Computer vision: lane detection

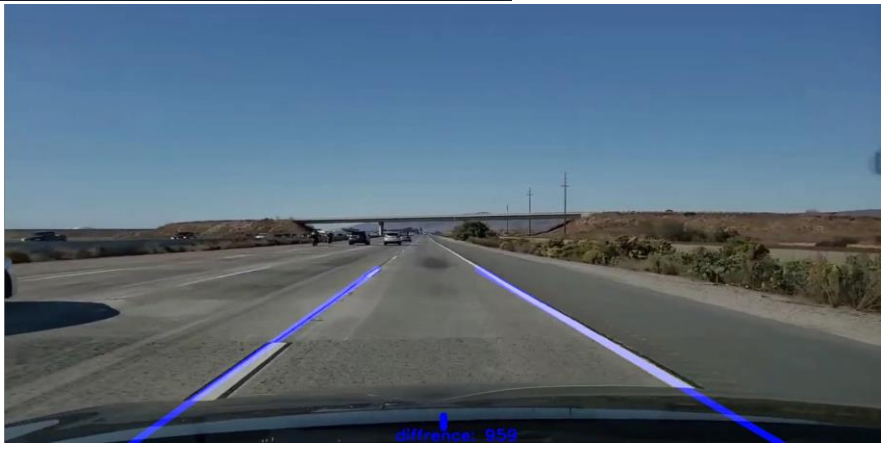
Assumptions:

1. We found that in each tested video there is a somewhat triangle shaped region that was important.

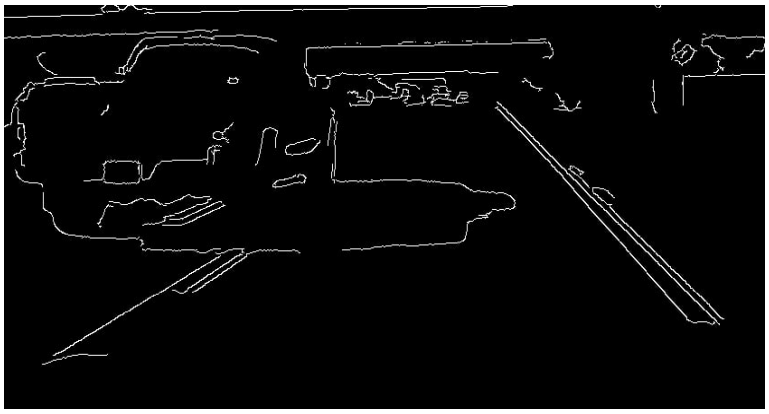
This area was formed from the starting of the lane to its end and the top of the triangle is at the point where the lines would intersect in the horizon.
The area around the triangle would only confuse the system as there are more lines to find from other lanes and other noise.

2. To remove any indication of color we turned the images to a grey scale format so the resulting image will not consider color as a factor but only the color brightness.
   To minimize the noise we used Gaussian function with parameters that we found after some tests.
   After we reduced noises we ran the Canny algorithm to find edges and hopefully the lanes we want.
   On top of that, in the Canny algorithm we tried a few different versions and concluded that those parameters are best.
   They reduced other edges found by accident and edges that are not interesting for this project.
3. After we found a few different lines using a combination of Canny for the edges and Hugh lines, we average the resulting points of each side using the slope to differentiate between right and left. We also removed lines with slope close to zero, aka horizontal lines because the lines of the lane are not a horizontal line.

Here a few examples of what the system outputs are:



Here are a few pictures of behind the scenes that show how the computer see the image. The left one shows the picture in grey scale without the blur effect with canny and the right one shows the picture in grey scale with the blur effect.



As we can see the addition of the blur effect reduces noise and irrelevant point of interest that the computer can mistake for points of a line.

In addition to the lane detection we also implemented a change in lane detection, when the car will shift lanes the system will inform the change and detect where the change in lane is, shift to the right or to the left.

We assumed that if the line we detected are in a difference of about 100 from the middle point for at least 5 frames it means that the car is about to shift to the side of the line, in other words if we look only on the right lane, the line detected is composed of two points and by so we can check the difference between this points and the middle point of the picture. If the difference is more than 100 absolute value then it means the lane is about to shift to the right.

Here is the code that checks the direction:

```
def __drawMiddleLine(self):
    """ draw a line in the middle of the image, and check if the diffrence between the lines are towrad the right or left """

    if self.__rightCords is None:
        return None

    distance_th = 100
    middle_x = int(self.__originalFrame.shape[1] / 2)
    middlePointText = (int(self.__originalFrame.shape[1] / 2) - 60 , 100)

    # Check left
    self.__checkDirection(self.__prevLeft, middle_x, distance_th, self.__Direction.RIGHT, self.__Direction.LEFT, self.__Direction.STRIGHT)

    # Check right
    self.__checkDirection(self.__prevRight, middle_x, distance_th, self.__Direction.LEFT, self.__Direction.RIGHT, self.__Direction.STRIGHT)

    display_text = ''
    if self.__movementDirection != self.__Direction.STRIGHT:
        display_text = self.__movementDirection.name

    cv2.putText(self.__originalFrame, display_text,
                middlePointText, cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 5, 2)

def __checkDirection(self, previosDirectionLane, middle, distanceThreshold, oposite Direction, wantedDirection, defaultDirection, maxFrames = 5):
    """ check if the wanted direction was the current direction for maxFrames frames back """
    laneShiftAmount = 0
    for direction in previosDirectionLane:
        if abs(direction - middle) < distanceThreshold:
            laneShiftAmount += 1
    if laneShiftAmount >= maxFrames and self.__movementDirection != oposite Direction:
        self.__movementDirection = wantedDirection
    elif self.__movementDirection == wantedDirection:
        self.__movementDirection = defaultDirection
```

As you can see we save groups of 5 frames and check at the end of each frame if the current direction is the same as the one that we want to check on, by so we can determine that the direction we want is the current direction and then change it and notify the user that the change has occurred.

As expected there are a few minor cases where our model misses, but in most of the time it detects the right lines and in close approximation to the real road lanes.
Our model fails when there are lines on the road that are bold in color, like when a car swerved it leaves some time a black line on the middle of the road, this line can be detected as a real line and can sometimes be in the right slope to be a real lane, one of the videos we picked for tests was with this exact description and by so our model struggled to understand what line was the real one, luckily at the end of the false line the model was back to his normal self and by so detected the real lanes as the lines in the picture.
Furthermore when changing lanes because we cropped the original picture to a triangle shaped picture we cannot see over the current lanes, that means that when the car shift lanes the farthest line is not in the cropped picture anymore and in the middle of the shift it is hard to understand what line is real because the system cannot see two lines, but when the shift is over and the car is in a lane the model recognize the two lanes as lines and the shift is finished, you can see an example for that in our video attached near the end where the system loses the right lane and need to recognize the next two lanes.