In [1]:
```
import cv2 as cv
import os
import numpy as np
```


- Srihari Bandarupalli
- 2021112006

Video to Images

Problem 1

The assignment involved creating a program that accomplishes two tasks: splitting a video into individual frames and storing them in a specified folder, and conversely, compiling a collection of images from a folder into a single video with an adjustable frame rate.

Task 1 - Splitting Video into Frames

Implementation:

- The function takes two arguments: `video_path` (the path to the source video) and `output_folder` (the path to the folder where extracted frames will be saved).
- It utilizes OpenCV's `VideoCapture` to open and process the video file.
- The function ensures the existence of the destination folder using `os.makedirs` if it's not already present.
- Frames are extracted from the video iteratively using `cap.read()`, and each frame is saved as an image file (e.g., "frame_0001.png", "frame_0002.png", etc.).
- The process continues until all frames are extracted, and then the video capture object (`cap`) is closed.

In [2]:
```
def video_to_images(video_path, output_folder):

    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    cap = cv.VideoCapture("./sample.mp4")

    count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        image_path = os.path.join(output_folder, f"frame_{count:04d}.jpg"
        cv.imwrite(image_path, frame)
        count += 1

    cap.release()
```

In [3]:
```
video_to_images('input_video.mp4', 'output_images_folder')
```

Images to Video

Task 2 - Compiling Images into a Video

Implementation:

- The function takes three parameters: `input_folder` (the folder containing the images), `output_video` (the path for the output video), and `frame_rate` (desired frame rate for the video, default is 30).
- It retrieves and sorts image filenames from the specified folder.
- The function reads the first image to determine video dimensions.
- A video file is created using OpenCV's `VideoWriter`, with specified dimensions and frame rate.
- Each image is read and sequentially added to the video.
- The video writer object is released after processing all images.

Testing and Challenges:

Testing involved using a video from YouTube, examining compatibility with various resolutions, and adjusting the frame rate.

Outcomes:

- Output video in img_to_video.mp4

In [4]:
```
def images_to_video(input_folder, output_video, frame_rate=30):

    images = [img for img in os.listdir(input_folder)]
    images.sort()

    img = cv.imread(os.path.join(input_folder, images[0]))
    height, width, layers = img.shape

    # Create video writer object
    video = cv.VideoWriter(output_video, cv.VideoWriter_fourcc(*'mp4v'),

    # Write images to video
    for image in images:
        img_path = os.path.join(input_folder, image)
        frame = cv.imread(img_path)
        video.write(frame)

    # Release the video writer object
    video.release()
```

In [5]:
```
images_to_video('output_images_folder', 'img_to_video.mp4', frame_rate=30
```

Capturing Images

Problem 2

Creating a function for capturing and storing frames from a webcam while simultaneously displaying the video feed.

Solution:

1. It initializes a webcam session, captures the specified number of frames, displays them, and saves each as a JPEG file.
2. A try-finally block ensures the proper release of webcam resources and closes display windows in case of interruptions.

Experiments and Challenges:

Challenges include handling varying default webcam indices and ensuring uninterrupted frame display.

Results:

An example of a captured frame:

In [6]:
```
%%capture
output_folder ="./webcam"
if not os.path.exists(output_folder):
        os.makedirs(output_folder)

cap = cv.VideoCapture(0)

count = 0
while True:
```

```
    ret, frame = cap.read()

    # if display:
    cv.imshow('Webcam', frame)

    image_path = os.path.join(output_folder, f"frame_{count:04d}.jpg")
    cv.imwrite(image_path, frame)
    count += 1

    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv.destroyAllWindows()
```

```
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)

QObject::moveToThread: Current thread (0x263e900) is not the object's thre
ad (0x357c8a0).
Cannot move to target thread (0x263e900)
```

Problem 3

The objective was to create a composite video using chroma keying, a process involving the replacement of a specific color (typically green or blue) in one video with scenes from another.

Solution:

For this task, two videos were utilized: one featuring a green screen background (foreground) and another serving as the background. Chroma keying techniques were employed, emphasizing precise color replacement and parameter adjustment for a seamless integration.

Procedure:

1. Obtain two distinct videos: one as the foreground with a green screen background and another as the background layer.
2. Implement chroma keying techniques, focusing on accurate color substitution and fine-tuning parameters for a natural blend of the two layers.

Observations and Challenges:

The critical aspect involved accurately replacing the green screen in the foreground video with the background video while ensuring a natural and seamless appearance.

Findings:

This approach successfully merged the two videos, creating a convincing composite that effectively blended elements from both sources.

In [7]:
```
def chroma_keying(green_screen_video_path, background_video_path, output_
    cap_foreground = cv.VideoCapture(green_screen_video_path)
    cap_background = cv.VideoCapture(background_video_path)

    # Get the frames' dimensions and frame rate from the green screen vid
    width = int(cap_foreground.get(3))
    height = int(cap_foreground.get(4))
    frame_rate = cap_foreground.get(5)  # Get the frame rate

    # Create a VideoWriter object to save the output with the same frame
    fourcc = cv.VideoWriter_fourcc(*'mp4v')
    out = cv.VideoWriter(output_video_path, fourcc, frame_rate, (width, h

    while True:
        # Read frames from both videos
        ret_foreground, frame_foreground = cap_foreground.read()
        ret_background, frame_background = cap_background.read()

        if not ret_foreground or not ret_background:
            break  # Break the loop if any of the videos end

        # Convert the frames to HSV color space
        hsv = cv.cvtColor(frame_foreground, cv.COLOR_BGR2HSV)
```

```
        # Define the range of green color in HSV
        lower_green = np.array([40, 50, 50])
        upper_green = np.array([80, 255, 255])

        # Create a mask for the green color
        mask = cv.inRange(hsv, lower_green, upper_green)

        # Invert the mask to get the non-green regions
        mask_inv = cv.bitwise_not(mask)

        # Extract the foreground using the mask
        foreground = cv.bitwise_and(frame_foreground, frame_foreground, m

        # Extract the background using the mask
        background = cv.bitwise_and(frame_background, frame_background, m

        # Combine the foreground and background
        result = cv.add(foreground, background)

        # Write the result to the output video
        out.write(result)

        # Display the result (optional, for visualization)
        cv.imshow('Chroma Keying', result)
        if cv.waitKey(1) & 0xFF == 27:  # Press 'Esc' to exit
            break

    # Release resources
    cap_foreground.release()
    cap_background.release()
    out.release()
    cv.destroyAllWindows()
```

In [8]:
```
chroma_keying('gs.mp4', 'sample.mp4', 'output_video.mp4')
```