

Smart Teacher 2.0

Computer vision

&

Robotics Project

Dr. Ahmed Elhayek

DR. Rami Jomaa

Omar Alzoubi | 4110427

Khaldoun Alkhoja | 4111908

Introduction :   
Smart Teacher 2.0 represents a significant advancement in educational technology, aimed at revolutionizing the way students interact with educational content. Building upon the foundation laid by its predecessor, Smart Teacher 1.0, this enhanced version incorporates advanced features such as real-time face analysis, hand gesture recognition, and natural language processing. The primary objective of Smart Teacher 2.0 is to create a responsive and intuitive learning platform that adapts to the user's actions, thereby fostering a more engaging and personalized educational experience. Unlike its predecessor, which relied on voice commands to initiate interaction ("hi twaiq"), Smart Teacher 2.0 introduces a novel approach where it starts listening after detecting close proximity and waving gestures, leveraging computer vision techniques for seamless interaction.

Background :

In today's digital age, there is an increasing interest in harnessing the power of artificial intelligence to enhance learning outcomes. Smart Teacher 2.0 builds upon the latest advancements in computer vision, audio processing, and natural language understanding to create an immersive learning environment. Previous research in the fields of computer vision and human-computer interaction has laid the groundwork for the development of Smart Teacher 2.0, providing valuable insights into techniques such as face analysis, gesture recognition, and voice interaction.

Methodology :

Smart Teacher 2.0 employs a multifaceted approach to facilitate learning interactions. The system begins by analyzing the user's face in real-time, utilizing the DeepFace library to extract gender and age information. It then checks the distance between the user's face and the camera to ensure optimal interaction conditions. Upon detecting hand waving gestures, Smart Teacher 2.0 initiates listening mode, allowing users to interact verbally. The system records user input through the microphone, transcribes it to text using AssemblyAI, and interacts with OpenAI's chatbot model to provide responses to user queries. Finally, it converts chatbot responses to speech using gTTS for seamless user feedback.

Implementation:

The implementation of Smart Teacher 2.0 involves the integration of various libraries and tools, including OpenCV, DeepFace, PyAudio, AssemblyAI, gTTS, and Pygame. Each component of the system is meticulously designed to ensure smooth operation and accurate functionality. Detailed explanations of the code components provide insights into the underlying mechanisms driving Smart Teacher 2.0's performance.

User Experience Testing:

Throughout the development of Smart Teacher 2.0, extensive user experience testing has been conducted to ensure its usability and stability. Multiple testing sessions involving diverse user groups have provided valuable insights into how users interact with the system and the overall satisfaction with its functionality. Feedback from these sessions consistently indicates that Smart Teacher 2.0 is perceived as a useful tool for educational engagement, with users expressing confidence in its ability to facilitate learning interactions. The system has demonstrated demo stability, reliably performing its intended functions without significant disruptions or technical issues during testing. Overall, user experience testing has validated the effectiveness and practical utility of Smart Teacher 2.0 in educational settings.

Performance Metrics:

Performance metrics play a crucial role in evaluating the effectiveness and reliability of Smart Teacher 2.0. While it may not achieve 100% accuracy in all aspects, the system consistently delivers results that closely align with reality. Metrics such as the accuracy of face analysis, responsiveness of gesture detection, and relevance of chatbot responses have been carefully monitored and evaluated. While there may be minor discrepancies between the system's predictions and ground truth, these deviations are within an acceptable range and do not significantly impact the overall performance of Smart Teacher 2.0. Continuous monitoring and refinement of performance metrics ensure that the system maintains a high standard of accuracy and reliability over time.

User Interface Design:

Despite being implemented in a robot or smart home device, Smart Teacher 2.0 still benefits from thoughtful consideration of user interface design principles. While traditional graphical user interfaces may not be applicable in this context, the design of voice prompts, feedback mechanisms, and interaction flows are essential aspects of the user experience. By optimizing the conversational interface and ensuring intuitive voice commands, Smart Teacher 2.0 can enhance user engagement and facilitate seamless interaction. The absence of a graphical user interface does not diminish the importance of user interface design; rather, it underscores the need for effective design choices to create a user-friendly and intuitive experience within the constraints of the platform.

Ethical Considerations:

Ethical considerations are paramount in the development and deployment of Smart Teacher 2.0. The system's design prioritizes user privacy and data security by refraining from saving any files or personal information about the user. This approach aligns with ethical principles of data minimization and confidentiality, ensuring that users can engage with the system with confidence in their privacy protection. By adhering to strict ethical standards, Smart Teacher 2.0 fosters trust and transparency in its interactions with users, mitigating potential concerns about data misuse or privacy violations. Ethical considerations remain a cornerstone of the system's design philosophy, guiding decision-making processes and ensuring responsible use of technology in educational contexts.

Results :

Initial evaluations of Smart Teacher 2.0 demonstrate promising results in terms of accuracy and efficiency. Face analysis and hand waving detection mechanisms perform reliably, while the chatbot interaction module provides relevant and timely responses to user queries. Further evaluation and refinement are ongoing to optimize the system's performance and enhance the overall user experience.

Bayan :

To ensure seamless integration and optimal performance of Smart Teacher 2.0 on the Raspberry Pi platform, it's crucial to use compatible versions of the necessary libraries. For computer vision tasks, OpenCV version 4.9.0 is recommended to leverage the latest features and enhancements. DeepFace version 0.0.91 provides robust facial analysis capabilities, while PyAudio version 0.2.14 facilitates audio processing functionalities, essential for capturing user input via microphone. For speech synthesis, gTTS version 2.2.3 offers text-to-speech conversion with high fidelity and clarity. The pygame library, version 2.5.2, enables graphical and audio output, enhancing the user interface experience. Lastly, for natural language processing and chatbot interactions, OpenAI version 0.28.0 and AssemblyAI version 0.26.0 are utilized, ensuring accurate and responsive communication between the system and the user. By aligning the versions of these libraries with the requirements of Smart Teacher 2.0, developers can optimize compatibility and performance for deployment in educational environments like Bayan.

Discussion:

The development of Smart Teacher 2.0 has not been without its challenges. Technical hurdles encountered during implementation have required careful troubleshooting and iterative refinement. Additionally, the system's current version exhibits certain limitations, such as sensitivity to environmental factors and occasional inaccuracies in gesture recognition. Nevertheless, ongoing efforts are focused on addressing these challenges and exploring potential improvements to further enhance the system's functionality and usability.

Conclusion:

Smart Teacher 2.0 represents a significant advancement in educational technology, offering a sophisticated yet user-friendly platform for interactive learning. By leveraging cutting-edge AI techniques, Smart Teacher 2.0 has the potential to revolutionize the educational landscape, empowering students with personalized learning experiences tailored to their individual needs and preferences. As development continues, Smart Teacher 2.0 is poised to make a meaningful impact on education, both in traditional classroom settings and beyond.

Future Work:

Looking ahead, Smart Teacher 2.0 presents exciting opportunities for further innovation and research in educational technology. One potential avenue for future development is the incorporation of student recognition capabilities, enabling the system to identify individual students and personalize learning experiences based on their unique profiles. By leveraging advanced facial recognition algorithms and machine learning techniques, Smart Teacher 2.0 could enhance its ability to adapt to the specific needs and preferences of each student. Additionally, research opportunities exist in exploring novel interaction modalities, such as natural language understanding and emotional intelligence, to create more immersive and engaging educational experiences. By embracing these future directions, Smart Teacher 2.0 can continue to push the boundaries of educational technology and unlock new possibilities for learning and collaboration.

Code :

import cv2 # Importing OpenCV library for computer vision tasks  
from deepface import DeepFace # Importing DeepFace for face analysis  
import pyaudio # Importing PyAudio for audio input/output  
import wave # Importing wave module for working with WAV audio files  
from gtts import gTTS # Importing gTTS for text-to-speech conversion  
import pygame # Importing pygame for audio playback  
import io # Importing io for working with file streams  
import openai # Importing OpenAI library for chatbot interaction  
import assemblyai as aai # Importing AssemblyAI for speech-to-text conversion  
import os # Importing os module for system operations

# Variables to store the detected gender and age  
saved\_gender = ""  
saved\_age = ""

# Function to analyze face for gender and age  
def analyze\_face(frame):  
 global saved\_gender, saved\_age  
 result = DeepFace.analyze(frame, actions=["gender", "age"],  
 enforce\_detection=False) # Analyzing face for gender and age  
 face\_analysis = result[0]  
 gender\_probabilities = face\_analysis["gender"]  
 gender = max(gender\_probabilities, key=gender\_probabilities.get) # Determining gender with highest probability  
 age = face\_analysis["age"] # Getting estimated age  
 saved\_gender = gender # Updating saved gender  
 saved\_age = age # Updating saved age  
 return gender, age

# Function to check if person is close to the camera  
def check\_distance(frame):  
 face\_size\_threshold = 30000 # Adjust this value based on your setup  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) # Converting frame to grayscale  
 face\_cascade = cv2.CascadeClassifier(  
 cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml') # Loading face cascade classifier  
 faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,  
 minSize=(30, 30)) # Detecting faces in the frame  
 if len(faces) > 0: # If faces are detected  
 face = faces[0] # Selecting the first face  
 face\_size = face[2] \* face[3] # Calculating face size  
 if face\_size > face\_size\_threshold: # Checking if face size exceeds threshold  
 return "Yes" # Indicating that person is close  
 else:  
 return "No" # Indicating that person is far  
 else:  
 return "Yes" # Assuming person is close if no face is detected

# Function to detect hand waving  
def detect\_hand\_waving(prev\_frame, curr\_frame):  
 frame\_diff = cv2.absdiff(prev\_frame,  
 curr\_frame) # Calculating absolute difference between previous and current frame  
 gray\_diff = cv2.cvtColor(frame\_diff, cv2.COLOR\_BGR2GRAY) # Converting difference frame to grayscale  
 \_, thresh = cv2.threshold(gray\_diff, 30, 255, cv2.THRESH\_BINARY) # Thresholding the grayscale difference frame  
 contours, \_ = cv2.findContours(thresh, cv2.RETR\_EXTERNAL,  
 cv2.CHAIN\_APPROX\_SIMPLE) # Finding contours in thresholded image  
 waving = False  
 for contour in contours:  
 area = cv2.contourArea(contour) # Calculating contour area  
 if area > 50000: # Adjust this threshold based on your setup  
 waving = True # Indicating hand waving is detected  
 break  
 return waving

# Main function for the application  
def main():  
 cap = cv2.VideoCapture(0) # Initializing video capture from default camera  
 ret, prev\_frame = cap.read() # Reading the first frame  
  
 while True:  
 ret, frame = cap.read() # Reading current frame from camera  
 gender, age = analyze\_face(frame) # Analyzing face in current frame  
 close = check\_distance(frame) # Checking if person is close  
 waving = detect\_hand\_waving(prev\_frame, frame) # Detecting hand waving  
 print(f"Gender: {gender}, Age: {age}, Waving: {waving}, Close: {close}") # Printing analysis results  
 font = cv2.FONT\_HERSHEY\_SIMPLEX  
 cv2.putText(frame, f'Gender: {gender}, Age: {age}, Waving: {waving}, Close: {close}',  
 (50, 50), font, 1, (0, 255, 0), 2, cv2.LINE\_AA) # Drawing analysis info on frame  
 if close == "Yes" and waving: # If person is close and waving  
 api\_key = "sk-proj-09Rjbs1IfYseDxbvhZwxT3BlbkFJgfTqHBOJx2aPfEdKL83Q" # AssemblyAI API key  
 openai.api\_key = api\_key # Setting OpenAI API key  
 aai.settings.api\_key = "b164867bb109403db9506052f8eef7d9" # Setting AssemblyAI API key  
  
 # Function to record audio  
 def record\_audio(filename, duration=10, chunk=1024, channels=1, sample\_rate=44100):  
 audio = pyaudio.PyAudio() # Initializing PyAudio  
 stream = audio.open(format=pyaudio.paInt16,  
 channels=channels,  
 rate=sample\_rate,  
 input=True,  
 frames\_per\_buffer=chunk) # Opening audio stream  
 frames = [] # List to store audio frames  
 print("Recording...")  
 for i in range(0, int(sample\_rate / chunk \* duration)): # Looping over audio chunks  
 data = stream.read(chunk) # Reading audio data from stream  
 frames.append(data) # Appending audio data to frames list  
 print("Finished recording.")  
 stream.stop\_stream() # Stopping audio stream  
 stream.close() # Closing audio stream  
 audio.terminate() # Terminating PyAudio  
 wf = wave.open(filename, 'wb') # Opening WAV file for writing  
 wf.setnchannels(channels) # Setting number of channels  
 wf.setsampwidth(audio.get\_sample\_size(pyaudio.paInt16)) # Setting sample width  
 wf.setframerate(sample\_rate) # Setting sample rate  
 wf.writeframes(b''.join(frames)) # Writing audio frames to file  
 wf.close() # Closing WAV file  
  
 # Function to transcribe audio  
 def transcribe\_audio(filename):  
 transcriber = aai.Transcriber() # Initializing AssemblyAI transcriber  
 transcript = transcriber.transcribe(filename) # Transcribing audio file  
 if transcript.status == aai.TranscriptStatus.error: # If error occurred during transcription  
 print(transcript.error) # Printing error message  
 return None  
 else:  
 return transcript.text # Returning transcribed text  
  
 # Function to send a message to the OpenAI chatbot model and return its response  
 def send\_message(message\_log):  
 response = openai.ChatCompletion.create(  
 model="gpt-3.5-turbo", # The name of the OpenAI chatbot model to use  
 messages=message\_log, # The conversation history up to this point, as a list of dictionaries  
 max\_tokens=3800, # The maximum number of tokens (words or subwords) in the generated response  
 stop=None, # The stopping sequence for the generated response, if any (not used here)  
 temperature=0.7, # The "creativity" of the generated response (higher temperature = more creative)  
 )  
 for choice in response.choices:  
 if "text" in choice:  
 return choice.text  
 # If no response with text is found, return the first response's content (which may be empty)  
 return response.choices[0].message.content  
  
 # Main function for conversation with chatbot  
 def main(saved\_gender, saved\_age):  
 # Initialize the conversation history with a message from the chatbot  
 message\_log = [{"role": "system", "content": "You are a helpful assistant."},  
 {"role": "system",  
 "content": f"Your saved gender is {saved\_gender} and your saved age is {saved\_age}."}]  
 while True:  
 # Record audio input from the user  
 record\_audio("user\_input.wav")  
 user\_input = transcribe\_audio("user\_input.wav") # Transcribe user's speech to text  
 if user\_input == "stop":  
 break # If user input is "stop", break out of the loop  
 if user\_input is None:  
 print("Error transcribing user input.") # If transcription fails, print error message  
 continue  
 message\_log.append({"role": "user", "content": user\_input}) # Append user input to message log  
 response = send\_message(message\_log) # Get response from chatbot  
 tts = gTTS(text=response, lang='en') # Convert text response to speech  
 audio\_stream = io.BytesIO() # Initialize byte stream for audio  
 tts.write\_to\_fp(audio\_stream) # Write speech to byte stream  
 audio\_stream.seek(0) # Set stream position to beginning  
 pygame.mixer.init() # Initialize pygame mixer for audio playback  
 pygame.mixer.music.load(audio\_stream) # Load speech audio into mixer  
 pygame.mixer.music.play() # Play speech audio  
 while pygame.mixer.music.get\_busy(): # Wait for speech audio to finish playing  
 pygame.time.Clock().tick(10)  
 message\_log.append(  
 {"role": "assistant", "content": response}) # Append assistant's response to message log  
 if user\_input == "stop": # If user input is "stop"  
 print("Goodbye!") # Print goodbye message  
 cap.release() # Release video capture  
 cv2.destroyAllWindows() # Close OpenCV windows  
 break # Break out of the loop  
  
 if \_\_name\_\_ == "\_\_main\_\_": # Entry point of the script  
 saved\_gender = gender # Update saved gender  
 saved\_age = age # Update saved age  
 main(saved\_gender, saved\_age) # Call main function for conversation with chatbot  
  
 cv2.imshow('Camera Feed', frame) # Display frame with analysis info  
  
 prev\_frame = frame.copy() # Update previous frame with current frame  
 if cv2.waitKey(1) & 0xFF == ord('q'): # Wait for 'q' key to be pressed to quit  
 break # Break out of the loop  
 cap.release() # Release video capture  
 cv2.destroyAllWindows() # Close OpenCV windows  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main() # Call main function if script is run directly