##### **Experiment 1**

##### **Experiment 2**

Code:

import folium

from geopy.geocoders import Nominatim

import requests

# Specify a user agent string for Nominatim

geolocator = Nominatim(user\_agent="my-app")

# Specify the address for which you want to get the latitude and longitude

address = "Dwarkadas J. Sanghvi College of Engineering, Vile Parle West, Mumbai, Maharashtra 400056, India"

# Use geolocator to get the location information for the address

location = geolocator.geocode(address)

# Get the latitude and longitude

lat, lng = location.latitude, location.longitude

# Create a map centered on the location

m = folium.Map(location=[lat, lng], zoom\_start=14)

folium.Marker([lat, lng],icon=folium.Icon(color='green'), popup=location.address).add\_to(m)

# Set up the search parameters (fire stations and police stations)

fire\_police\_query = ['fire\_station', 'police']

radius\_fire\_police = 10000 # 10 km

# Make the API request for fire stations and police stations

Url\_fire\_police = f"https://overpassapi.de/api/interpreter?data=[out:json];node(around:{radius\_fire\_police},{lat},{lng})[amenity~'{'|'.joi

n(fire\_police\_query)}'];out;"

response\_fire\_police = requests.get(url\_fire\_police)

data\_fire\_police = response\_fire\_police.json()

# Loop through the results and add markers to the map

for result in data\_fire\_police["elements"]:

if "lat" in result and "lon" in result:

lat, lon = result["lat"], result["lon"]

folium.Marker([lat, lon], icon=folium.Icon(color='red'), popup=f"{lat}, {lon}").add\_to(m)

lat, lng = location.latitude, location.longitude

# Set up the search parameters (hospitals, clinics, doctors, pharmacies)

emergency\_query = ["hospital", "clinic", 'doctors', 'pharmacy']

radius\_emergency = 2000 # 2 km

# Make the API request for emergency services

url\_emergency = f"https://overpassapi.de/api/interpreter?data=[out:json];node(around:{radius\_emergency},{lat},{lng})[amenity~'{'|'.joi

n(emergency\_query)}'];out;"

response\_emergency = requests.get(url\_emergency)

data\_emergency = response\_emergency.json()

# Loop through the results and add markers to the map

for result in data\_emergency["elements"]:

if "lat" in result and "lon" in result:

lat, lon = result["lat"], result["lon"]

folium.Marker([lat, lon], icon=folium.Icon(color='blue'), popup=f"{lat}, {lon}").add\_to(m)

# Display the map

m.save("map\_fire\_police.html")

lat, lng = location.latitude, location.longitude

print(lat, lng)

# Set up the search parameters (hospitals, clinics, doctors, pharmacies)

emergency\_edu = ["school","college"]

radius\_edu = 5000 # 2 km

# Make the API request for emergency services

url\_edu = f"https://overpassapi.de/api/interpreter?data=[out:json];node(around:{radius\_edu},{lat},{lng})[amenity~'{'|'.join(emer

gency\_edu)}'];out;"

response\_edu = requests.get(url\_edu)

data\_edu = response\_edu.json()

# Loop through the results and add markers to the map

for result in data\_edu["elements"]:

if "lat" in result and "lon" in result:

lat, lon = result["lat"], result["lon"]

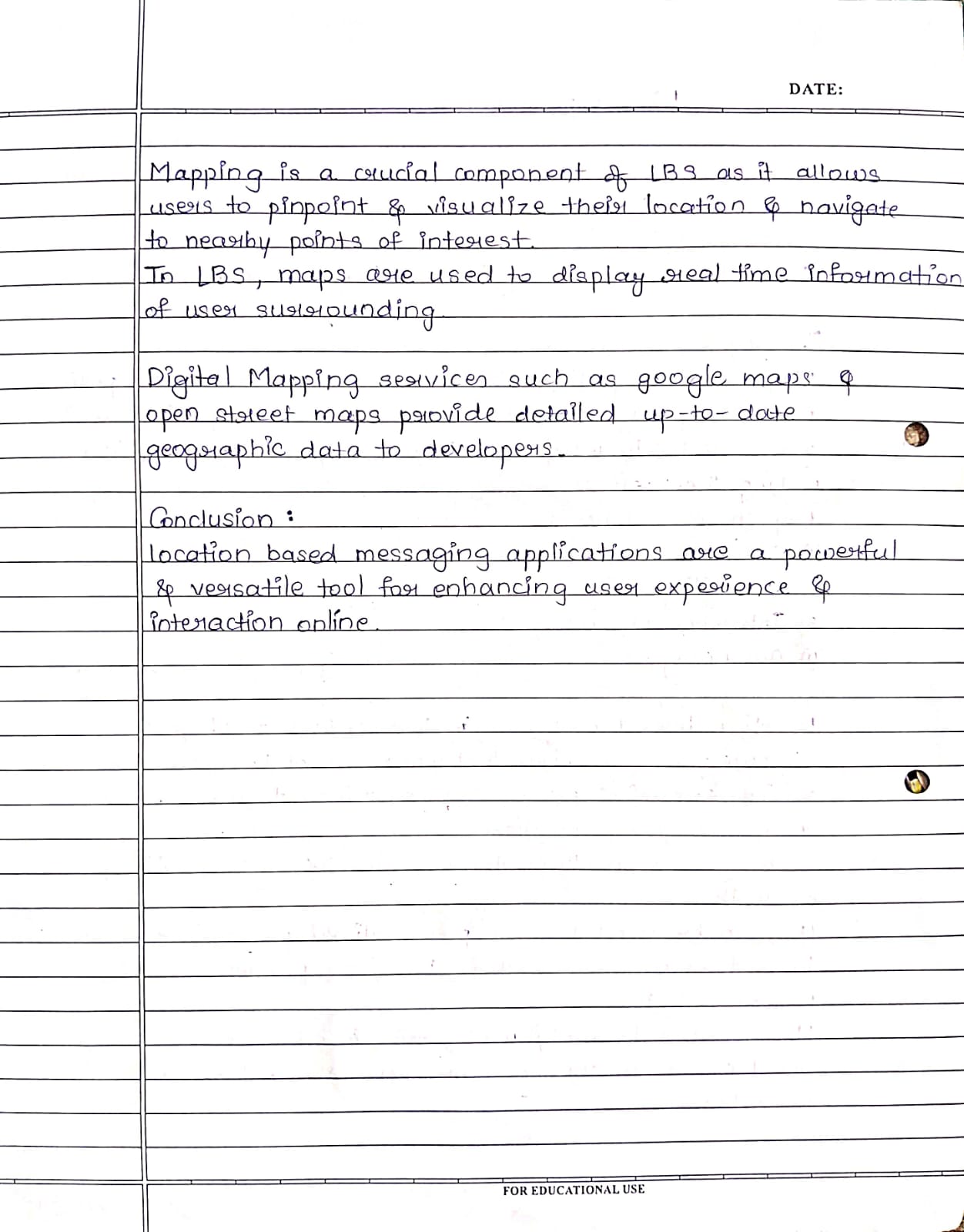
folium.Marker([lat, lon], icon=folium.Icon(color='orange'), popup=f"{lat}, {lon}").add\_to(m)

# Display the map

m.save("map\_fire\_police.html")

Output :



****

##### **Experiment 3**

Code :

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Display a map on a webpage</title>

<meta name="viewport" content="initial-scale=1, maximum-scale=1, userscalable=no">

<link href="https://api.mapbox.com/mapbox-gl-js/v2.13.0/mapbox-gl.css" rel="stylesheet">

<script src="https://api.mapbox.com/mapbox-gl-js/v2.13.0/mapboxgl.js"></script>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>

<style>

body {

margin: 0;

padding: 0;

}

#map {

position: absolute;

top: 0;

bottom: 0;

width: 100%;

}

</style>

</head>

<body>

<div id="map"></div>

<script>

var lat, lon;

function getLocation() {

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(showPosition);

}

}

function showPosition(position) {

lat = position.coords.latitude;

lon = position.coords.longitude;

console.log(lat, lon);

plt(lon, lat);

}

</script>

<script>

getLocation();

function nearby(lon, lat, map\_obj, val) {

let txt = "museum";

if (val == 1) {

txt = "college";

}

else if (val == 2) {

txt = "beach";

}

else if (val == 4) {

txt = "theater";

}

else if (val == 5) {

txt = "amusement";

}

else if (val == 3) {

txt = "garden";

}

else {

txt = "restaurant";

}

const settings = {

"async": true,

"crossDomain": true,

"url":"https://api.mapbox.com/geocoding/v5/mapbox.places/"+txt+".json?type=poi&proximity="+lon+"%2C"+lat+"&access\_token=pk.eyJ1IjoibWloaXIxMDUzIiwiYSI6ImNsczlzeXk3MjA5ejAya25xajUzcjkwenMifQ.eievf1bHUoEuCIn0WjYn4A","method": "GET",

"headers": {

"Accept": "\*/\*"

}

};

$.ajax(settings).done(function (response) {

console.log(response.features);

response.features.map((item) => {

console.log(item.center.reverse());

var marker = new mapboxgl.Marker({ color: 'yellow' })

.setLngLat(item.center.reverse())

.setPopup(new

mapboxgl.Popup().setHTML("<p>" + item.place\_name + "</p>"))

.addTo(map\_obj); }) }); }

function plt(lon, lat) {

mapboxgl.accessToken = "pk.eyJ1IjoibWloaXIxMDUzIiwiYSI6ImNsczlzeXk3MjA5ejAya25xajUzcjkwenMifQ.eievf1bHUoEuCIn0W jYn4A";

const map = new mapboxgl.Map({ container: 'map', // container ID

// Choose from Mapbox's core styles, or make your own style withMapbox Studio

style: 'mapbox://styles/mapbox/streets-v12', // style URL

center: [lon, lat], // starting position [lng, lat]

zoom: 20 // starting zoom

});

let foo = prompt('How is your mood?\n1. Happy\n2. Sad\n3. Angry\n4.Bored\n5Excited\n');

let val = parseInt(foo);

nearby(lon, lat, map, val);

const marker1 = new mapboxgl.Marker({ color: 'green' }) .setLngLat([lon, lat]) .setPopup(new mapboxgl.Popup().setHTML("You are here!")) .addTo(map);

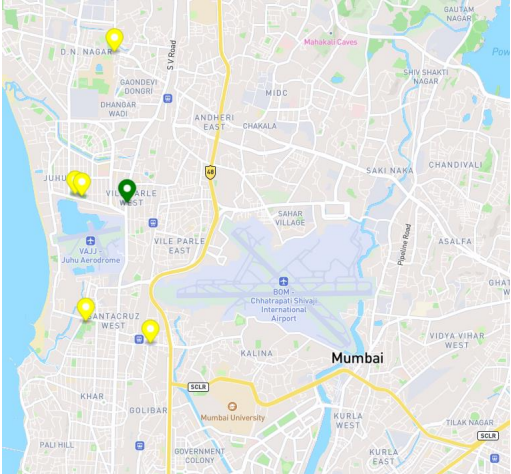
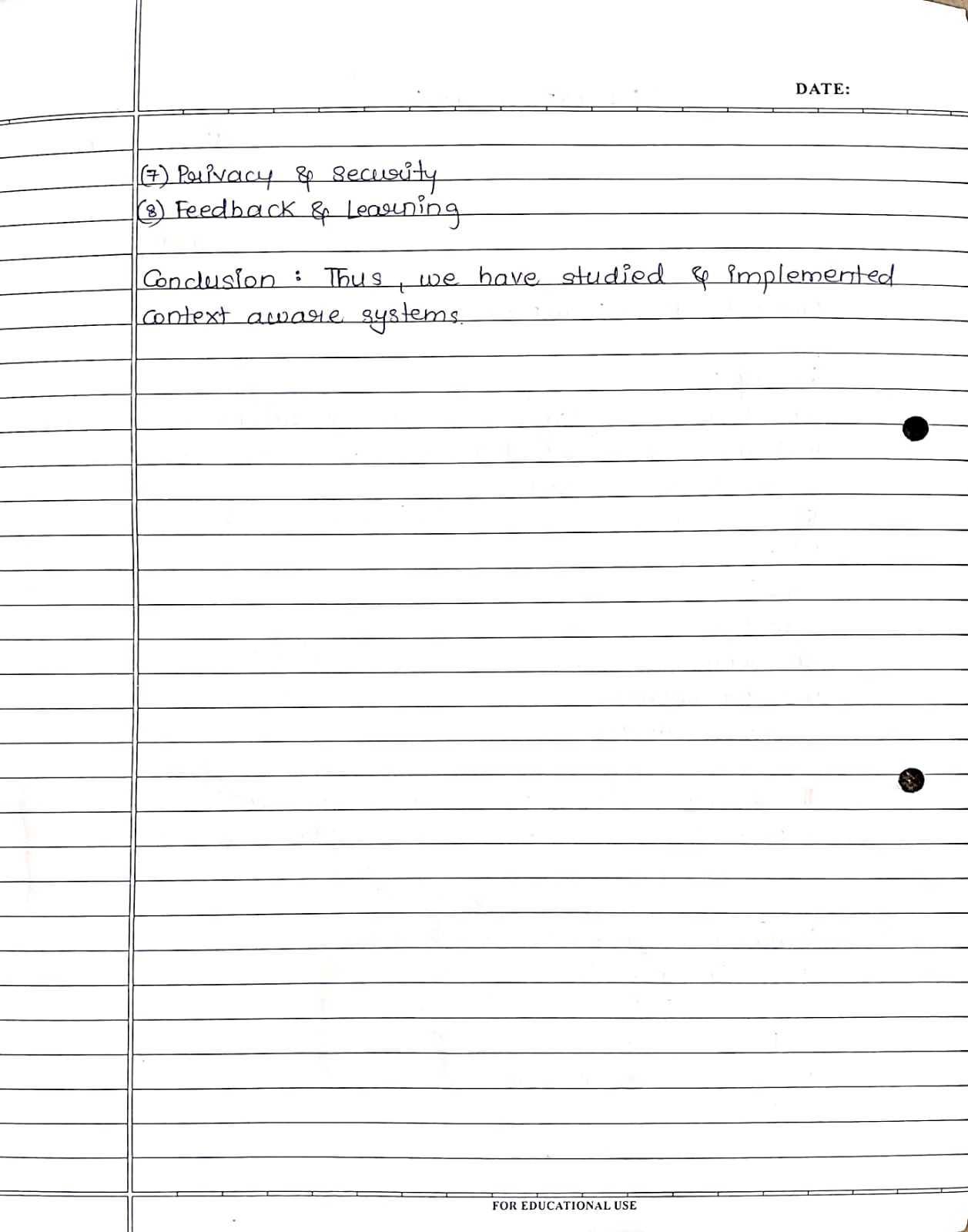
}

</script>

</body>

</html>

Output :

****

##### **Experiment 4**

##### 

Project Overview

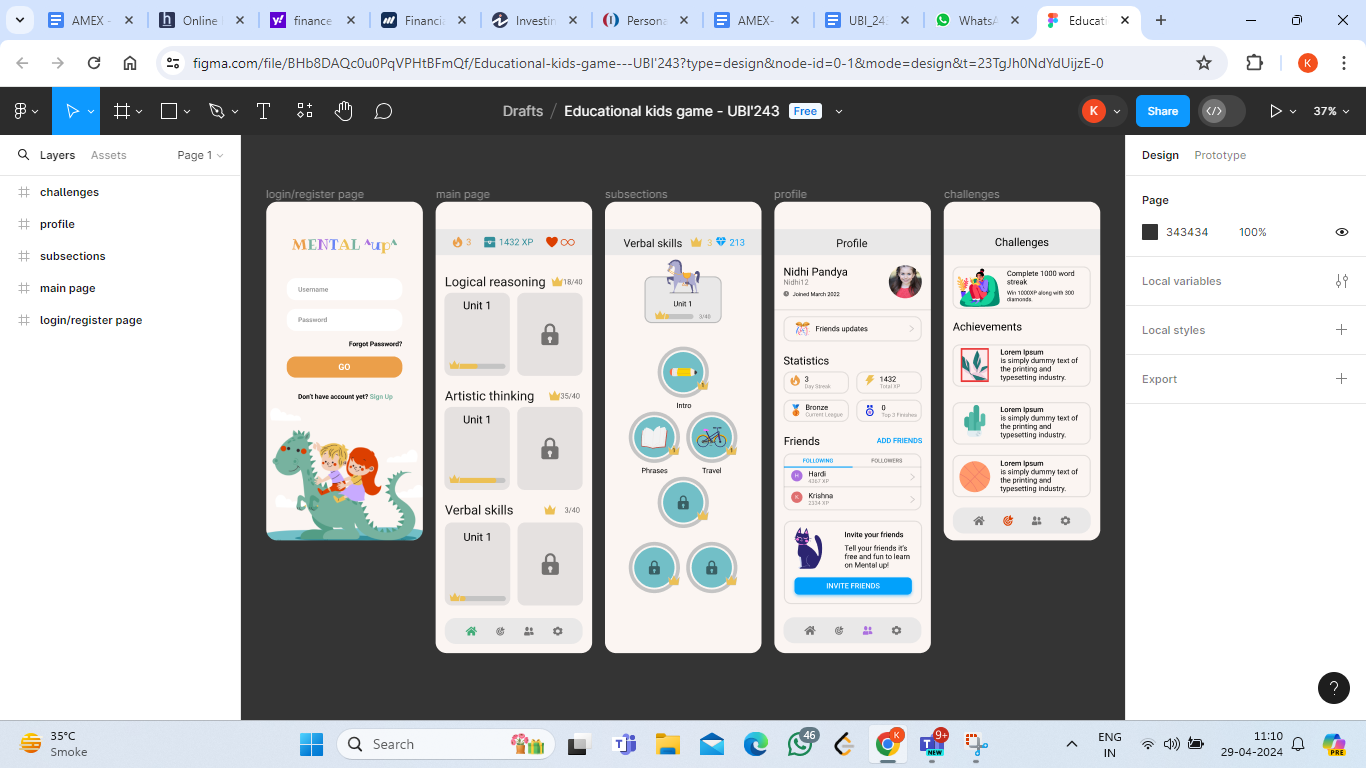
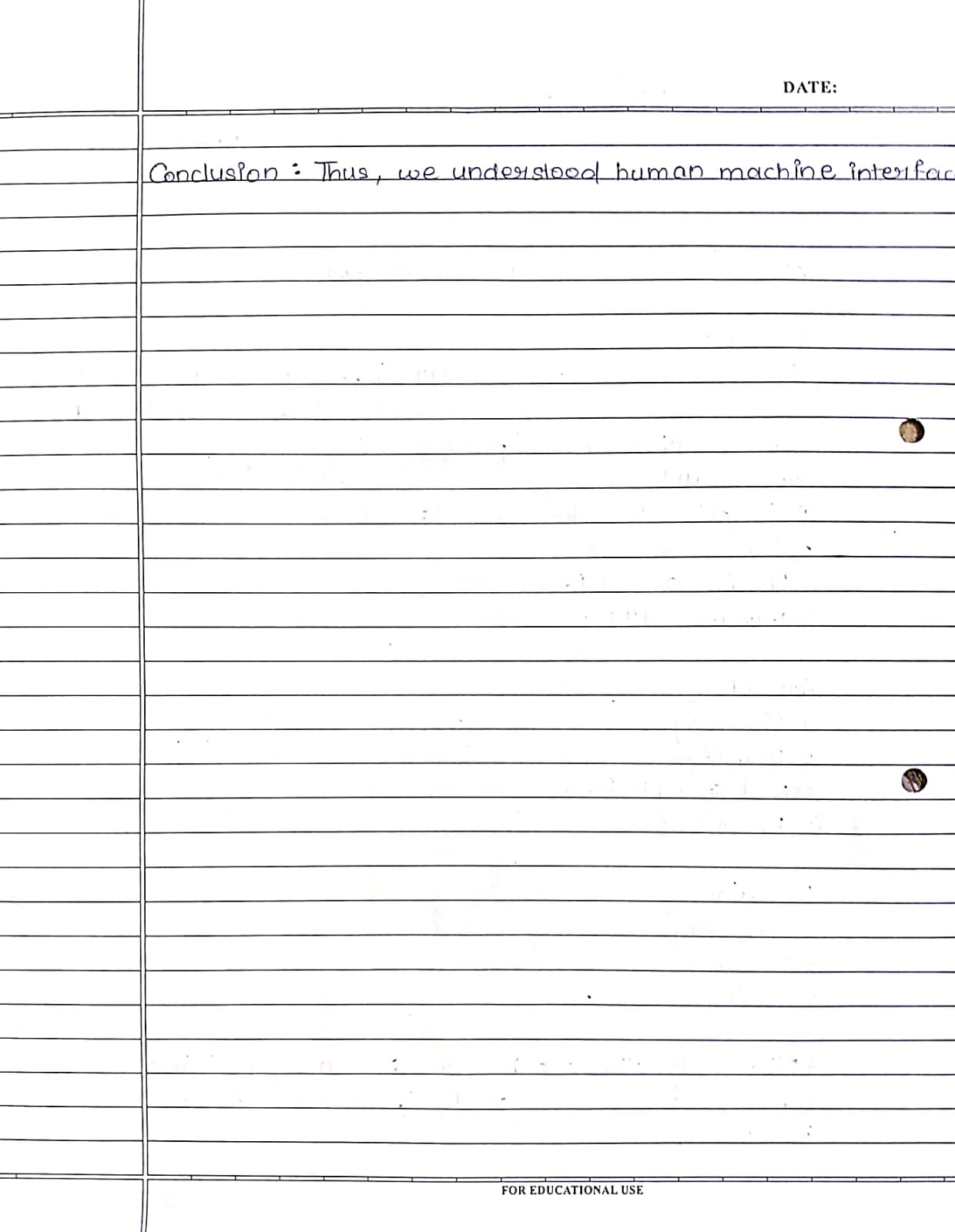
In the realm of children's education, the integration of ubiquitous computing technologies has the potential to revolutionize learning experiences by providing seamless access to educational resources and fostering interactive engagement.

The project focuses on the development of a children's learning app that harnesses the principles of ubiquitous computing to create a dynamic and interactive educational platform.

Key features of the app include:

1. Adaptive Interface: The app employs adaptive user interfaces that adjust dynamically based on users' preferences, learning styles, and progress. This ensures that each child receives a personalized learning experience tailored to their individual needs.
2. Context-Awareness: By leveraging sensor data from mobile devices, the app incorporates context-awareness to adapt its content and activities to the user's environment. For example, outdoor learning activities may be suggested when the device detects that the child is in a park or garden.
3. Multi-Modal Interaction: The app supports multi-modal interaction, allowing children to engage with content through a variety of input methods such as touch, voice, gestures, and even physical manipulation of objects. This promotes inclusivity and accommodates diverse learning preferences.
4. Collaborative Learning: The app facilitates collaborative learning experiences by enabling children to interact with peers, teachers, and mentors in real-time, regardless of their physical location. Collaborative activities, such as group projects or multiplayer games, encourage teamwork and social interaction.
5. Gamification: To enhance engagement and motivation, the app incorporates gamification elements such as rewards, badges, challenges, and progress tracking. These gamified features encourage active participation and make learning enjoyable and rewarding.

<https://www.figma.com/file/BHb8DAQc0u0PqVPHtBFmQf/Educational-kids-game---UBI'243?type=design&node-id=0-1&mode=design&t=23TgJh0NdYdUijzE-0>

****

##### 

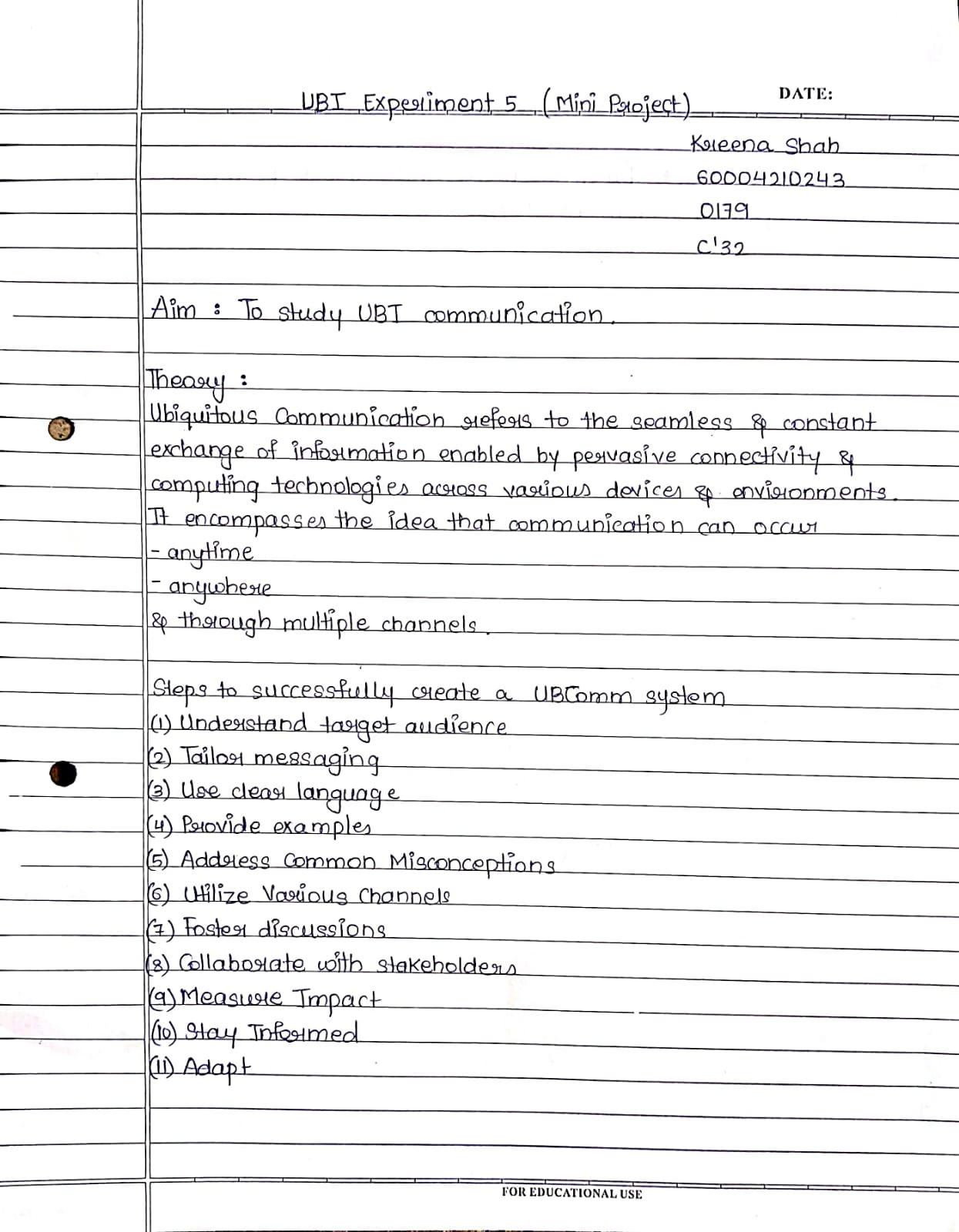
##### 

##### 

##### 

##### 

##### **Experiment 5**



Project Overview

The project aims to leverage gesture-based controls to enable intuitive interactions with digital content. Specifically, the system utilizes hand gestures captured through a camera to perform various actions, including drawing, annotation, and slide navigation. The following gestures are supported:

Single Finger Gesture: Drawing/Annotation

Two-Finger Gesture: Cursor Movement

Three-Finger Gesture: Erasing

Swipe Gesture (Left/Right): Slide Navigation

By mapping specific hand gestures to predefined actions, users can seamlessly interact with digital content without the need for traditional input devices such as keyboards or mice.

The project addresses several challenges and limitations associated with conventional input methods and interaction techniques:

Accessibility: Gesture-based controls offer an intuitive and accessible alternative to traditional input devices, catering to individuals with mobility impairments or disabilities.

Natural Interaction: By mimicking real-world gestures and movements, gesture-based controls provide a more natural and immersive user experience, enhancing user engagement and satisfaction.

Efficiency: The use of gestures for common tasks such as drawing, annotation, and slide navigation streamlines the interaction process, reducing the cognitive load and improving efficiency.

Versatility: Gesture-based controls are versatile and adaptable, allowing for customization and expansion to support a wide range of applications and use cases beyond the scope of conventional input methods.

Code :

from cvzone.HandTrackingModule import HandDetector

import cv2

import os

import numpy as np

# Parameters

width, height = 800, 400

gestureThreshold = 200

folderPath = "Presentation"

# Camera Setup

cap = cv2.VideoCapture(0)

cap.set(2, width)

cap.set(3, height)

# Hand Detector

detectorHand = HandDetector(detectionCon=0.8, maxHands=1)

# Variables

imgList = []

delay = 30

buttonPressed = False

counter = 0

drawMode = False

imgNumber = 0

delayCounter = 0

annotations = [[]]

annotationNumber = -1

annotationStart = False

hs, ws = int(60 \* 1), int(100 \* 1) # width and height of small image

# Get list of presentation images

pathImages = sorted(os.listdir(folderPath), key=len)

print(pathImages)

while True:

# Get image frame

success, img = cap.read()

img = cv2.flip(img, 1)

pathFullImage = os.path.join(folderPath, pathImages[imgNumber])

imgCurrent = cv2.imread(pathFullImage)

# Find the hand and its landmarks

hands, img = detectorHand.findHands(img)

# Draw Gesture Threshold line

cv2.line(img, (0, gestureThreshold), (width, gestureThreshold), (0, 255, 0), 10)

if hands and buttonPressed is False: # If hand is detected

hand = hands[0]

cx, cy = hand["center"]

lmList = hand["lmList"] # List of 21 Landmark points

fingers = detectorHand.fingersUp(hand) # List of which fingers are up

# Constrain values for easier drawing

xVal = int(np.interp(lmList[8][0], [width // 2, width], [0, width]))

yVal = int(np.interp(lmList[8][1], [150, height-150], [0, height]))

indexFinger = xVal, yVal

if cy <= gestureThreshold: # If hand is at the height of the face

if fingers == [1, 0, 0, 0, 0]:

print("Left")

buttonPressed = True

if imgNumber > 0:

imgNumber -= 1

annotations = [[]]

annotationNumber = -1

annotationStart = False

if fingers == [0, 0, 0, 0, 1]:

print("Right")

buttonPressed = True

if imgNumber < len(pathImages) - 1:

imgNumber += 1

annotations = [[]]

annotationNumber = -1

annotationStart = False

if fingers == [0, 1, 1, 0, 0]:

cv2.circle(imgCurrent, indexFinger, 12, (0, 0, 255), cv2.FILLED)

if fingers == [0, 1, 0, 0, 0]:

if annotationStart is False:

annotationStart = True

annotationNumber += 1

annotations.append([])

print(annotationNumber)

annotations[annotationNumber].append(indexFinger)

cv2.circle(imgCurrent, indexFinger, 12, (0, 0, 255), cv2.FILLED)

else:

annotationStart = False

if fingers == [0, 1, 1, 1, 0]:

if annotations:

annotations.pop(-1)

annotationNumber -= 1

buttonPressed = True

else:

annotationStart = False

if buttonPressed:

counter += 1

if counter > delay:

counter = 0

buttonPressed = False

for i, annotation in enumerate(annotations):

for j in range(len(annotation)):

if j != 0:

cv2.line(imgCurrent, annotation[j - 1], annotation[j], (0, 0, 200), 12)

imgSmall = cv2.resize(img, (ws, hs))

h, w, \_ = imgCurrent.shape

imgCurrent[0:hs, w - ws: w] = imgSmall

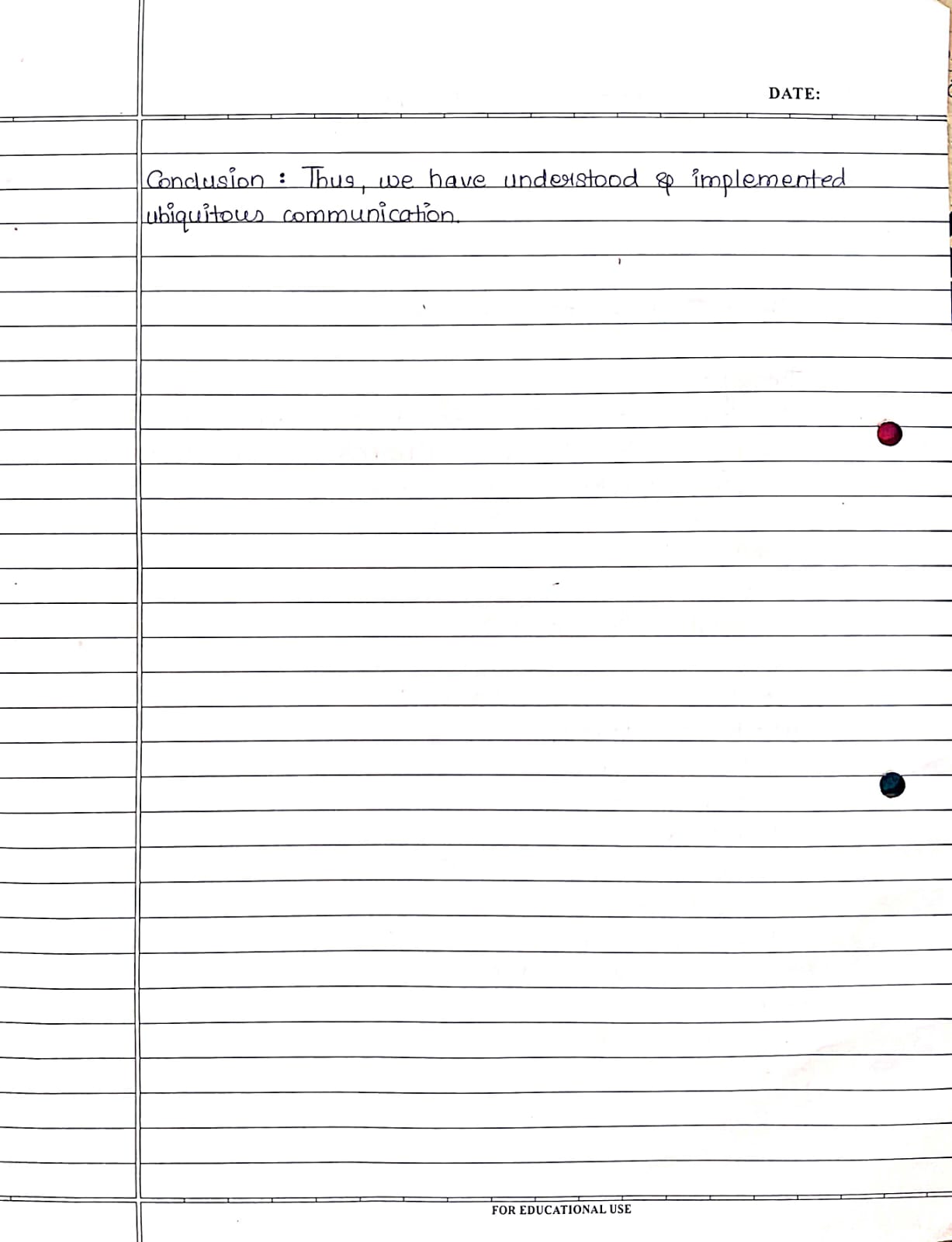
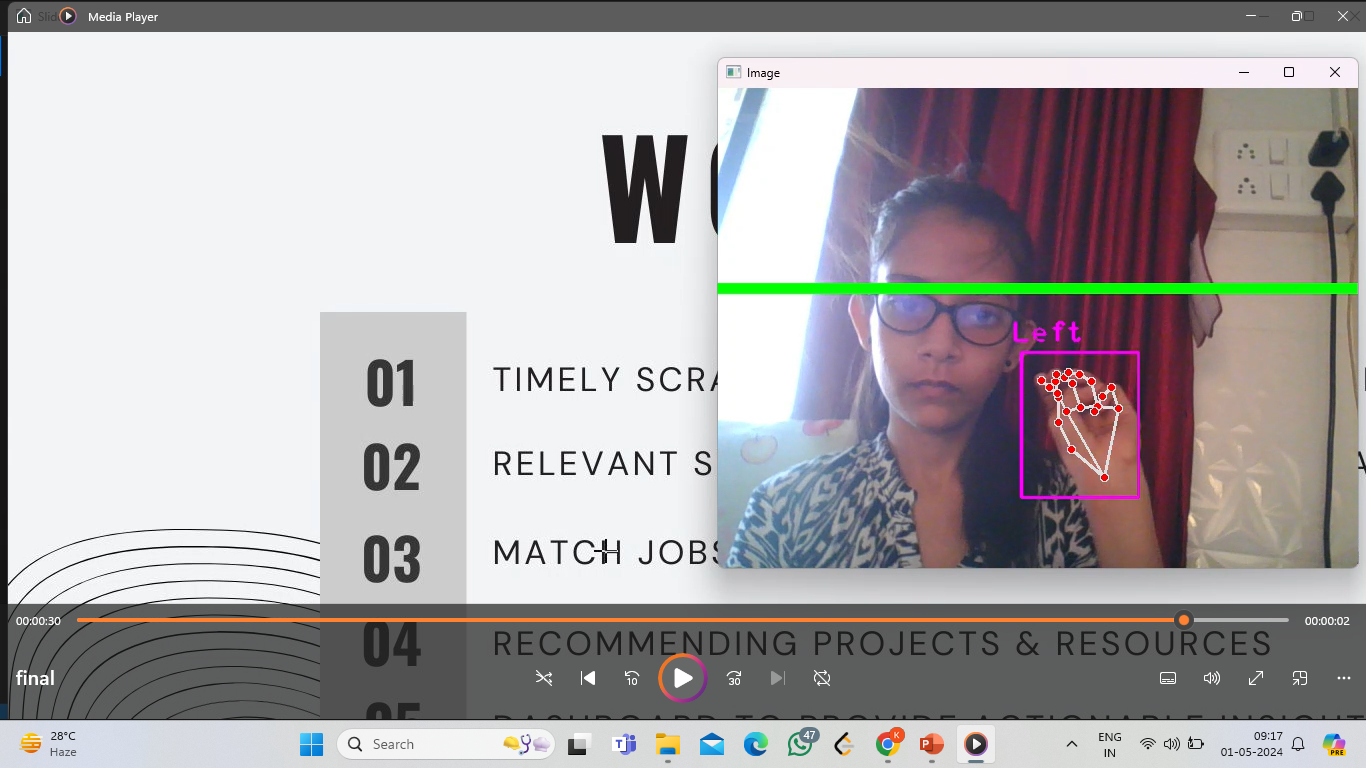
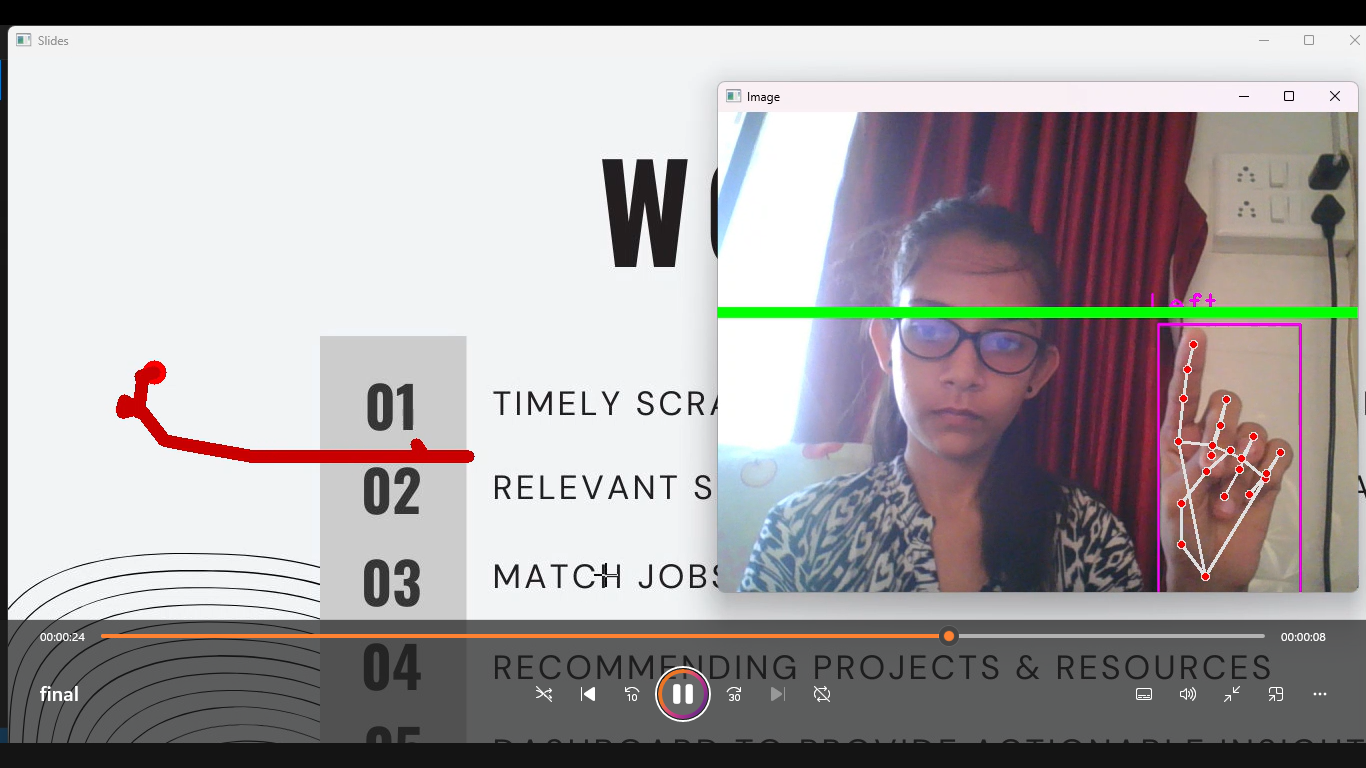
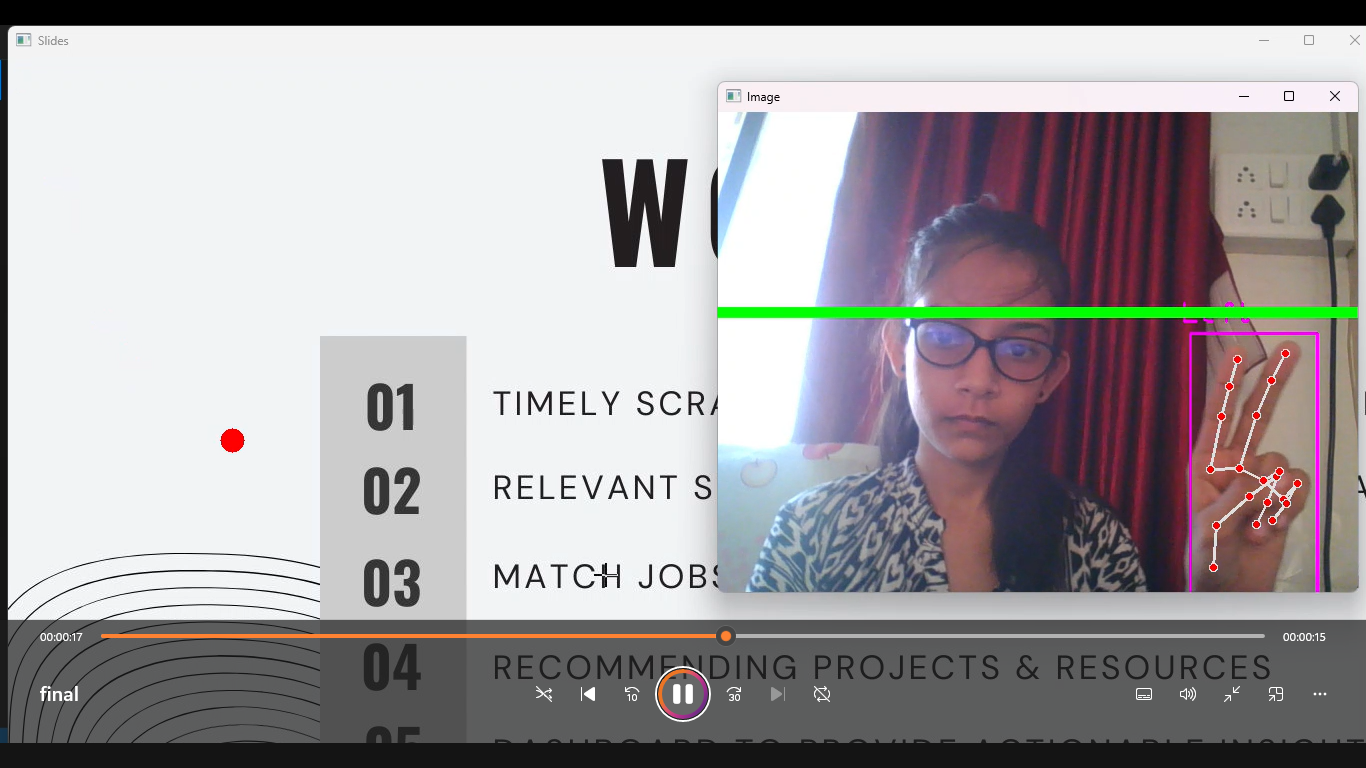
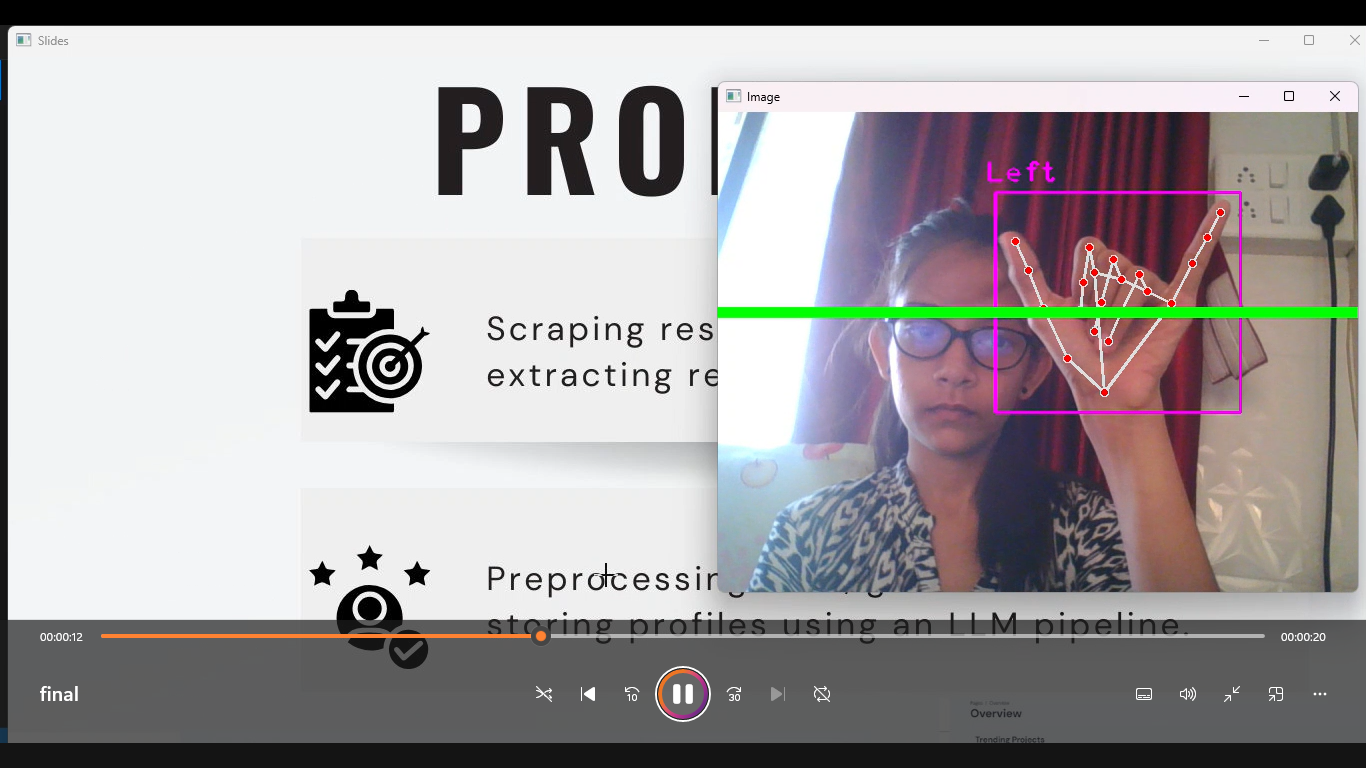
cv2.imshow("Slides", imgCurrent)

cv2.imshow("Image", img)

key = cv2.waitKey(1)

if key == ord('q'):

break

****

##### **Experiment 6**

##### 

##### **Experiment 7**

##### 

##### 

##### **Experiment 8**

##### 

##### **Experiment 9**

##### 