REQUIREMENT ANALYSIS DOCUMENT

COMPREHENSIVE ASSIGNMENT

Team: Imperium

Bokhodir Urinboev (leader):	U1610249
Dilorom Alieva:	U1610061
Feruza Latipova:	U1610072
Rakhmatjon Khasanov:	U1610183

TABLE OF CONTENTS

Structure of the code..... **Ошибка! Закладка не определена.**

structure of the code..... 2

Algorithm explonation..... 4

working video url..... 15

statement of team member contribution..... 16

STRUCTURE OF THE CODE

We divided our code into 8 functions and one while loop.

```
def image_denoising(img):
```

```
def edge_detection(img, sigma = 0.33):
```

```
def image_masking(img, vertices):
```

```
def hough_lines_detection(img):
```

```
def left_and_right_lines_separation(yy, lines):
```

```
def complete_line_drawing(img,lines):
```

```
def turn_prediction(img):
```

```
def demo_with_image():
```

```
while cap.isOpened():

    ret, frame = cap.read()
    if ret is False:
        break
    frame = cv2.resize(frame, size, interpolation=cv2.INTER_AREA)
    timer = cv2.getTickCount()

    denoised_img = image_denoising(frame)

    edge = edge_detection(denoised_img)

    masked_img = image_masking(edge, vertice)

    lines = hough_lines_detection(masked_img)

    left_and_right_lines = left_and_right_lines_separation(frame.shape[0], lines)

    line_image = complete_line_drawing(frame,  left_and_right_lines)

    prediction = turn_prediction(masked_img)
    fps = cv2.getTickFrequency()/(cv2.getTickCount() - timer)
    result = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
    cv2.putText(result, "FPS:"+ str(int(fps)), (50*width//1000, 50*width//1000), cv2.FONT_HERSHEY_SIMPLEX, 1.5*
    cv2.putText(result, "Prediction:" + prediction, (50*width//1024, 100*width//1000), cv2.FONT_HERSHEY_SIMPLEX

    out.write(result)
    cv2.imshow("Result", result)
    q = cv2.waitKey(1) & 0xff

    if q == ord('q'):
        break
cap.release()
out.release()
cv2.destroyAllWindows()
```

ALGORITHM EXPLONATION

Project is done using python programing language and illustrated in the Jupyter notebook.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

image on the left the libraries used in the project.

1) cv2 - Open Computer Vision Library.
2) NumPy – library for multi-dimensional arrays and matrices manipulation, along with a large collection of high-level mathematical functions to operate on these arrays.
3) Matplotlib - a plotting library.

```
cap = cv2.VideoCapture(filename)
width = int(cap.get(3))
height = int(cap.get(4))
size = (width, height)
vertice = np.array([[
    ((width*2)//10, height),
    ((width*9)//10, height),
    ((width*4)//7, (7*height)//10),
    ((width*75)//140, (9*height)//10),
    ((width*5)//10, (7*height)//10)]])

fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter("output_"+str(width)+"-"+str(height)+".mp4", fourcc, 30, size)
```

First of all, we take the cap object in order to run through frames of the video.

```
while cap.isOpened():
```

While we have a frame to run, we will do next operations.

1. We take the frame and resize it.

```
ret, frame = cap.read()
if ret is False:
    break
frame = cv2.resize(frame, size, interpolation=cv2.INTER_AREA)
```

In our case it is original size of the video.

```
width = int(cap.get(3))
height = int(cap.get(4))
size = (width, height)
```

2. Then we run the timer in order to calculate frame per second.

```
timer = cv2.getTickCount()
```

3. Then we call image_denoising function passing the frame as argument.

```
denoised_img = image_denoising(frame)
```

There we convert image to grey scale and using GaussianBlur we denoise the image

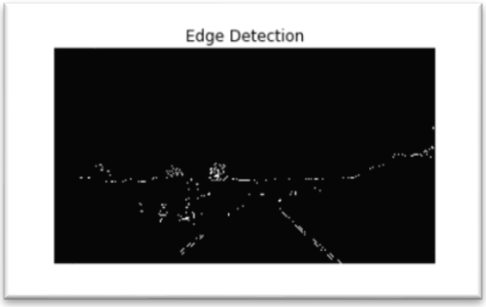
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (5, 5), 0, 0)

return blurred
```

4. Then denoised image we pass to the edge detection function.

```
edge = edge_detection(denoised_img)
```

There we detect the edges of the frame using Canny edge detection methodology.

```
v = np.median(img)
lower = int(max(0, (1.0 - sigma) * v))
upper = int(min(255, (1.0 + sigma) * v))
edged = cv2.Canny(img, lower, upper)
```



5. Then detected edge frame will be passed to the image masking function with ROI interest points.

```
masked_img = image_masking(edge, vertice)
```

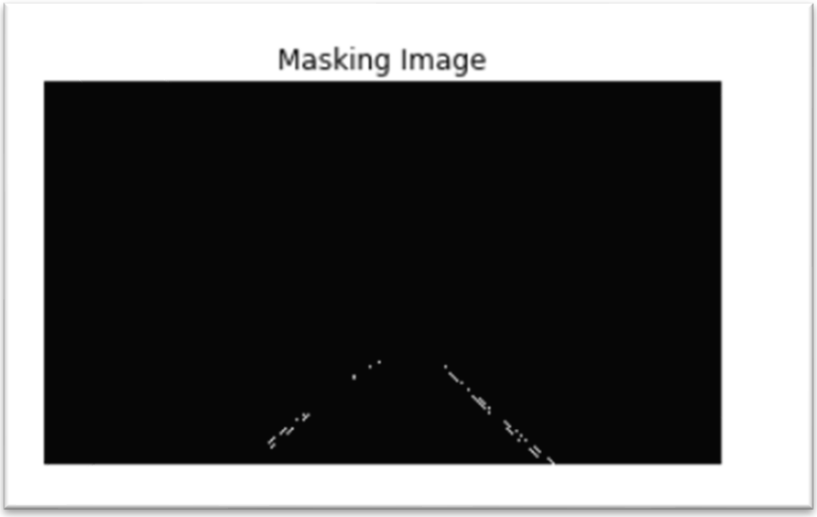
There, we apply a mask and set our region of interest

```
mask = np.zeros_like(img)

cv2.fillPoly(mask, vertices, 255)

masked = cv2.bitwise_and(img, mask)

return masked
```

In our case, region of interest is dynamic and automatically changed according to the size of the video.

```
vertice = np.array([[
    ((width*2)//10, height),
    ((width*9)//10, height),
    ((width*4)//7, (7*height)//10),
    ((width*75)//140, (9*height)//10),
    ((width*5)//10, (7*height)//10)]])
```

Our region of interest is M shaped so even we have a car ahead of the camera it will not confuse the lines.



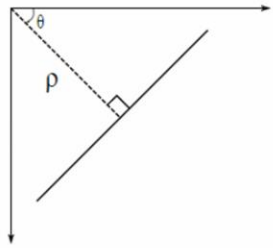
6. Then we will apply HoughLinesP in order to detect the lines.

```
lines = hough_lines_detection(masked_img)
```

```
def hough_lines_detection(img):

    return cv2.HoughLinesP(img, 1, np.pi/180, 25, minLineLength=10, maxLineGap=50)
```

Hough Transform is a popular technique of detecting any shape, if you can mathematically represent that shape. It can detect the shape even if a little bit of it is broken or distorted.

A line can be represented as $y = mx + c$ or in parametric form, as $\rho = x\cos\theta + y\sin\theta$ where ρ is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise (That direction varies on how you represent the coordinate system. This representation is used in OpenCV). Check below image:



7. Then we pass the lines to left and right separation function.

```
left_and_right_lines = left_and_right_lines_separation(frame.shape[0], lines)
```

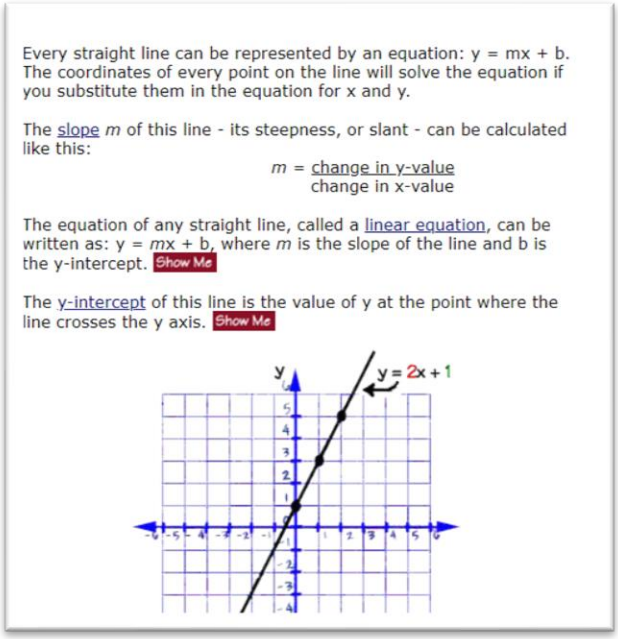
```
    def make_coordinates(y, line_parameters):
        slope, intercept = line_parameters
        y1 = y
        y2 = np.int64(y1*73/100)
        x1 = np.int64((y1 - intercept)/slope)
        x2 = np.int64((y2 - intercept)/slope)
        return np.array([x1,y1,x2,y2])

    global left_line
    global right_line
    left_fit = []
    right_fit = []

    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line.reshape(4)
            parameters = np.polyfit((x1, x2), (y1, y2), 1)
            slope = parameters[0]
            intercept = parameters[1]
            if slope < 0:
                left_fit.append((slope, intercept))
            else:
                right_fit.append((slope, intercept))
        left_fit_avg = np.average(left_fit, axis=0)
        right_fit_avg = np.average(right_fit, axis=0)

        if left_fit !=[]:
            left_line = make_coordinates(yy, left_fit_avg)

        if right_fit != []:
            right_line = make_coordinates(yy, right_fit_avg)
```

Combining multiple lines and taking the average line we will have two lines, left and right.

So, lets deep down into process.

```
global left_line
global right_line
left_fit = []
right_fit = []

if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line.reshape(4)
        parameters = np.polyfit((x1, x2), (y1, y2), 1)
        slope = parameters[0]
        intercept = parameters[1]
        if slope < 0:
            left_fit.append((slope, intercept))
        else:
            right_fit.append((slope, intercept))
    left_fit_avg = np.average(left_fit, axis=0)
    right_fit_avg = np.average(right_fit, axis=0)

    if left_fit !=[]:
        left_line = make_coordinates(yy, left_fit_avg)

    if right_fit != []:
        right_line = make_coordinates(yy, right_fit_avg)
```

First, we take the coordinates of each line and pass it to the polyfit function. That gives us slope and the intercept. By checking the slope, we will separate the lines to the right and left. Particularly if the slope is negative it is left line otherwise right line.



Then we will take the average and construct the coordinates of that line. So, it will make one line from some lines. It is called line regression.
We construct the coordinates for the left and right lines by passing the average slope and intercept values in the make_coordinates function.

```
def make_coordinates(y, line_parameters):
    slope, intercept = line_parameters
    y1 = y
    y2 = np.int64(y1*73/100)
    x1 = np.int64((y1 - intercept)/slope)
    x2 = np.int64((y2 - intercept)/slope)
    return np.array([x1,y1,x2,y2])
```

First, we take the point in the bottom of the video and the using slope and intercept we construct the line. We do it separately for the right and left lines.

8. So, we have coordinates of the lines now it is time to draw the lines.

```
line_image = complete_line_drawing(frame,  left_and_right_lines)
```

In some cases, we are might not be able to detect the line at that case we take the line from previous frame in order to make program run in the bad conditioned roads.
Then using the line function of OpenCV we draw the line on the image.

```
def complete_line_drawing(img,lines):
    global _x1
    global _y1
    global _x2
    global _y2
    line_img = np.zeros_like(img)
    # print(lines)
    if lines is not None:
        for line in lines:
            if line.size>0:
                x1, y1, x2, y2 = line.reshape(4)
                if x1 < 0 or x1 > width:
                    x1 = _x1
                if y1 < 0 or y1 > height:
                    y1 = _y1
                if x2 < 0 or x2 > width:
                    x2 = _x2
                if y2 < 0 or y2 > height:
                    y2 = _y2
                _x1, _y1, _x2, _y2 = x1, y1, x2, y2
                cv2.line(line_img,(x1, y1),(x2,y2),(255,0,0),10)
    return line_img
```

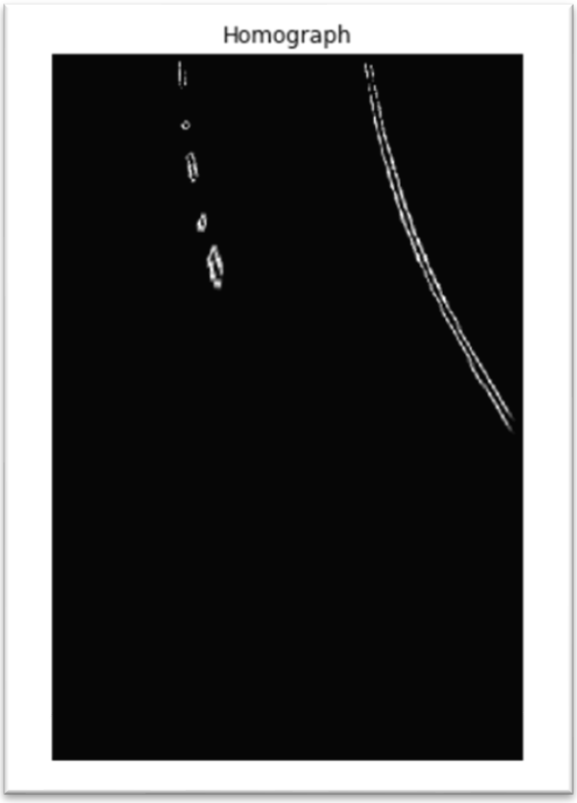
9. So as we have lines visable lets predict the turn.

```
prediction = turn_prediction(masked_img)
```

So in order to predict the turn we need the prospective view of the ROI.

```
# Source points for homography
src = np.array([
    ((width*2)/10, height),
    ((width*9)/10, height),
    ((width*4)/7, (7*height)/10),
    ((width*5)/10, (7*height)/10)], dtype="float32")

# Destination points for homography
shift = 80
dst = np.array([[0+shift, 0], [250+shift, 0], [340+shift, 500], [40+shift, 400]], dtype="float32")

#Homography
H_matrix = cv2.getPerspectiveTransform(src, dst)
```

Using WarpPrespective function we transform the image to the frontal view. Basically to the homograph.



Then using histogram method of NumPy function we get the pixel with max Y axis.
And accordingly, we compute the left and right max pixels using argmax function of NumPy. Then we will take thus pixels as left and right lane. Then we compute the center of the lane. Then taking image center we will compare with lane center.

```
if (lane_center - image_center < -15):
    return ("Turning left")
elif (lane_center - image_center < 15):
    return ("straight")
else:
    return ("Turning right")
```

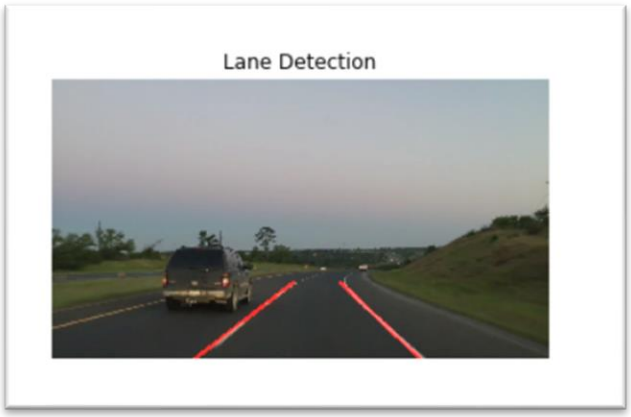
If lane_center -image_center is less than negative 15 it will be considered as turning left. If less than positive 15 it in straight way otherwise it is turning to the right.

10. Then we compute the fps using getTickFrequency method and getting the frequency then dividing it by getTickCount subtracting timer which we initialized earlier at starting point after resizing the image.

```
timer = cv2.getTickCount()
fps = cv2.getTickFrequency()/(cv2.getTickCount() - timer)
```

11. Then using add weight function we store the draw lines to the original image.

```
result = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
```



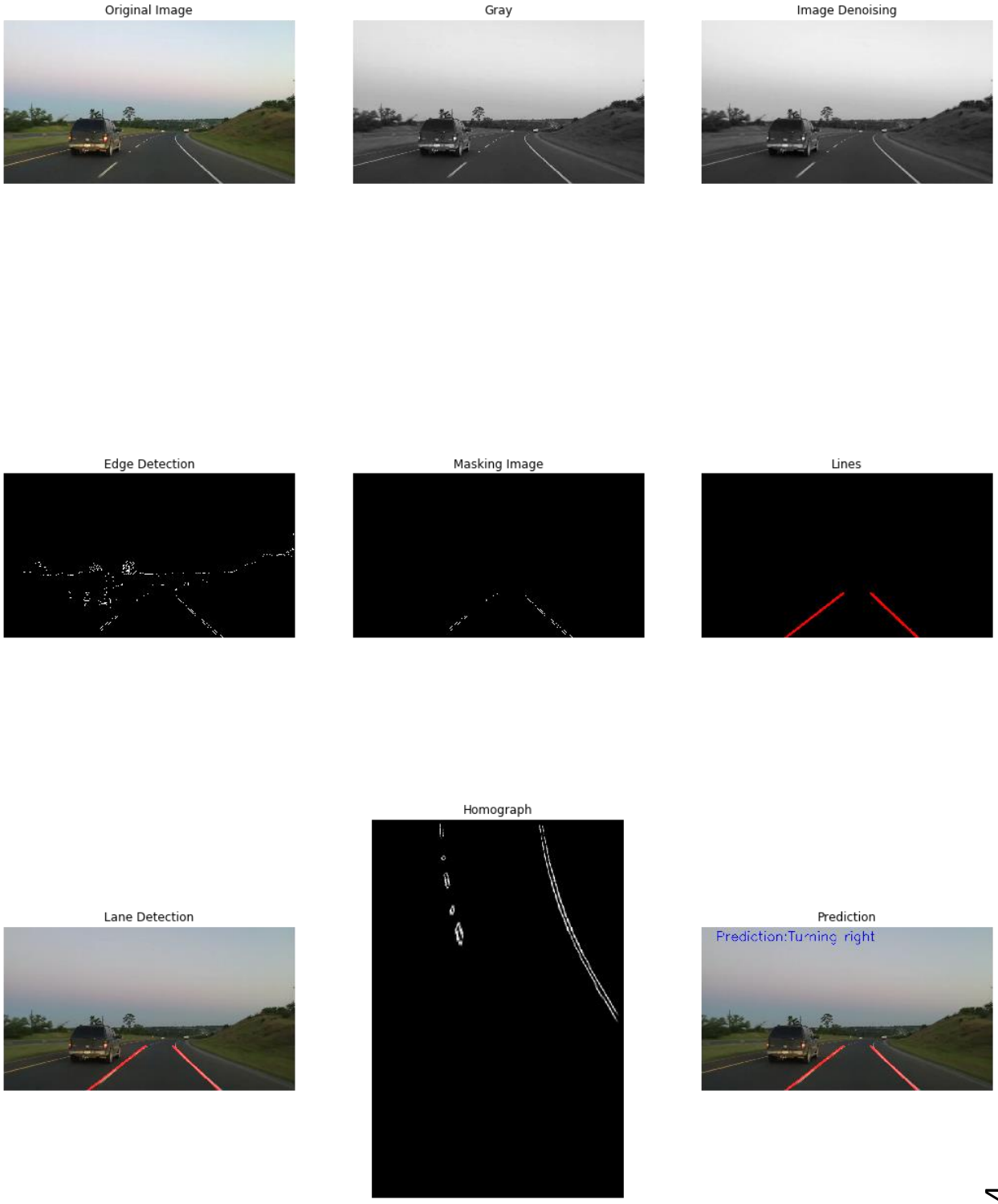
12. Then using putText method we add the fps and the prediction to the image.
Then using write function we store the frame into video at the we will get the video.

This was all about one iteration so each frame goes through this iteration.

We have one more function to display the images for illustration.

```
def demo_with_image():
    fig, ax = plt.subplots(3, 3,figsize=(18, 24))

    img=mpimg.imread('meta/original.png', 1)
    ax[0,0].imshow(img)
    ax[0,0].set_title('Original Image')
    ax[0,0].axis('off')  # clear x-axis and y-axis

    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    mpimg.imsave("meta/gray.png",gray_img, cmap='gray')
    ax[0,1].imshow(gray_img, cmap='gray')
    ax[0,1].set_title('Gray')
    ax[0,1].axis('off')  # clear x-axis and y-axis

    denoised_img = image_denoising(img)
    mpimg.imsave("meta/denoised.png",denoised_img, cmap='gray')
    ax[0,2].imshow(denoised_img, cmap='gray')
    ax[0,2].set_title('Image Denoising')
    ax[0,2].axis('off')  # clear x-axis and y-axis

    edge = edge_detection(denoised_img)
    mpimg.imsave("meta/edged.png",edge, cmap='gray')
    ax[1,0].imshow(edge, cmap='gray')
    ax[1,0].set_title('Edge Detection')
    ax[1,0].axis('off')  # clear x-axis and y-axis

    masked_img = image_masking(edge, vertice)
    mpimg.imsave("meta/masked.png",masked_img, cmap='gray')
    ax[1,1].imshow(masked_img, cmap='gray')
    ax[1,1].set_title('Masking Image')
    ax[1,1].axis('off')  # clear x-axis and y-axis
```

```
lines = hough_lines_detection(masked_img)
left_and_right_lines = left_and_right_lines_separation(img.shape[0], lines)
line_image = complete_line_drawing(img,  left_and_right_lines)
mpimg.imsave("meta/draw_lines.png",line_image)
ax[1,2].imshow(line_image)
ax[1,2].set_title('Lines')
ax[1,2].axis('off')  # clear x-axis and y-axis


result = cv2.addWeighted(img, 0.8, line_image, 1, 1)
mpimg.imsave("meta/lane_lines.png",result)
ax[2,0].imshow(result)
ax[2,0].set_title('Lane Detection')
ax[2,0].axis('off')  # clear x-axis and y-axis

prediction = turn_prediction(masked_img)
cv2.putText(result, "Prediction:" + prediction, (50*width//1024, 50*width//1000), cv2.FONT_HERSHEY_SIMPLEX, 1.


global H_matrix
new_img = cv2.warpPerspective(masked_img, H_matrix, (400, 600))
mpimg.imsave("meta/Homograph.png",new_img, cmap='gray')
ax[2,1].imshow(new_img, cmap='gray')
ax[2,1].set_title('Homograph')
ax[2,1].axis('off')  # clear x-axis and y-axis


mpimg.imsave("meta/prediction.png",result)
ax[2,2].imshow(result)
ax[2,2].set_title('Prediction')
ax[2,2].axis('off')  # clear x-axis and y-axis

plt.show()
```



The result of function illustrated above.

WORKING VIDEO URL

https://www.youtube.com/watch?v=7nxXQ-ayJac&list=PLZrsE2_darjJYr_MUIkQHNBj2Y1frJgpc&index=1

This link to the playlist

We uploaded for output videos for 5 sizes.

1) **400-300**
2) **640-480**
3) **800-600**
4) **1024-768**
5) **1280-720**

STATEMENT OF TEAM MEMBER CONTRIBUTION

Role	Team Member
Team leader	Bokhodir Urinboev
Recorder	Feruza Latipova
Data collector	Bokhodir Urinboev
Report creator	Rakhmatjon Khasanov
Presenter	Dilorom Alieva
Facilitator	Rakhmatjon Khasanov
Surveyor	Dilorom Alieva
Course observer	Feruza Latipova

Team Member	Contribution
Bokhodir Urinboev	Conducted online meetings, motivated and guided the team. Developed line drawing and prediction parts.
Feruza Latipova	Worked on report, collected data from all team members. Developed denoising, blurring, binarization, masking, line detection and separation parts
Dilorom Alieva	Provided software design of the code and documented requirement analysis. Developed denoising, blurring, binarization, masking, line detection and separation parts.
Rakhmatjon Khasanov	Concentrated on optimization and testing of the final code. Developed line drawing and prediction parts