* Introduction

As technology has evolved throughout human history, so has advertisement and entertainment. A long time ago, we did not have the technology that we have today to provide both advertisement and entertainment. But now, we have the technology to record videos and change them upon the completion of the recording. In the middle of said video, the editor may fade in an image or cut to an image to produce some sort of comedic effect or to advertise a sponsor. But what if such changes could be made while recording a live video? What if we could see the edits we make as we are recording? We decided to explore this concept by using a Raspberry Pi to create and run Python code that runs two copies of a single live video. One video would be the original stream, and the other would have color changes we make to the color values collected from the original video.

* Problem Characterization

For our project, we recorded the ten most common colors from an original real-time video and made changes to them to customize the coloring of said video. A camera was used to take a live video and showed two copies on a desktop. One video showed the original live video with no changes made while the other showed the same video with the color changes. From the first frame of the video, the ten most common colors were recorded and shown on ten buttons and input fields. To make the color changes, the input fields containing the default color values would be changed to a value representing a different color. Once the button with the new input field was clicked, the color changes to one of the live video copies was made and displayed on the desktop. But it was also possible that the setting in which the camera recorded its live video could have changed dramatically, so we needed some way to address this to make it convenient for the customer. (Can Reilly Fix This Up?)

* Proposed Solution and Implementation Strategy

The first step to creating our project was to implement a user interface. We used the python GUI library called tkinter, which, according to the website TutorialsPoint, “provides a fast and easy way to create GUI application.” Michael Coffey was responsible for much of the UI while Hayden implemented much of the functionality. Michael’s original plan for the UI was to have a header frame with a “refresh” button at the top. Below that, there would be two side by side frames that store copies of the live video, with the right showing the actual color changes. At the bottom, there would be the color palette with input fields and buttons where the colors would change upon entering the information and clicking the button below.

As mentioned before, the task for this project is to allow for color elements in a live video to be changed to that which the user desires. Although Hayden was responsible for much of the functionality, Michael decided to collect the ten most common colors from the live video. However, it proved to be quite difficult to imagine how this is done. Hayden suggested to Michael that he should find a way to collects the most common color data according to the color values of each pixel displaying the video. Following this advice, Michael changed his search strategy.

Soon, he found an article written by M. Rake Linggar L. on the website “Towards Data Science”. Lingger presented three methods for collecting the most common colors, and the chosen method was the one that used K-Means clustering. After learning about K-Means clustering, Michael downloaded the necessary modules necessary for the algorithm as shown by Linggar: sklearn and collections. Michael then adapted the code provided in the article for the purpose of collecting 10 colors from a single video frame as opposed to collecting 5 from an image. His plan was for Hayden to load the collected colors from the first video frame into the color palette shown near the bottom of the window. In order to test his code, he printed out the cluster of RGB values, which, as Linggar showed in an image, was represented as an array with up to ten integer arrays with three integers in each one. The returned integer arrays each represented the R, G, and B values of each color respectively. Once this was implemented, the next step was to apply the colors to the palette and apply the functionality of changing the colors of a live video feed.

Before the development of the functionality could begin, a second video frame had to be put on display so that one can be shown without any changes and one with the changes. Ultimately, all it took was a second copy of the RealTime video object that ran on the exact same code. Aside from this, it was soon clear to Michael that because the video captured frames on a while loop basis, the program would slow down if the color palette captured colors for every single frame. So he had the idea of only capturing the colors of the first frame and presenting a refresh button in the header to recapture the colors of the latest frame. Although this would delay the start of the live feed, it would not slow the program down once a video in motion was shown.

After finalizing the design, Hayden set to work on the…..(For Hayden to write up)

At first, the UI was built using several frames that call the method “pack()” to display the interface. It was a great starting point for visualizing the implementation of the functionality and several Widgets, but after much of it was complete, it was clear some major changes were necessary. In order to clean up the UI, Michael then researched a way to make the default size of the window be just below the width and height of the window while also making it so that its size cannot be adjusted for consistent video and palette visibility. From the website TutorialsPoint, he found out how to get the dimension of the screen size. From the website Plus2Net, he learned about the tkinter method “grid()” and its greater flexibility over using “pack().” And from the website TutorialsPoint, he learned how to disable the resizing of a tkinter window. After learning what he needed for his idea, Michael changed the calls to the method “pack()” for every frame on the window to a call to “grid(),” which allows for a cleaner and more consistent UI. Michael was able to remove both a master frame and another that used to contain one of the streams to reduce the amount of necessary storage. Next, he set the dimensions of the tkinter window to be slightly smaller than those of the screen on which the code runs so that it is compatible with any screen. Lastly, he disabled the resizability of the window so that the UI remains consistent, no matter what screen is used for display.

Discuss all of the modules implemented (in separate paragraphs for each one)

* Conclusions

Restating the problem, giving a summary of the problem and solution, describe how, with further evolution, this could be used to create more editing opportunities for content creation. (Maybe Reily could help)

**Works Cited**

A., M. Rake Linggar. “Finding Most Common Colors in Python.” *Medium*, Towards Data Science, 9 Oct. 2020, <https://towardsdatascience.com/finding-most-common-colors-in-python-47ea0767a06a>.

“How Can I Prevent a Window from Being Resized with Tkinter.” *Tutorials Point*, <https://www.tutorialspoint.com/how-can-i-prevent-a-window-from-being-resized-with-tkinter>.

“How to Get the Screen Size in Tkinter.” *Tutorials Point*, <https://www.tutorialspoint.com/how-to-get-the-screen-size-in-tkinter#:~:text=In%20order%20to%20get%20the,of%20the%20screen%20in%20pixels>.

“Python - GUI Programming (TKINTER).” *Tutorials Point*, <https://www.tutorialspoint.com/python/python_gui_programming.htm>.

This article is written by plus2net.com team.https://www.plus2net.com. “Tkinter Rowconfigure & Columnconfigure to Assign Relative Weight to Rows and Columns Width & Height.” *Python Tkinter Rowconfigure & Columnconfigure Using Weight*, plus2net.Com, 5 Feb. 2019, <https://www.plus2net.com/python/tkinter-rowconfigure.php>.