Background Video Preparation Module (background_video_prep.py) Error Log

This module handles basic video operations, like resizing and speed adjustment.

1. Loading Image Crosshair into the Video

Problem Statement
I want to overlay an image (especially a crosshair image) onto a video. The overlay should be transparent, and the image should be at the video's center.

Initial Issues
1. Broadcast Error: The initial code tried to overlay the image onto the video, leading to a broadcast error. This was due to a shape mismatch when trying to perform element-wise operations between the overlay layer and the region of interest (ROI) in the background image.
2. Size Mismatch: The overlay image's size was larger than the video frame's size, causing it to go out of bounds when trying to place it in the center.

Debugging Steps
1. Check Image and Video Sizes: I first determined the sizes of the overlay image and video frame to understand the mismatch.
2. Print Statements: I added debugging print statements to check the calculated overlay start and end coordinates, ensuring they were within the video frame's size.
3. Visual Inspection: I loaded the image to visually inspect its properties, especially its transparency.

Implemented Solutions
1. Resize Overlay Layer: The overlay image was resized to fit the video frame. This directly addressed the size mismatch issue.
2. Adjust Start Index: The calculation of the start index was adjusted to ensure they never become negative, and the overlay image's size was adjusted accordingly. This ensured the overlay never went out of bounds.
3. Alpha Blending: Instead of placing the image directly onto the video frame, I used alpha blending. This involved using the alpha channel (transparency channel) of the overlay image to determine how the overlay and background should mix.

Final Solution
I wrote a function "overlay_centered," where:
1. The overlay image's size is adjusted to a specific dimension (e.g., 300x300).
2. The center of the background video frame is calculated.
3. The overlay start and end positions are determined based on the resized overlay image.
4. The image is overlaid onto the calculated position on the video frame using alpha blending.
This solution successfully placed the overlay image at the video's center, ensuring it stayed within the video frame's bounds and maintained the overlay's transparency.

Conclusion
Overlaying an image onto a video requires careful handling of size and position, especially when the image has transparency. By understanding the properties of the image and video, and using methods like alpha blending, I achieved precise overlay effects. Debugging steps, like using print statements and visually inspecting the image, were helpful in identifying and resolving issues.

2. Black Detection, Color Replacement

The problem might not be the cross being too thin, but how to detect and replace black pixels. The method was to check for pure black pixels ([0, 0, 0]) and replace them with orange-red. If the cross isn't pure black (due to anti-aliasing or other reasons), the pixels might not be detected as black and therefore not replaced.
I modified the method to be more lenient in detecting black pixels:
Convert the image to grayscale.
Use a threshold to detect dark pixels, considering them as "black" pixels.
Replace those detected pixels with orange-red.

3. North Challenge + Determining Left or Right

The issue of discontinuity between 0° (North) and 360° (also North) is a well-known challenge in handling angles. For instance, a jump from 350° to 10° should actually be seen as a 20° difference.
To solve this, I used a technique called "shortest angle difference." It works like this:
Given two angles (a) and (b), the difference (d) is:

d = arctan2(sin(b-a), cos(b-a))

This formula gives the shortest difference between two angles, considering the 360°/0° wrap-around. The result is within the \(-180°\) to \(180°\) range.
For example, if my target angle is 10° and the detected angle is 350°, the formula will correctly give a -20° difference (the same as a 20° magnitude difference).
Using this formula, I correctly handle transitions around North and ensure my calculations for determining if the user is facing the correct direction within a threshold are accurate.

1. If the difference is “positive”, the user is to the “left” of the expected direction.
2. If the difference is “negative”, the user is to the “right” of the expected direction.
3. If the difference is “zero”, the user is exactly facing the expected direction

For example, if the target direction is North (0°), and the user is facing 10° (North-East), then the difference will be positive, indicating the user is to the left of North. Conversely, if the user faces 350° (North-West), then the difference will be negative, indicating the user is to the right of North.

4. Day/Night Threshold

The threshold for determining day or night based on image brightness can vary depending on context and specific application. However, if looking for a common starting point, the average brightness of an image is usually calculated in the 0-255 range (for 8-bit grayscale images).

For a simple day/night discriminator:

1. 128 is the midpoint value. Any time higher than this can be considered day, and any time lower can be considered night. However, considering indoor scenes during the day might fall below this threshold, this could be a very naive threshold.
2. More conservative values, like 100, are often used in practice to account for the above situation.
3. In some applications, a value between 90-110 might be chosen based on empirical testing.

In controlled environments, like specific camera setups and consistent shooting conditions, the best threshold can be empirically determined by analyzing sample clips. But for varying conditions, no single value is perfect.

For academic rigor, researchers might derive this value based on a training dataset. But for general projects, considering the 0-255 grayscale range, around 100 is a reasonable starting point.

5. Main Program Compiled with No Output, No Processor, Switched to xvido.

6. First Frame Not Showing Correct Results, Turned Gray

In my code, I used the following logic to decide whether to apply the grayscale effect:
grayscale_effect = is_default_orientation
This means the grayscale effect is applied only when the orientation matches the default orientation.