The following code is an implementation of a real-time video cartoonizer using OpenCV (cv2) and NumPy.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import os
```

In this section, the necessary libraries are imported. `cv2` (OpenCV) is used for image and video processing, `numpy` is used for numerical computations, `matplotlib.pyplot` is used for plotting, and `os` is used for operating system-related functionalities.

```
def edge_mask(img, line_size, blur_value):
    gray = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
    gray_blur = cv.medianBlur(gray, blur_value)
    edges = cv.adaptiveThreshold(gray_blur, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, line_size, blur_value)
    return edges
```

The `edge_mask` function takes an image `img`, line size, and blur value as inputs. It converts the image to grayscale using `cv.cvtColor`, applies median blurring using `cv.medianBlur`, and then applies adaptive thresholding using `cv.adaptiveThreshold` to obtain the edges of the image. The resulting edges are returned.

```
def color_quantization(img, k):
    data = np.float32(img).reshape((-1, 3))
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 20, 0.001)
    ret, label, center = cv.kmeans(data, k, None, criteria, 10, cv.KMEANS_RANDOM_CENTERS)
    center = np.uint8(center)
    result = center[label.flatten()]
```

```
    result = result.reshape(img.shape)
    return result
```

The `color_quantization` function performs color quantization on the input image `img` using the k-means clustering algorithm. It converts the image to a float32 array, reshapes it to have a single row of pixels, and applies k-means clustering using `cv.kmeans`. The resulting color centers are converted back to uint8 format, and the pixels of the input image are replaced with the corresponding color centers. The quantized image is returned.

```
def cartoon(blurred):
    c = cv.bitwise_and(blurred, blurred, mask=edges)
    return c
```

The `cartoon` function takes the blurred image and applies a bitwise AND operation using the `edges` mask. This helps to preserve the edges from the original image while maintaining the cartoon-like appearance. The resulting image is returned.

```
vid = cv.VideoCapture(0)
```

Here, a `VideoCapture` object is created to capture video frames from the default camera (index 0). If you want to use a different video file, you can provide the file path instead of `0`.

```
while True:
    ret, frame = vid.read()
    scale_percent = 175  # percent of original size
    width = int(frame.shape[1] * scale_percent / 100)
    height = int(frame.shape[0] * scale_percent / 100)
    dim = (width, height)
    frame_resized = cv.resize(frame, dim, interpolation=cv.INTER_AREA)
    img = cv.cvtColor(frame_resized, cv.COLOR_BGR2RGB)
```

```
    line_size, blur_value = 9, 5
    edges = edge_mask(img, line_size, blur_value)
    img = color_quantization(img, k=7)
    blurred = cv.bilateralFilter(img, d=7, sigmaColor=200, sigmaSpace=200)



    cv.imshow('frame', cartoon(blurred))
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
```

This section contains the main loop where each video frame is processed. Inside the loop, the next frame is read using `vid.read()`. The frame is then resized based on a scale percentage provided (175% in this case) using `cv.resize`. The resized frame is converted from BGR to RGB color space using `cv.cvtColor`.

The `edge_mask` and `color_quantization` functions are applied to obtain the edges and perform color quantization on the resized frame.

After that, a bilateral filter is applied to the quantized image using `cv.bilateralFilter`. The bilateral filter helps to smooth the image while preserving the edges.

Finally, the `cartoon` function is called to combine the blurred image with the edges mask, and the resulting image is displayed using `cv.imshow`.

The loop continues until the 'q' key is pressed, at which point the loop breaks, and the video capture is released using `vid.release()`. The windows created by `cv.imshow` are closed using `cv.destroyAllWindows()`.

This code essentially captures video frames, applies edge detection, color quantization, and bilateral filtering to each frame, and displays the cartoonized video in real-time.