We started by creating a class " LaneDetector " which contains the following attributes:

- **Frame_processed**: a counter that indicates the number of frames processed to the assessed point of time. This information allowed us to control the sampling of new lines in consecutive frames.
- **Prev_line_coords**: a list of tuples which 2 lines (lane separators). Initialized to the starting point of the video
- **right_lane_anchor_TH**: Used to determine whether a lane change is taking place. If the right lane separator is out of the range we know a lane change is occurring.
- **Message_frame_counter**: helps limit the amount of frames in which the lane changing message shows.

The class implements the following methods:

- **drawLane**: every 11 frames calculate 2 lines, than using an overlay it prints the colored lines and a colored plane between them
- **createMask**: create a trapeze mask which indicates the point of interest in the frame (dashcam point of view morphs the angle of a lane separator) or a right angled trapeze (left or right) if a lane change has started.
- **laneSwitchDetection**: using the *right_lane_anchor_range* it determines if a lane change takes place by inspect the distance of the current right lane lower x value from a given threshold.
- **yellowAndwhiteMask**: creating a mask containing only the white and yellow object in a frame using HSV domain
- **chooseTwoLines**: it sorts a given lines list (if exists) to groups representing line-forming candidate by calculating each line's angle. It calculates a corresponding segment by taking the median of each coordinates group to later plot on the frame.
- **createSafeCoords**: supply a reasonable coordinate (fix X values outliers)
- **preprocess**: it converts a frame to HSV domain, create a trapeze mask based on the frame, isolates the yellow and white objects in the cropped frame. Using gaussian blur to smooth the frame and then using Canny's edge detection algorithm.
- **detect**: using Hough line algorithm, it extract considerable lines from the frame. Then it uses *drawLane()* find and plot the most suitable lines on the frame. Finishing with a call to *laneSwitchDetection().*

The Algorithm strong points:

- shadowed segments of the frame are handled successfully.

Frame that contain significant shadowed segments are difficult to process since the pixels intensities scale down dramatically. The general assumptions made on a lighted frame are irrelevant.

Our pre-process pipeline will not recognize any lines on a shadowed frame (i.e. 3rd example). So to overcome this, our heuristic was to choose the previous lines that fit most, as we assume no significant change is made under less then a second which is a reasonable timeframe between shadowed frame in our video.

- Nearby passing cars do not interfere with the lane detection process.

Since the program uses HoughlinesP which is affected by the pixels in the frame, that's why a car inside the frame will create redundant noise, pixel-wise.

To overcome this, our program generates 3 different masks for the following scenarios:

Lane switch (1. left /2. right) and 3. Keeping Lane. Each mask crops out the irrelevant areas, keep only the area of interest.

The Algorithm's weak point:

- discontinuous lane separator angle is detected.

Since the algorithm chooses the median of discovered X values which represents the lane seperators,

It is affected by the range of X's which is detected by houghlinesP. We figure this works better over choosing the mean, as it's less sensitive to noise.

- identifying the next lane seperators in a lane change.

In our program, due to noise cancellation and shadow handling strategies, we limit the possible change in location of a lane separator between two following frames.

Therefore when switching lanes the lane separator in the corresponding direction moves in small steps instead of going straight to the new lane separator