082

Deepak Kumar Singh Reg. No.: 20215141

Isha Vishwakarma Reg. No.: 20215057

Abstract

This project focuses on the development of a robust face detection and recognition system using MATLAB. The system incorporates state-of-the-art algorithms for face detection and recognition, including the Viola-Jones algorithm for face detection and Eigenfaces for face recognition.

The objectives of the project are to implement a face detection module capable of detecting faces in static images, video streams, and real-time webcam feeds, and to develop a face recognition module that can accurately identify individuals based on their facial features. The project leverages principles from computer vision, image processing, and machine learning to achieve its goals.

The methodology involves the implementation of algorithms for face detection and recognition using MATLAB's Computer Vision Toolbox. The Viola-Jones algorithm is utilized for real-time face detection, while Eigenfaces are employed for face recognition based on principal component analysis (PCA). The project also includes the development of a user-friendly interface to facilitate interaction with the system.

The results of the project demonstrate the effectiveness of the developed system in accurately detecting and recognizing faces in various scenarios. Performance evaluation metrics such as detection rates, recognition accuracy, and computational efficiency are presented and discussed. The project highlights the potential of MATLAB as a powerful tool for implementing facial recognition systems and lays the groundwork for future research and development in this field.

Overall, the project contributes to the advancement of facial recognition technology and its practical applications, offering a reliable and efficient solution for identity verification and authentication in diverse settings.

Motivation

The project is driven by the increasing demand for robust facial recognition technology in various sectors. Facial recognition offers a seamless and secure method for identity verification, enhancing security measures and streamlining user experiences. Its non-intrusive nature makes it particularly appealing for applications requiring quick and convenient authentication.

In today's security-conscious environment, there is a pressing need for reliable authentication mechanisms to protect sensitive information and physical spaces. Facial recognition technology provides a viable solution, replacing traditional authentication methods such as passwords and PINs with a more efficient and user-friendly alternative.

Furthermore, facial recognition has the potential to revolutionize personalized services and experiences by leveraging individual characteristics and preferences. From personalized advertising to tailored user interfaces, the ability to customize offerings based on facial recognition data enhances user engagement and satisfaction.

By developing a comprehensive facial detection and recognition system using MATLAB, this project aims to address current challenges and drive innovation in the field. Through careful research and implementation, we seek to contribute to the advancement and widespread adoption of facial recognition technology.

Introduction

Facial recognition technology has witnessed significant advancements in recent years, emerging as a vital tool in various applications such as security, access control, and personalized user experiences. This project endeavours to develop a robust face detection and recognition system using MATLAB, a powerful platform for numerical computing and algorithm development. The system aims to accurately detect faces in images, videos, and real-time webcam feeds, and to identify individuals based on their facial features. Additionally, the project explores the integration of face detection and recognition algorithms into a userfriendly interface, facilitating seamless interaction with the system.

✈ Objective

The primary objective of this project is to implement a comprehensive face detection and recognition system using MATLAB. Specific objectives include:

- Developing algorithms for face detection using the Viola-Jones method.
- Implementing face recognition techniques based on Eigenfaces.
- Integrating the algorithms into a user-friendly interface for easy interaction.
- Evaluating the performance of the system in terms of accuracy, speed, and usability. ✈ **Technologies Used**

The project utilizes MATLAB and its Computer Vision Toolbox for the implementation of face detection and recognition algorithms. MATLAB provides a comprehensive set of tools for image processing, computer vision, and machine learning, making it an ideal platform for developing advanced facial recognition systems.

Face Detection

✈ Principal

Face detection is a fundamental task in facial recognition systems, involving the localization of human faces within digital images or video frames. The Viola-Jones algorithm is a popular technique for face detection, known for its speed and accuracy. It operates by scanning an image using a cascade of classifiers trained on Haar-like features, which capture variations in pixel intensities associated with facial features.

✈Advantages and Limitations

The Viola-Jones algorithm offers several advantages, including real-time performance, robustness to variations in lighting and facial expressions, and scalability to large datasets. However, it may struggle with detecting faces in extreme poses or under challenging conditions such as occlusions or low-resolution images.

✈ Applications

Face detection has numerous applications across various industries, including security and surveillance, human-computer interaction, and entertainment. It is used in biometric authentication systems, video analytics, social media platforms, and more.

Face Recognition

✈Principal

Face recognition is the process of identifying individuals based on their facial features. The Eigenfaces method is a classic technique for face recognition, which involves representing faces as linear combinations of principal components obtained through principal component analysis (PCA). By comparing the similarity between the eigenface representations of a query face and known faces in a database, the system can recognize individuals.

✈ Advantages and Limitations

Eigenfaces offer advantages such as simplicity, scalability, and robustness to variations in lighting and facial expressions. However, they may struggle with variations in pose, occlusions, and image quality. Additionally, Eigenfaces require a sufficiently large and diverse training dataset to capture the variability in facial appearances.

✈Applications

Face recognition has diverse applications, including access control, surveillance, forensic identification, and personalized user experiences. It is used in law enforcement, banking, retail, healthcare, and more.

Functions Used in the Project

1. vision.CascadeObjectDetector()

Purpose: This function creates a cascade object detector for face detection using the Viola-Jones algorithm.

Description: It initializes a cascade object detector with default parameters for detecting faces in images or video frames.

Usage: faceDetector = vision.CascadeObjectDetector();

2. step(faceDetector, image)

Purpose: This function performs face detection on an input image using the specified cascade object detector.

Description: It applies the cascade object detector to the input image and returns the bounding boxes of detected faces.

Usage: bbox = step(faceDetector, image);

3. detectMinEigenFeatures(image)

Purpose: This function detects corners and feature points in an input grayscale image.

Description: It employs the minimum eigenvalue algorithm to identify regions of high texture variation, which often correspond to corners or key points.

Usage: points = detectMinEigenFeatures(image);

4. vision.PointTracker()

Purpose: This function creates a point tracker object for tracking feature points in video frames.

Description: It initializes a point tracker with default parameters for tracking feature points in consecutive video frames.

Usage: pointTracker = vision.PointTracker();

5. initialize(pointTracker, points, image)

Purpose: This function initializes the point tracker with a set of feature points in the first frame of a video sequence.

Description: It sets the initial position of the feature points and prepares the tracker for subsequent frame-by-frame tracking.

Usage: initialize(pointTracker, points, image);

6. step(pointTracker, image)

Purpose: This function tracks feature points in a video frame using the initialized point tracker.

Description: It updates the positions of the tracked feature points in the current frame based on their positions in the previous frame.

Usage: [points, isFound] = step(pointTracker, image);

7. estimateGeometricTransform(oldPoints, newPoints, 'similarity')

Purpose: This function estimates the geometric transformation (e.g., translation, rotation, scaling) between two sets of corresponding points.

Description: It computes the transformation matrix that best aligns the old points to the new points using a similarity transformation model.

Usage: [tform, oldPoints, newPoints] = estimateGeometricTransform(oldPoints, newPoints, 'similarity');

8. insertShape(image, shape, position)

Purpose: This function inserts geometric shapes (e.g., rectangles, polygons) into an image at specified positions.

Description: It overlays the specified shape onto the input image at the specified position or region of interest.

Usage: image = insertShape(image, shape, position);

9. insertMarker(image, points)

Purpose: This function inserts markers (e.g., plus signs) at specified feature points in an image.

Description: It marks the specified feature points by overlaying markers onto the input image.

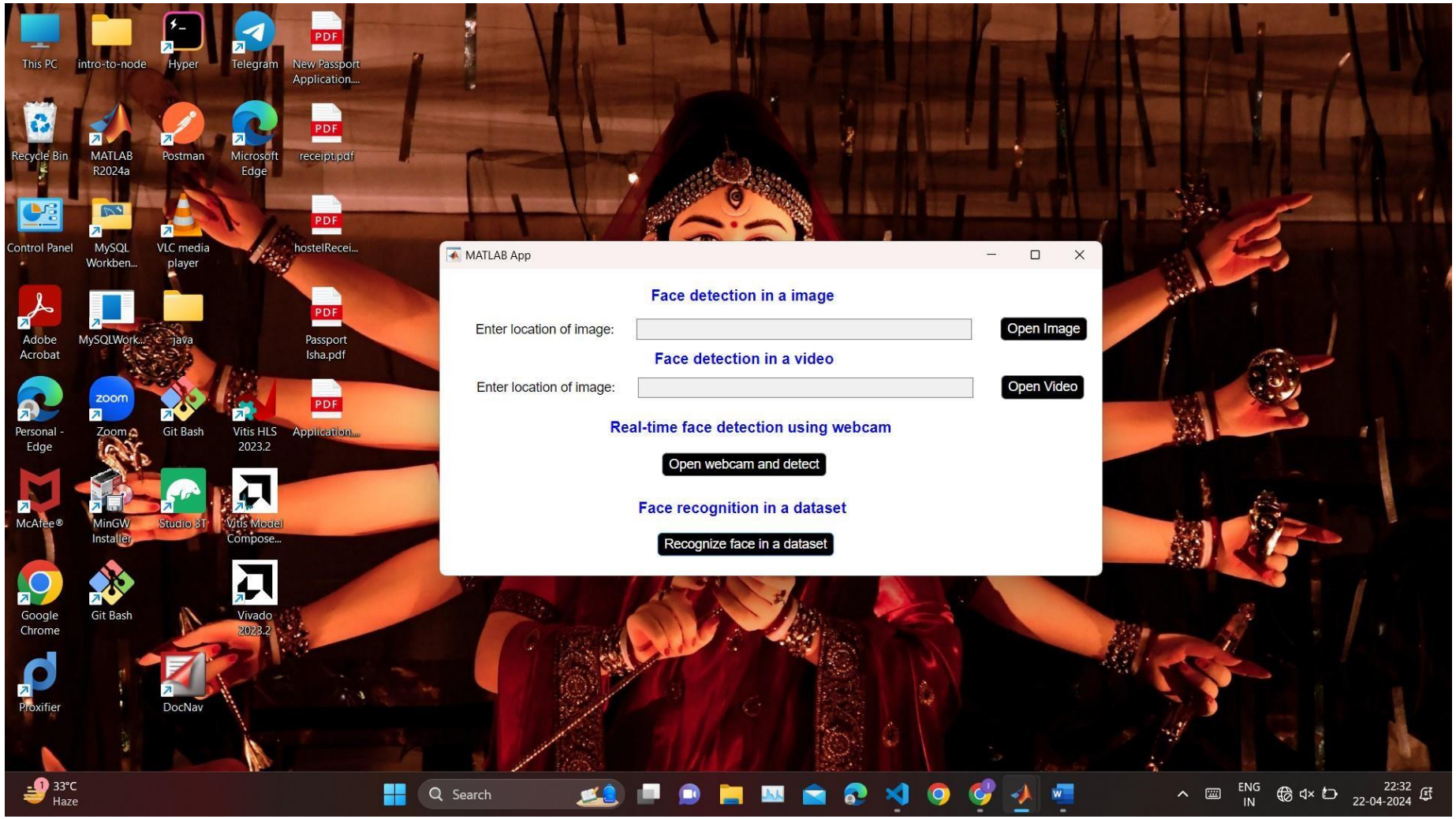
Usage: image = insertMarker(image, points);

Code to detect face on image stored in local storage:

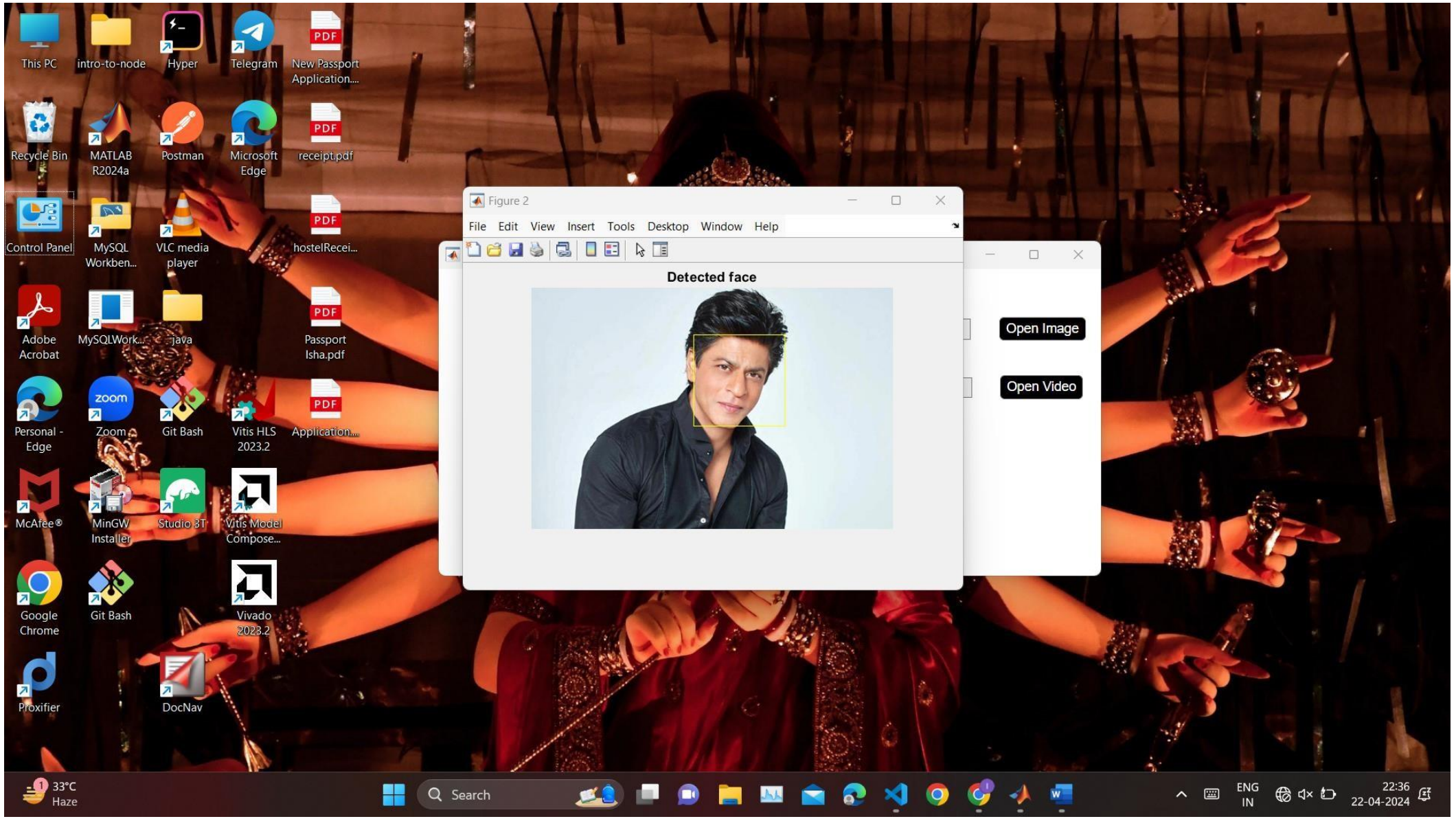
```
%loading the image the_Image      =
imread('download (1) (1).jpg');
[width, height] = size(the_Image);

if          width>320          the_Image          =
imresize(the_Image,[320 NaN]); end
```

```
% Create a cascade detector object.
faceDetector = vision.CascadeObjectDetector();

%finding the bounding box that encloses the face on video frame
face_Location = step(faceDetector, the_Image);

% Draw the returned bounding box around the detected face.
the_Image = insertShape(the_Image, 'Rectangle', face_Location);
figure;
imshow(the_Image); title('Detected
face');
```

Home page UI



Detected face in image



Code to detect face in video stored in local storage:

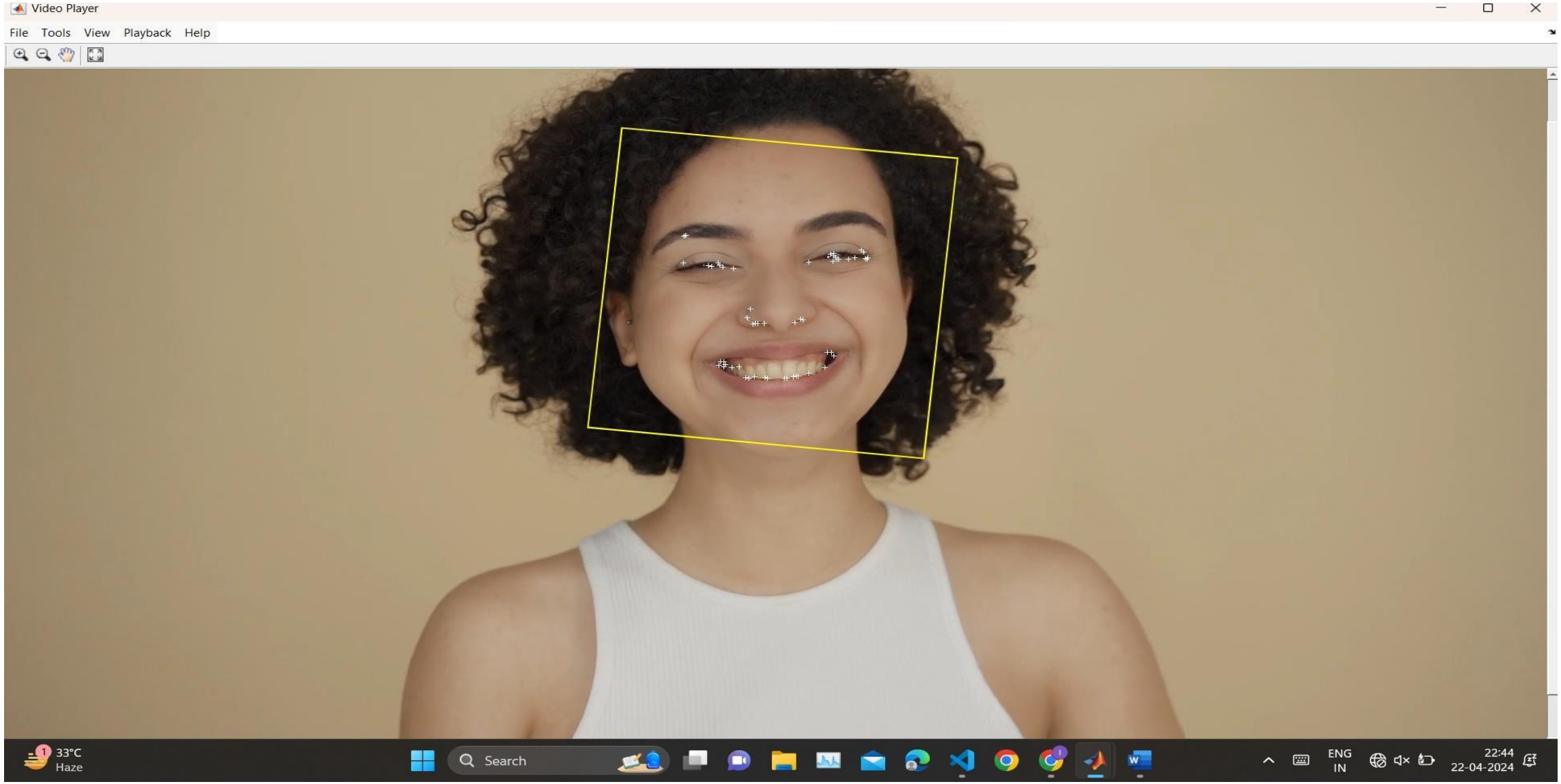
```
% Open the video file the_Video =
VideoReader('video.mp4'); % Read the
first frame of the video video_Frame =
readFrame(the_Video);

% Create a cascade object detector for face detection
face_Detector = vision.CascadeObjectDetector();
% Detect faces in the first frame
location_of_the_Face = step(face_Detector, video_Frame);

% Draw rectangles around detected faces detected_Frame =
insertShape(video_Frame,'Rectangle', location_of_the_Face);
% Convert the bounding box of the first detected face to points rectangle_to_Points = bbox2points
(location_of_the_Face(1,:)); % Detect feature points within the detected face region feature_Points
= detectMinEigenFeatures(rgb2gray(detected_Frame), 'ROI', location_of_the_Face);

% Create a point tracker object
pointTracker = vision.PointTracker('MaxBidirectionalError', 2);
% Extract the location of the feature points
feature_Points = feature_Points.Location;
% Initialize the point tracker with the feature points and the first frame
Initialize (pointTracker, feature_Points, detected_Frame);
```

```
% Set up parameters for displaying the video
left = 100; bottom = 100; width =
size(detected_Frame, 2); height =
size(detected_Frame, 1);
video_Player = vision.VideoPlayer('Position', [left bottom width height]);

% Store the feature points for the previous frame
previous_Points = feature_Points;

% Loop through each frame of the video
while hasFrame(the_Video)

    video_Frame = readFrame(the_Video);
    % Track feature points in the current frame
    [feature_Points, isFound] = step(pointTracker, video_Frame);
 % Select only the found feature points
new_Points = feature_Points(isFound,:);
    old_Points = previous_Points(isFound,:);

    % Check if enough feature points are found for transformation estimation
if size(new_Points, 1) >= 2
```

```
        % Estimate the geometric transformation between the old and new points
[transformed_Rectangle, old_Points, new_Points] = ...
estimateGeometricTransform(old_Points, new_Points, ...
            'similarity', 'MaxDistance', 4);
        % Transform the bounding box points using the estimated transformation
        rectangle_to_Points = transformPointsForward(transformed_Rectangle, rectangle_to_Points);

    % Reshape the transformed points into a polygon
reshaped_Rectangle = reshape(rectangle_to_Points', 1, []);
% Draw the transformed polygon on the current frame
    detected_Frame = insertShape(video_Frame, 'Polygon', reshaped_Rectangle,'LineWidth', 2);

    % Mark the new feature points on the frame
    detected_Frame = insertMarker(detected_Frame, new_Points, '+', 'Color', 'White');

 % Update the previous feature points for the next iteration
previous_Points = new_Points;
    % Set the new feature points for the point tracker
setPoints(pointTracker, previous_Points);
   end
   % Display the current frame with tracking information
step(video_Player, detected_Frame) end
```

```
% Release the video player object
release(video_Player);
```

Detected face in Video



Code to detect face in real time through web cam:

```
%open webcam of system
        cam=webcam();

        %set resolution of the webcam
cam.Resolution='640x480';

        %set first video frame by taking snapshot from webcam
video_Frame = snapshot(cam);

        %open the video player
        video_Player = vision.VideoPlayer('Position', [100 100 640 480]);

        %create cascade object detector
face_Detector=vision.CascadeObjectDetector();

        % Create a point tracker object
        point_Tracker=vision.PointTracker('MaxBidirectionalError', 2);
run_loop=true;          number_of_Points=0;
        frame_Count=0;

while run_loop && frame_Count<400
```

```
        %take snapshot from webcam for video frame
video_Frame=snapshot(cam);

        %convert video frame to gray frame
gray_Frame = rgb2gray(video_Frame);

        %increase frame count
frame_Count=frame_Count+1;

        %for detecting face movement
if number_of_Points<10
%creating a face rectangle
            face_Rectangle = face_Detector.step(gray_Frame);
            if ~isempty(face_Rectangle)
               % Detect feature points within the detected face region
               points = detectMinEigenFeatures(gray_Frame, 'ROI', face_Rectangle(1, :));

               %get location of feature points
xy_Points = points.Location;

               %get the number of feature points
number_of_Points=size(xy_Points, 1); %release previous point tracker
```

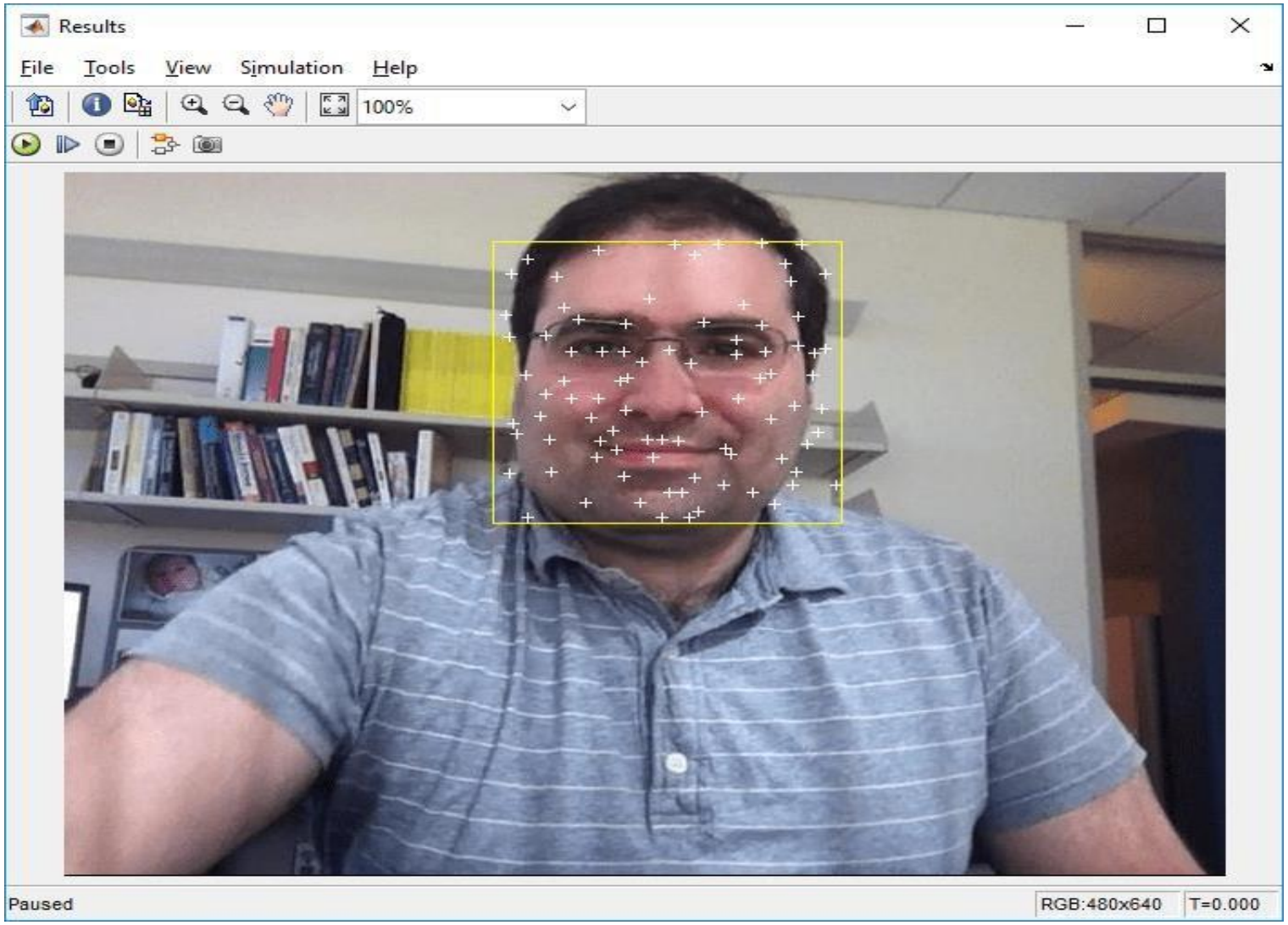
```
                release(point_Tracker);

                %initialize new point tracker object
                initialize(point_Tracker, xy_Points, gray_Frame);

                %get previous feature points
previous_Points = xy_Points;

                % Convert the bounding box of the detected face to points
rectangle=bbox2points(face_Rectangle(1, :));

                % Reshape the transformed points into a polygon
face_Polygon=reshape(rectangle', 1, [])
              % Draw the transformed polygon on the current frame
                video_Frame=insertShape(video_Frame, 'Polygon', face_Polygon, 'LineWidth', 3);

                % Mark the new feature points on the frame
                video_Frame=insertMarker(video_Frame, xy_Points, '+', 'MarkerColor','white');
end             else

% Track feature points in the current frame
              [xy_Points, isFound] = step(point_Tracker, gray_Frame);
```

```
            % Select only the found feature points
new_Points = xy_Points(isFound, :);
old_Points = previous_Points(isFound, :);

            %get size of new feature points
            number_of_Points=size(new_Points, 1);
            % Check if enough feature points are found for transformation estimation
if number_of_Points>=10
                % Estimate the geometric transformation between the old and new points
                [xform, old_Points, new_Points] = estimateGeometricTransform(old_Points, new_Points, 'similarity',
'MaxDistance',4);

                % Transform the bounding box points using the estimated transformation
rectangle = transformPointsForward(xform, rectangle);

                % Reshape the transformed points into a polygon
face_Polygon = reshape(rectangle', 1, []);
 % Draw the transformed polygon on the current frame
                video_Frame=insertShape(video_Frame, 'Polygon', face_Polygon, 'LineWidth', 3);

                 % Mark the new feature points on the frame
                video_Frame=insertMarker(video_Frame, new_Points, '+', 'MarkerColor','white');
```

```
                % Update the previous feature points for the next iteration
previous_Points = new_Points;

                % Set the new feature points for the point tracker
setPoints(point_Tracker, previous_Points);              end
end
            % Display the current frame with tracking information
step(video_Player, video_Frame);

           %check if video player is still open
           run_loop=isOpen(video_Player);
end
 %clear camera
        clear('cam');

        %release video player
release(video_Player);

        %release point tracker
release(point_Tracker);
```

```
        %release face detector
release(face_Detector);
```

detected face in real time through web cam:



Code to upload the data sheet for face recognition:

```
function output_value = load_database();
persistent loaded; persistent
numeric_Image; if(isempty(loaded))
all_Images = zeros(10304,40);     for
i=1:40        cd(strcat('s',num2str(i)));
for j=1:10
          image_Container    =    imread(strcat(num2str(j),'.pgm'));                                    all_Images(:,(i-
1)*10+j)=reshape(image_Container,size(image_Container,1)*size(image_Container,2),1);        end
       display('Loading Database');
cd ..     end     numeric_Image =
uint8(all_Images); end
loaded = 1;
output_value = numeric_Image;
```

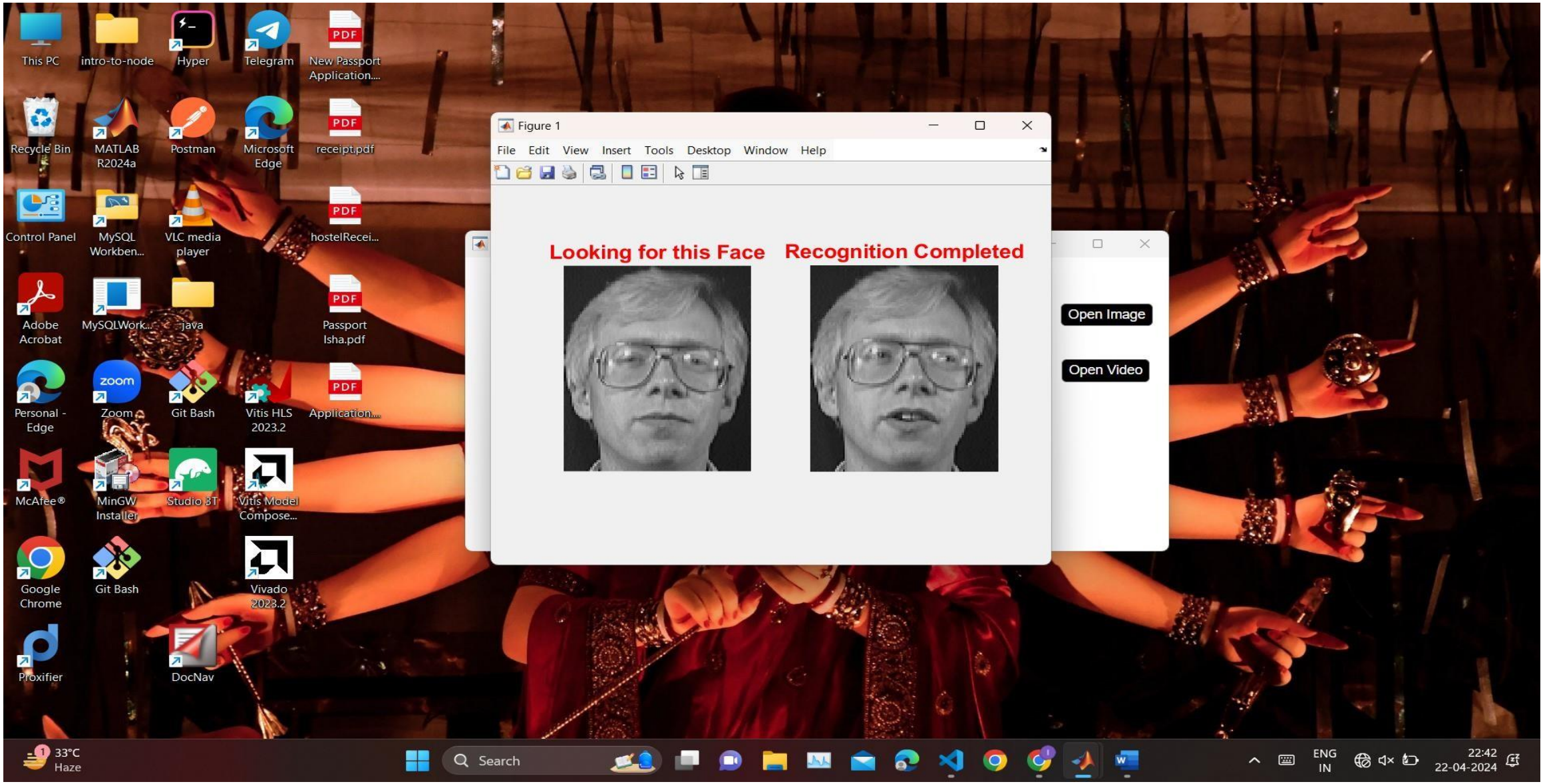
Code for face recognition

```
loaded_Image=load_database();
random_Index=round(400*rand(1,1));
random_Image=loaded_Image(:,random_Index);
rest_of_the_images=loaded_Image(:,[1:random_Index-1 random_Index+1:end]);        image_Signature=20;
```

```
white_Image=uint8(ones(1,size(rest_of_the_images,2)));
mean_value=uint8(mean(rest_of_the_images,2));              mean_Removed=rest_of_the_images-
uint8(single(mean_value)*single(white_Image));
L=single(mean_Removed)'*single(mean_Removed);
[V,D]=eig(L);
V=single(mean_Removed)*V;
V=V(:,end:-1:end-(image_Signature-1));
all_image_Signatire=zeros(size(rest_of_the_images,2),image_Signature);
for i=1:size(rest_of_the_images,2);
all_image_Signatire(i,:)=single(mean_Removed(:,i))'*V;  end
subplot(121);
imshow(reshape(random_Image,112,92));
title('Looking for this Face','FontWeight','bold','Fontsize',16,'color','red');
subplot(122);
p=random_Image-mean_value;
s=single(p)'*V; z=[];
for i=1:size(rest_of_the_images,2)     z=[z,norm(all_image_Signatire(i,:)-
s,2)];
if(rem(i,20)==0),imshow(reshape(rest_of_the_images(:,i),112,92)),end;
drawnow; end
[a,i]=min(z);
subplot(122);
```

imshow(reshape(rest_of_the_images(:,i),112,92));

title('Recognition Completed','FontWeight','bold','Fontsize',16,'color','red'); **Face recognition**



Conclusion

In conclusion, the project demonstrates the feasibility and effectiveness of using MATLAB for developing a face detection and recognition system. By leveraging the Viola-Jones algorithm for face detection and the Eigenfaces method for face recognition, the system achieves accurate and efficient identification of individuals. The project highlights the potential of facial recognition technology for enhancing security, improving user experiences, and driving innovation across various domains.

References

[1] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. CVPR.

[2] Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. Journal of cognitive neuroscience.

[3] MATLAB Documentation. (n.d.). Computer Vision Toolbox. Retrieved from https://www.mathworks.com/products/computer-vision.html .