```
import cv2
import mediapipe as mp
from math import hypot
import screen_brightness_control as sbc
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
import numpy as np
import tkinter as tk
from PIL import Image, ImageTk

# Global variables
last_hand_detected_time = 0
no_hand_message_displayed = False

def update_controls():
    global cap, last_hand_detected_time, no_hand_message_displayed

    # Read frame from the webcam
    _, frame = cap.read()
    frame = cv2.flip(frame, 1)
    frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Process hand landmarks
    Process = hands.process(frameRGB)

    landmarkList = []
    handSide = []

    if Process.multi_hand_landmarks:
        # Reset the timer if hand is detected
        last_hand_detected_time = 0
        no_hand_message_displayed = False

        for handlm in Process.multi_hand_landmarks:
            for _id, landmarks in enumerate(handlm.landmark):
                height, width, color_channels = frame.shape
                x, y = int(landmarks.x * width), int(landmarks.y * height)
                landmarkList.append([_id, x, y])

            thumb_x = landmarkList[4][1]
            if thumb_x < width / 2:
                handSide.append("Left")
            else:
                handSide.append("Right")

    # If no hand is detected
    else:
        current_time = cv2.getTickCount() / cv2.getTickFrequency()
        if current_time - last_hand_detected_time > 10 and not no_hand_message_displayed:
            volume_label.config(text="No hand detected!")
            brightness_label.config(text="No hand detected!")
            no_hand_message_displayed = True
```

```
    if landmarkList:
        for hand, landmarks in zip(handSide, Process.multi_hand_landmarks):
            x_1, y_1 = landmarks.landmark[4].x * width, landmarks.landmark[4].y * height
            x_2, y_2 = landmarks.landmark[8].x * width, landmarks.landmark[8].y * height

            L = hypot(x_2 - x_1, y_2 - y_1)

            if hand == "Right":
                vol = np.interp(L, [50, 220], [minVol, maxVol])
                volume.SetMasterVolumeLevel(vol, None)
                volBar = np.interp(L, [50, 220], [400, 150])
                volPer = np.interp(L, [50, 220], [0, 100])
                volume_label.config(text=f'Volume: {int(volPer)}%')

            elif hand == "Left":
                b_level = np.interp(L, [15, 220], [0, 100])
                sbc.set_brightness(int(b_level))
                brightness_label.config(text=f'Brightness: {int(b_level)}%')


    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    img = Image.fromarray(frame)
    img = ImageTk.PhotoImage(img)
    video_label.img = img
    video_label.configure(image=img)
    video_label.after(10, update_controls)  # Update every 10 milliseconds

# Create the main window
root = tk.Tk()
root.title("Hand Gesture Control")
root.geometry("800x600")
root.configure(bg="#8B8589")  # Background color

# Webcam setup
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

# Volume Control Setup
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
volRange = volume.GetVolumeRange()
minVol, maxVol, volBar, volPer = volRange[0], volRange[1], 400, 0

# Initialize the Hand Tracking Model
mpHands = mp.solutions.hands
hands = mpHands.Hands(
    static_image_mode=False,
    model_complexity=1,
    min_detection_confidence=0.75,
    min_tracking_confidence=0.75,
    max_num_hands=2
```

```
)
Draw = mp.solutions.drawing_utils

# Labels for volume and brightness
volume_label = tk.Label(root, text="Volume: 0%", font=("Helvetica", 14), bg="#2C3E50", fg="white")
volume_label.pack(pady=10)

brightness_label = tk.Label(root, text="Brightness: 0%", font=("Helvetica", 14), bg="#2C3E50", fg="white")
brightness_label.pack(pady=10)

# Label to display webcam feed
video_label = tk.Label(root, bg="#2C3E50")
video_label.pack()

# Start updating controls
update_controls()

# Run the main loop
root.mainloop()

# Release the webcam when the window is closed
cap.release()
cv2.destroyAllWindows()
```