WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Script to Animation: Developing a Video Generator using Python and Tkinter

Syed Khaja Osmane Haroon [1], Syed Anas [2], Mohammed Nizam Uddin Saif [3]

Department of IT, Nawab Shah Alam Khan College of Engineering and Technology , Hyderabad , India
Email: stoa4451@gmail.com.

ABSTRACT :

This research paper presents the development of a video generator application that converts script prompts into animated videos using Python and Tkinter. The application leverages the Gradio Client API and the OpenCV library to generate and display videos based on user inputs. The primary objective is to create an intuitive and interactive interface for users to transform text prompts into animations efficiently. This approach demonstrates the potential of integrating machine learning with animation technologies, opening new avenues for automated content creation.

Keywords: Video Generation, Python, Tkinter, OpenCV, Gradio Client, Animation

1. Introduction :

The advancement of artificial intelligence and machine learning has significantly impacted the multimedia and animation industry. Traditionally, creating animations involves multiple stages, including scriptwriting, storyboarding, character design, and animation. This process can be time-consuming and requires considerable skill. By automating part of this process using AI, it becomes possible to create animations more efficiently and with fewer resources. This paper explores a practical application of these technologies by developing a video generator that transforms script prompts into animated videos. The application is built using Python, Tkinter for the graphical user interface (GUI), and integrates external libraries and APIs to enhance its functionality.

2. Literature Review :

Previous research has shown the potential of AI in generating multimedia content. Various studies have focused on image synthesis, video generation, and text-to-video transformation. Deep learning models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), have been used to create realistic images and videos. Text-to-image models like DALL-E and Imagen have demonstrated impressive results in generating images from textual descriptions. However, there is limited research on text-to-animation generation. This paper builds upon existing studies by implementing a user-friendly application that automates the video creation process using text prompts. By leveraging the Gradio Client API and OpenCV, the application offers a novel approach to generating animations.

3. Methodology :

The application was developed using Python, with Tkinter as the GUI framework. The Gradio Client API is utilized to generate videos based on user inputs, and the OpenCV library handles video playback within the application. The development process involved the following steps:

3.1. Setting Up the Environment

The initial step involved setting up the development environment. This included installing necessary libraries such as Tkinter, OpenCV, Pillow, and the Gradio Client.

3.2. Designing the GUI

The graphical user interface was designed using Tkinter. The interface includes an entry box for user input, buttons for generating and resetting videos, and a display area for showing the generated video.

3.3. Integrating Gradio Client API

The Gradio Client API was integrated to handle the video generation. A function “generate_video” sends the user's text prompt to the API, which processes the prompt and returns a video.

3.4. Implementing Video Playback

OpenCV was used to manage the video playback within the Tkinter window. The “`display_video`” function reads and displays the video file returned by the Gradio Client API.

4. Illustrations :

```python
def generate_video(prompt):
    sys.stdout.reconfigure(encoding='utf-8')
    client = Client("KingNish/Instant-Video")
    base = "3d"
    motion = "guoyww/animatediff-motion-lora-zoom-in"
    step = 8
    api_name = "/generate_image"

    try:
        print(f"Input parameters - Prompt: {prompt}, Base: {base}, Motion: {motion}, Step: {step}, API Name: {api_name}")
        result = client.predict(
            prompt=prompt,
            base=base,
            motion=motion,
            step=step,
            api_name=api_name
        )

        print("Prediction result:", result)
        video_file_path = result.get('video')
        if video_file_path:
            return video_file_path
        else:
            print("No video file found in the prediction result.")
            return None

    except ValueError as e:
        print(f"An error occurred during prediction: {e}")
        return None

    except Exception as e:
        print(f"An unexpected error occurred during prediction: {e}")
        return None
```

Fig. 1 – Generating The Video

```python
def on_submit():
    global video_path, cap
    prompt = entry.get()
    if prompt:
        video_path = generate_video(prompt)
        if video_path:
            display_video(video_path)
        else:
            messagebox.showerror("Error", "Failed to generate video.")
    else:
        messagebox.showwarning("Input Error", "Prompt cannot be empty.")
```

Fig. 2 – Handling User Input

```python
def display_video(video_path):
    global cap, replay_button
    cap = cv2.VideoCapture(video_path)
    width = 640
    height = 480
    root.geometry(f"{width}x{height}")

    def update_frame():
        ret, frame = cap.read()
        if ret:
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame_resized = cv2.resize(frame_rgb, (width, height))
            image = Image.fromarray(frame_resized)
            photo = ImageTk.PhotoImage(image)
            video_label.config(image=photo)
            video_label.image = photo
            root.after(30, update_frame)
        else:
            cap.release()
            replay_button.pack(pady=10)

    def replay_video():
        global cap
        cap.release()
        cap = cv2.VideoCapture(video_path)
        replay_button.pack_forget()
        update_frame()
    if replay_button.winfo_ismapped():
        replay_button.pack_forget()
    replay_button = tk.Button(root, text="Replay Video", command=replay_video, bg="white", fg="black", font=("Comic Sans MS", 11))
    replay_button.pack_forget()
    update_frame()
```

Fig. 3 – Displaying the Video

5. Results :

The application successfully generates videos based on user inputs and displays them within the Tkinter interface. The integration of the Gradio Client API allows for seamless video generation, while OpenCV handles video playback efficiently. Users can input text prompts and receive an animated video corresponding to their prompt, showcasing the application's practicality and ease of use.

6. Requirements :

6.1. Hardware Requirements

- RAM*: 4GB.*
- CPU*: Dual-core processor.*
- Storage*: 10GB free space.*
- Network*: Stable Internet Connection.*

6.2. Software Requirements

- Operating System: Windows 7+, macOS 10.12+, or any modern Linux distribution.
- Python: Version 3.7 or later.
- Required Packages: 'tkinter', 'Pillow', 'opencv-python', 'gradio-client'.

7. Conclusion :

This paper presents a practical approach to developing a video generator application using Python. The application provides a user-friendly interface for converting script prompts into animated videos, showcasing the potential of AI and machine learning in multimedia content creation. This project illustrates how modern technologies can simplify and expedite the animation creation process, making it accessible to a broader audience.

Appendix A. Detailed Algorithm

Step 1. Import Required Libraries:

- Import tkinter, messagebox, Image, ImageTk from PIL, cv2, Client from gradio_client, and sys.

Step 2. Initialize Tkinter:

- Create a tk.Tk() instance and set its title and background color.

Step 3. Define Global Variables:

- video_path and cap for storing video file path and OpenCV capture object.

Step 4. Define Functions:

- generate_video(prompt): Use Gradio client to generate a video based on the user's input prompt.
- on_submit(): Retrieve user input, generate video, and display it if successful.
- reset(): Clear input and reset video display area.
- display_video(video_path): Play and display the generated video, allowing replay.

Step 5. GUI Layout:

- Create labels, entry for input, buttons for submission and reset, and a label for video display using pack() or grid() methods.

Step 6. Main Loop:

- Start the tkinter main loop with root.mainloop() to run the application.

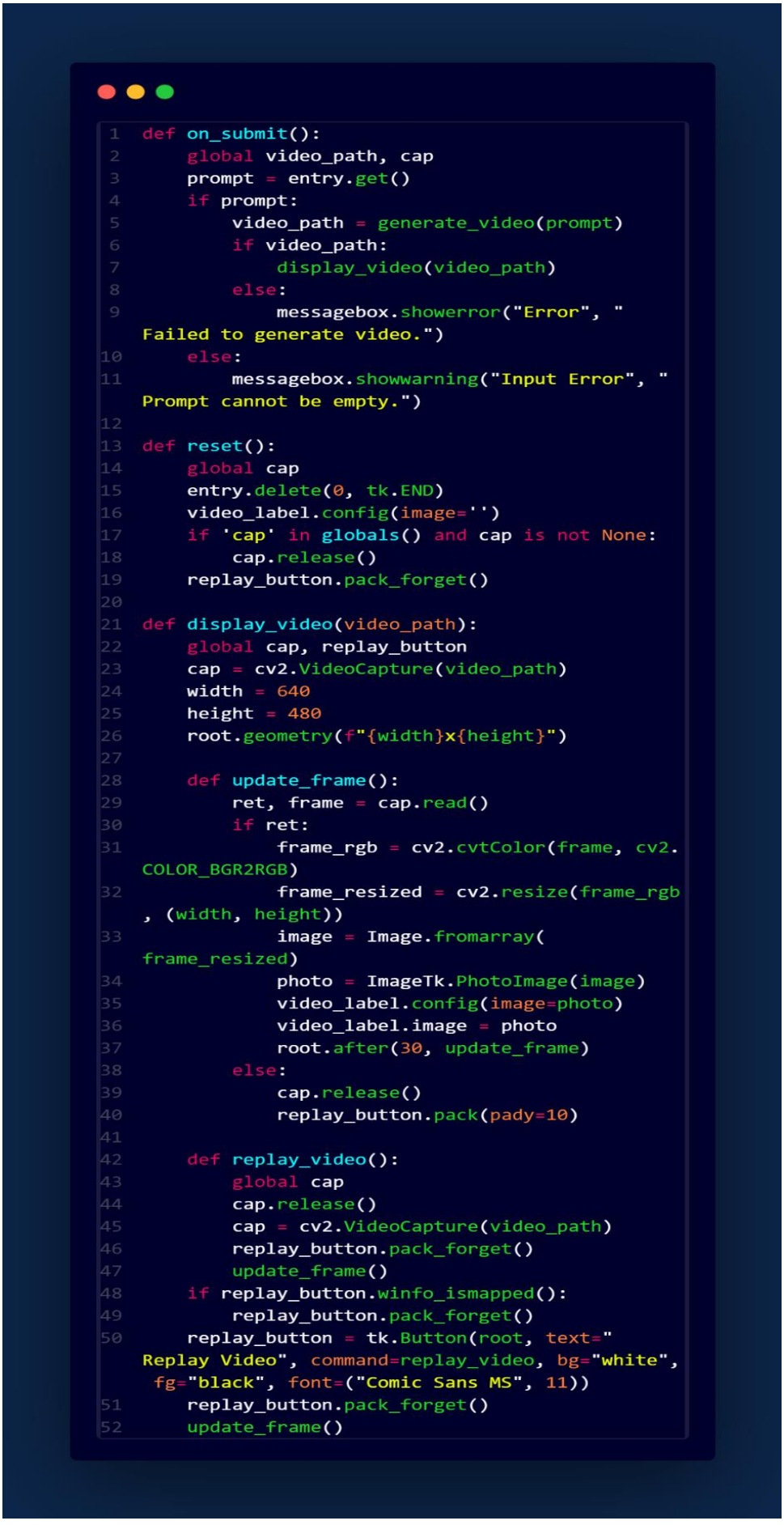
Appendix B. Survey Questionnaire

The following questionnaire was used in the study:

1. How intuitive did you find the user interface of the Video Generator application?
2. Were you satisfied with the replay functionality provided after video generation?
3. Were error messages clear and helpful when issues occurred, such as failed video generation?
4. Were you able to understand how to input a prompt and generate a video easily?

Appendix C. Full Code

```
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import cv2
from gradio_client import Client
import sys

def generate_video(prompt):
    sys.stdout.reconfigure(encoding='utf-8')
    client = Client("KingNish/Instant-Video")
    base = "3d"
    motion = "guoyww/animatediff-motion-lora-zoom-in
"
    step = 8
    api_name = "/generate_image"

    try:
        print(f"Input parameters - Prompt: {prompt}
, Base: {base}, Motion: {motion}, Step: {step}
, API Name: {api_name}")
        result = client.predict(
            prompt=prompt,
            base=base,
            motion=motion,
            step=step,
            api_name=api_name
        )

        print("Prediction result:", result)
        video_file_path = result.get('video')
        if video_file_path:
            return video_file_path
        else:
            print("
No video file found in the prediction result.")
            return None

    except ValueError as e:
        print(f
"An error occurred during prediction: {e}")
        return None

    except Exception as e:
        print(f
"An unexpected error occurred during prediction: {e}
")
        return None
```



```
def on_submit():
    global video_path, cap
    prompt = entry.get()
    if prompt:
        video_path = generate_video(prompt)
        if video_path:
            display_video(video_path)
        else:
            messagebox.showerror("Error", "
Failed to generate video.")
    else:
        messagebox.showwarning("Input Error", "
Prompt cannot be empty.")

def reset():
    global cap
    entry.delete(0, tk.END)
    video_label.config(image='')
    if 'cap' in globals() and cap is not None:
        cap.release()
    replay_button.pack_forget()

def display_video(video_path):
    global cap, replay_button
    cap = cv2.VideoCapture(video_path)
    width = 640
    height = 480
    root.geometry(f"{width}x{height}")

    def update_frame():
        ret, frame = cap.read()
        if ret:
            frame_rgb = cv2.cvtColor(frame, cv2.
COLOR_BGR2RGB)
            frame_resized = cv2.resize(frame_rgb
, (width, height))
            image = Image.fromarray(
frame_resized)
            photo = ImageTk.PhotoImage(image)
            video_label.config(image=photo)
            video_label.image = photo
            root.after(30, update_frame)
        else:
            cap.release()
            replay_button.pack(pady=10)

    def replay_video():
        global cap
        cap.release()
        cap = cv2.VideoCapture(video_path)
        replay_button.pack_forget()
        update_frame()
    if replay_button.winfo_ismapped():
        replay_button.pack_forget()
    replay_button = tk.Button(root, text="
Replay Video", command=replay_video, bg="white",
 fg="black", font=("Comic Sans MS", 11))
    replay_button.pack_forget()
    update_frame()
```

```
root = tk.Tk()
root.title("Video Generator")

width = 640
height = 480
root.geometry(f"{width}x{height}")


root.configure(bg="lightblue")

label = tk.Label(root, text="Enter your prompt:", bg
="lightblue", fg="black", font=("Comic Sans MS", 16
))
label.pack(pady=10)

entry = tk.Entry(root, width=50, bg="white", fg="
black", font=("Arial", 12))
entry.pack(pady=10)

button_frame = tk.Frame(root, bg="lightblue")
button_frame.pack(pady=10)

submit_button = tk.Button(button_frame, text="
Generate Video", command=on_submit, bg="white", fg="
black", font=("Comic Sans MS", 11))
submit_button.pack(side=tk.LEFT, padx=5)

reset_button = tk.Button(button_frame, text="Reset",
 command=reset, bg="white", fg="black", font=("
Comic Sans MS", 11))
reset_button.pack(side=tk.LEFT, padx=5)

video_label = tk.Label(root, width=width, height=
height, bg="lightblue")
video_label.pack()

replay_button = tk.Button(root, text="Replay Video")

root.mainloop()
```

REFERENCES :

1. Van der Geer, J., Hanraads, J. A. J., & Lupton, R. A. (2000). The art of writing a scientific article.*Journal of Science Communication, 163*, 51–59.
2. Strunk, W., Jr., & White, E. B. (1979).*The elements of style* (3rd ed.). New York: MacMillan.
3. Mettam, G. R., & Adams, L. B. (1999).How to prepare an electronic version of your article. In B. S. Jones & R. Z. Smith (Eds.), *Introduction to the electronic age* (pp. 281–304). New York: E-Publishing Inc.
4. Fachinger, J., den Exter, M., Grambow, B., Holgerson, S., Landesmann, C., Titov, M., et al. (2004).Behavior of spent HTR fuel elements in aquatic phases of repository host rock formations, 2nd International Topical Meeting on High Temperature Reactor Technology. Beijing, China, paper #B08.
5. Fachinger, J. (2006). Behavior of HTR fuel elements in aquatic phases of repository host rock formations.*Nuclear Engineering & Design,236*, 54.