MoviePy

What is **MoviePy**?

"**MoviePy** is Python module for video editing, which can be used for basic operations (like cuts, concatenations, title insertions)… video processing, or to create advanced effects. It can read and write the most common video formats."

Source: https://zulko.github.io/moviepy/index.html

MoviePy cannot read from a webcam but for demo purposes, this library will help us process the video from dash cameras and apply the process_pipeline () method to apply whole lane detection pipeline to an input color frame.

```
131  if __name__ == '__main__':
132
133      # first things first: calibrate the camera
134      ret, mtx, dist, rvecs, tvecs = calibrate_camera(calib_images_dir='camera_cal')
135
136      mode = 'images'
137
138      if mode == 'video':
139
140          selector = 'project'
141          clip = VideoFileClip('{}_video.mp4'.format(selector)).fl_image(process_pipeline)
142          clip.write_videofile('out_{}_{}.mp4'.format(selector, time_window), audio=False)
```

The main advantage of this is that we can test our algorithms without concerning about processing speed at the moment. As live video processing will increase the computing demand and the cost to process will be more expensive, it's good to test our assumptions with a low cost solution.

"The actual processing pipeline is implemented in function process_pipeline () in main.py. As it can be seen, when a detection of lane-lines is available for a previous frame, new lane-lines are searched through line_utils.get_fits_by_previous_fits (): otherwise, the more expensive sliding windows search is performed."

Source: https://github.com/ndrplz/self-driving-car/tree/master/project_4_advanced_lane_finding

```
111      # fit 2-degree polynomial curve onto lane lines found
112      if processed_frames > 0 and keep_state and line_lt.detected and line_rt.detected:
113          line_lt, line_rt, img_fit = get_fits_by_previous_fits(img_birdeye, line_lt, line_rt, verbose=False)
114      else:
115          line_lt, line_rt, img_fit = get_fits_by_sliding_windows(img_birdeye, line_lt, line_rt, n_windows=9, verbose=False)
```

Below a demo how the invert_green_blue () method is linked using fl_image.

You can also modify the display of a clip with `clip.fl_image`. The following takes a clip and inverts the green and blue channels of the frames:

```
def invert_green_blue(image):
    return image[:,:,[0,2,1]]

modifiedClip = my_clip.fl_image( invert_green_blue )
```

Similarly we link to the process_pipeline () method:

```
140          selector = 'project'
141          clip = VideoFileClip('{}_video.mp4'.format(selector)).fl_image(process_pipeline)
```

Installation

```
conda install -c conda-forge moviepy
```