```cpp
//James Rogers Mar 2022 (c) Plymouth University
#include<iostream>
#include<fstream>
#include<opencv2/opencv.hpp>

using namespace std;
using namespace cv;

//a drawing function that can draw a line based on rho and theta values.
//useful for drawing lines from the hough line detector.
void lineRT(Mat &Src, Vec2f L, Scalar color, int thickness){
    Point pt1, pt2;
    double a = cos(static_cast<double>(L[1]));
    double b = sin(static_cast<double>(L[1]));
    double x0 = a*static_cast<double>(L[0]);
    double y0 = b*static_cast<double>(L[0]);
    pt1.x = cvRound(x0 + 10000*(-b));
    pt1.y = cvRound(y0 + 10000*(a));
    pt2.x = cvRound(x0 - 10000*(-b));
    pt2.y = cvRound(y0 - 10000*(a));
    line(Src, pt1, pt2, color, thickness, LINE_AA);
}

int main()
{
    //Open video file
    VideoCapture CarVideo("../Task5/DashCam.mp4");
    //VideoCapture CarVideo("../Task5/road.mp4");
    //VideoCapture CarVideo("../Task5/Media2.mp4");
    if(!CarVideo.isOpened()){
        cout<<"Error opening video"<<endl;
        return -1;
    }
    //main program loop
    while(true){
        //open the next frame from the video file, or exit the loop if its the end
        Mat Frame;
        CarVideo.read(Frame);
        if(Frame.empty()){
            break;}
        //==========================Your code goes here==========================
        //Point start(0,0), end(0,BlurFrame.cols);
        Mat CanFrame, GreyFrame, BlurFrame;
        int lowerThreshold = 50, upperThreshold = 100;
        double rhoRes = 1;
        double thetaRes = M_PI/180;
        int HoughThreshold = 212;
        cvtColor(Frame, GreyFrame,COLOR_BGR2GRAY);  //greyscale, get rid of useless colour info
        blur(GreyFrame,BlurFrame, Size(3,3));
        Point start(0,0), end(BlurFrame.cols,BlurFrame.rows/2);
        rectangle(BlurFrame,start,end, Scalar(255,255,255), -1);
        Canny(BlurFrame, CanFrame, lowerThreshold, upperThreshold);
        vector<Vec2f> lines;
        vector<Vec4f> lines2;
        HoughLines(CanFrame, lines, rhoRes, thetaRes, HoughThreshold, 0 ,0 );
        //temporal differencing to reduce jitter
        //set-up variables
        int LineBottom = Frame.rows -1;
        int LineTop = Frame.rows - 300;
        vector< Point> corners;
        int workx[4]={0};
        int prevx[4]={0};
        int currx[4]={4};
        double upperbound = 1.01;
        double lowerbound = 0.99;

        for (unsigned int i = 0; i < lines.size(); i++) {
            //check the angle - only want to render the vertical lines
                for(int g = Frame.rows; g > Frame.rows - 300; g--){
                    //temporal differencing to reduce differ
                    for(unsigned int k = 0; k < lines.size(); k++){
                        if(lines[k][1]<=1){     //lower angle threshold check
                            currx[0] = (lines[k][0]/cos(lines[k][1]) - (LineTop*tan(lines[k][1])));
                            currx[2] = (lines[k][0]/cos(lines[k][1]) - (LineBottom*tan(lines[k]
[1])));
                            if ((currx[0] >= lowerbound*prevx[0])&&(currx[0] <=
upperbound*prevx[0])){

                                workx[0] = prevx[0] * 0.9 + currx[0] * 0.1;
                            } else {
                                workx[0] = prevx[0];
                            }

                            if ((currx[2] >= lowerbound*prevx[2])&&(currx[2] <=
upperbound*prevx[2])){

                                workx[2] = prevx[2] * 0.9 + currx[2] * 0.1;
                            } else {
                                workx[2] = prevx[2];
                            }

                            prevx[0] = currx[0];
                            prevx[2] = currx[2];
                        } else if(lines[k][1]>=2.3){        //upper angle threshold check
                            currx[1] = (lines[k][0]/cos(lines[k][1]) - (LineTop*tan(lines[k][1])));
                            currx[3] = (lines[k][0]/cos(lines[k][1]) - (LineBottom*tan(lines[k]
[1])));
                            if ((currx[1] >= lowerbound*prevx[1])&&(currx[1] <=
upperbound*prevx[1])){

                                workx[1] = prevx[1] * 0.9 + currx[1] * 0.1;
                            } else {
                                workx[1] = prevx[1];
                            }

                            if ((currx[3] >= lowerbound*prevx[3])&&(currx[3] <=
upperbound*prevx[3])){

                                workx[3] = prevx[3] * 0.9 + currx[3] * 0.1;
                            } else {
                                workx[3] = prevx[3];
                            }
                            prevx[1] = currx[1];
                            prevx[3] = currx[3];
                        }
                    }
                }
                //push back onto corners vector
                corners.push_back(Point(workx[0], LineTop));
                corners.push_back(Point(workx[1], LineTop));
                corners.push_back(Point(workx[3], LineBottom));
                corners.push_back(Point(workx[2], LineBottom));

                //midpoint coords
                Point topMid (((workx[0]+workx[1]) / 2), LineTop);
                Point btmMid (((workx[2]+workx[3]) / 2), LineBottom);
                //centreline
                line(Frame, topMid, btmMid, Scalar(255,0,0), 2);

                Mat overlay;
                double alpha = 0.2;
                Frame.copyTo(overlay);          //create duplicate frame for drawing on
                const Point *pts = (const cv::Point*) Mat(corners).data;
                int npts = Mat(corners).rows;
                fillPoly(overlay, &pts, &npts, 1, Scalar(0, 255, 0));   //draw lane overlay
                addWeighted(overlay, alpha, Frame, 1 - alpha, 0, Frame);    //blend overlay onto
frame
            }
            imshow("Video", Frame);
            waitKey(10);
        }
}
```