```
#include <opencv2/opencv.hpp>
#include <raspicam_cv.h>
#include <iostream>
#include <chrono>
#include <ctime>
#include <vector>
#include <wiringPi.h>
#include <string>

using namespace std;
using namespace cv;
using namespace raspicam;

//Creates the frames
Mat Frame, FramePerspective, FrameGray, Matrix, FrameThreshold, FrameEdge, FrameFinal;
Mat RnLane, FrameFinalDuplicate, FrameFinalDuplicate0, RnLaneEnd;
RaspiCam_Cv Camera;

void Capture();
void Threshold();
void RegionofInterest();
void Histogram();
void LineDetector();
void LineCenter();
void ScreenOutput(string direction);

Point2f Source[] ={Point2f(20,160), Point2f(390,160), Point2f(0,210), Point2f(400,210)};
Point2f Destination[] ={Point2f(130,0), Point2f(310,0), Point2f(130,240), Point2f(310,240)};

//Global variables
stringstream ss;
```

```
vector<int> HistogramLane;
vector<int> HistogramLaneEnd;
int LeftLane, RightLane, linecenter, framecenter, diff, LaneEnd;

void Setup (int argc, char **argv, RaspiCam_Cv &Camera){
	Camera.set( CAP_PROP_FRAME_WIDTH,("-w", argc, argv, 400));
	Camera.set( CAP_PROP_FRAME_HEIGHT, ("-h", argc, argv, 240));
	Camera.set( CAP_PROP_BRIGHTNESS, ("-br", argc, argv, 50));
	Camera.set( CAP_PROP_CONTRAST, ("-co", argc, argv, 50));
	Camera.set( CAP_PROP_SATURATION, ("-sa", argv, argc, 50));
	Camera.set( CAP_PROP_GAIN, ("-g", argc, argv, 50));
	Camera.set( CAP_PROP_FPS, ("-fps", argc, argv,0));
	}

int main(int argc, char **argv){

	wiringPiSetup();
	pinMode(21,OUTPUT);
	pinMode(22,OUTPUT);
	pinMode(23,OUTPUT);
	pinMode(24,OUTPUT);

	Setup(argc, argv, Camera);
	cout << "Connection to the Camera " << endl;
	if(!Camera.open()){
		cout << "Failed to connect to camera " << endl;
		return -1;
		}
	cout << "Camera ID= " << Camera.getId() << endl;

	while(1){
```

```
auto start = std::chrono::system_clock::now();

Capture();

waitKey(1);
auto end = std::chrono::system_clock::now();

std::chrono::duration<double> elapsed_seconds = end-start;

float t = elapsed_seconds.count();
int FPS = 1/t;

if(diff > -5 && diff < 5){
        digitalWrite(21,0);                     //0
        digitalWrite(22,0);
        digitalWrite(23,0);
        digitalWrite(24,0);
        ScreenOutput("Forward");
}else if(diff > -80 && diff < 70){
        digitalWrite(21,1);                     //1
        digitalWrite(22,0);
        digitalWrite(23,0);
        digitalWrite(24,0);
        ScreenOutput("Right");
}else if(diff >= 15 && diff < 25){
        digitalWrite(21,0);                     //2
        digitalWrite(22,1);
        digitalWrite(23,0);
        digitalWrite(24,0);
        ScreenOutput("Right");
}else if(diff > 25){
```

```
                digitalWrite(21,1);                     //3
                digitalWrite(22,1);
                digitalWrite(23,0);
                digitalWrite(24,0);
                ScreenOutput("Right");
            }else if(diff < 5 && diff > -15){
                digitalWrite(21,0);                     //4
                digitalWrite(22,1);
                digitalWrite(23,1);
                digitalWrite(24,0);
                ScreenOutput("Left");
            }else if(diff <= -15 && diff > -25){
                digitalWrite(21,1);                     //5
                digitalWrite(22,0);
                digitalWrite(23,1);
                digitalWrite(24,0);
                ScreenOutput("Left");
            }else if(diff < -25){
                digitalWrite(21,0);                     //6
                digitalWrite(22,1);
                digitalWrite(23,1);
                digitalWrite(24,0);
                ScreenOutput("Left");
            }
else if(diff > -5|| diff < -25 || diff > -90){
   digitalWrite(21, 0);
   digitalWrite(22, 0); // decimal = 8
   digitalWrite(23, 0);
   digitalWrite(24, 1);
   cout << "Stop" << endl;
}
```

```
RegionofInterest();
Threshold();
Histogram();
LineDetector();
LineCenter();

ss.str(" ");
ss.clear();
ss << "FPS: "  << FPS;
putText(Frame, ss.str(), Point2f(1,20), 0, 0.6, Scalar(0,0,0), 2);

ss.str(" ");
ss.clear();
ss << "Dist: "  << diff;
putText(Frame, ss.str(), Point2f(1,47), 0, 0.8, Scalar(0,0,255), 2);

namedWindow("Frame_RGB", WINDOW_KEEPRATIO);
moveWindow("Frame_RGB", 640,10);
resizeWindow("Frame_RGB",360,240);
imshow("Frame_RGB", Frame);

namedWindow("Perspective", WINDOW_KEEPRATIO);
moveWindow("Perspective", 640,250);
resizeWindow("Perspective",360,240);
imshow("Perspective", FramePerspective);

namedWindow("Frame_Edge", WINDOW_KEEPRATIO);
moveWindow("Frame_Edge", 640,490);
```

```
                resizeWindow("Frame_Edge",360,240);
                imshow("Frame_Edge", FrameFinal);

                }
        return 0;
        }

void RegionofInterest(){
        line(Frame,Source[0], Source[1], Scalar(0,0,255), 1);
        line(Frame,Source[1], Source[3], Scalar(0,0,255), 1);
        line(Frame,Source[3], Source[2], Scalar(0,0,255), 1);
        line(Frame,Source[2], Source[0], Scalar(0,0,255), 1);

        Matrix = getPerspectiveTransform(Source, Destination);   //Perspective transformation of
Region of interest

        warpPerspective(Frame,FramePerspective, Matrix, Size(400,240));
        }
        void Capture(){

                Camera.grab();
                Camera.retrieve(Frame);
                cvtColor(Frame,Frame,COLOR_BGR2RGB);
}

void Threshold(){
        cvtColor(FramePerspective, FrameGray, COLOR_RGB2GRAY);
        inRange(FrameGray,180,255,FrameThreshold);
        inRange(FrameGray,160,255,FrameGray);
        Canny(FrameGray, FrameEdge, 150, 400 ,3,false);
        add(FrameThreshold, FrameEdge, FrameFinal);
```

```
        cvtColor(FrameFinal, FrameFinal, COLOR_GRAY2RGB);
        cvtColor(FrameFinal, FrameFinalDuplicate, COLOR_RGB2GRAY); // for histogram only
        cvtColor(FrameFinal, FrameFinalDuplicate0, COLOR_RGB2GRAY); // for histogram only
        }

void Histogram(){

        HistogramLane.resize(400);
        HistogramLane.clear();

        for(size_t i{0}; i < Frame.size().width; ++i){

                RnLane = FrameFinalDuplicate(Rect(i,140,1,100));
                divide(255, RnLane, RnLane);
                HistogramLane.push_back((int)(sum(RnLane)[0]));

                }

        HistogramLaneEnd.resize(400);
        HistogramLaneEnd.clear();

        for(size_t i{0}; i < Frame.size().width; ++i){

                RnLaneEnd = FrameFinalDuplicate0(Rect(i,0,1,240));
                divide(255, RnLaneEnd, RnLaneEnd);
                HistogramLaneEnd.push_back((int)(sum(RnLaneEnd)[0]));

                }
                LaneEnd = sum(HistogramLaneEnd)[0];
                cout << "Lane End: " << LaneEnd << endl;
}
```

```
void LineCenter(){
	linecenter = (RightLane - LeftLane) / 2 + LeftLane;
	framecenter = 216;

	line(FrameFinal, Point2f(linecenter, 0 ), Point2f(linecenter, 240), Scalar(255,255,0), 3);
	line(FrameFinal, Point2f(framecenter, 0 ), Point2f(framecenter, 240), Scalar(255,0,0), 3);

	diff =  linecenter - framecenter;
	}

void LineDetector(){
	vector<int>:: iterator LeftPtr;
	LeftPtr = max_element(HistogramLane.begin(), HistogramLane.begin() +200);
	LeftLane = distance(HistogramLane.begin(),LeftPtr);

	vector<int>:: iterator RightPtr;
	RightPtr = max_element(HistogramLane.begin() + 240 , HistogramLane.end());
	RightLane = distance(HistogramLane.begin(), RightPtr);

	line(FrameFinal, Point2f(LeftLane, 0), Point2f(LeftLane, 240), Scalar(0,255,0),3);
	line(FrameFinal, Point2f(RightLane, 0), Point2f(RightLane, 240), Scalar(0,255,0),3);
	}

void ScreenOutput(string direction){
		ss.str(" ");
		ss.clear();
		ss << "Direction: "  << direction;
		putText(Frame, ss.str(), Point2f(110,20), 0, 0.7, Scalar(230,216,173), 2);
	}
```