**Final Code (Lane End detection & U-Turn Implementation)**

#include <opencv2/opencv.hpp>

#include <raspicam\_cv.h>

#include <iostream>

#include <chrono>

#include <ctime>

#include <wiringPi.h>

using namespace std;

using namespace cv;

using namespace raspicam;

Mat frame, Matrix, framePers, frameGray, frameThresh, frameEdge, frameFinal,

frameFinalDuplicate, frameFinalDuplicate1;

Mat ROILane, ROILaneEnd;

int LeftLanePos, RightLanePos, frameCenter, laneCenter, Result, laneEnd;

RaspiCam\_Cv Camera;

stringstream ss;

vector<int> histrogramLane;

vector<int> histrogramLaneEnd;

Point2f Source[] = {Point2f(40,135),Point2f(360,135),Point2f(0,185), Point2f(400,185)};

Point2f Destination[] = {Point2f(100,0),Point2f(280,0),Point2f(100,240), Point2f(280,240)};

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",argc,argv,50 ) );

Camera.set ( CAP\_PROP\_GAIN, ( "-g",argc,argv ,50 ) );

Camera.set ( CAP\_PROP\_FPS, ( "-fps",argc,argv,0));

}

void Capture()

{

Camera.grab();

Camera.retrieve( frame);

cvtColor(frame, frame, COLOR\_BGR2RGB);

}

void Perspective()

{

line(frame,Source[0], Source[1], Scalar(0,0,255), 2);

line(frame,Source[1], Source[3], Scalar(0,0,255), 2);

line(frame,Source[3], Source[2], Scalar(0,0,255), 2);

line(frame,Source[2], Source[0], Scalar(0,0,255), 2);

Matrix = getPerspectiveTransform(Source, Destination);

warpPerspective(frame, framePers, Matrix, Size(400,240));

}

void Threshold()

{

cvtColor(framePers, frameGray, COLOR\_RGB2GRAY);

inRange(frameGray, 230, 255, frameThresh);

Canny(frameGray,frameEdge, 900, 900, 3, false);

add(frameThresh, frameEdge, frameFinal);

cvtColor(frameFinal, frameFinal, COLOR\_GRAY2RGB);

cvtColor(frameFinal, frameFinalDuplicate, COLOR\_RGB2BGR); //used in histrogram

function only

cvtColor(frameFinal, frameFinalDuplicate1, COLOR\_RGB2BGR); //used in histrogram

function only

}

void Histrogram()

{

histrogramLane.resize(400);

histrogramLane.clear();

for(int i=0; i<400; i++)

//frame.size().width = 400

{

ROILane = frameFinalDuplicate(Rect(i,140,1,100));

divide(255, ROILane, ROILane);

histrogramLane.push\_back((int)(sum(ROILane)[0]));

}

histrogramLaneEnd.resize(400);

histrogramLaneEnd.clear();

for (int i = 0; i < 400; i++)

{

ROILaneEnd = frameFinalDuplicate1(Rect(i, 0, 1, 240));

divide(255, ROILaneEnd, ROILaneEnd);

histrogramLaneEnd.push\_back((int)(sum(ROILaneEnd)[0]));

}

laneEnd = sum(histrogramLaneEnd)[0];

cout<<"Lane END = "<<laneEnd<<endl;

}

void LaneFinder()

{

vector<int>:: iterator LeftPtr;

LeftPtr = max\_element(histrogramLane.begin(), histrogramLane.begin() + 150);

LeftLanePos = distance(histrogramLane.begin(), LeftPtr);

vector<int>:: iterator RightPtr;

RightPtr = max\_element(histrogramLane.begin() +250, histrogramLane.end());

RightLanePos = distance(histrogramLane.begin(), RightPtr);

line(frameFinal, Point2f(LeftLanePos, 0), Point2f(LeftLanePos, 240), Scalar(0, 255,0), 2);

line(frameFinal, Point2f(RightLanePos, 0), Point2f(RightLanePos, 240), Scalar(0,255,0), 2);

}

void LaneCenter()

{

laneCenter = (RightLanePos-LeftLanePos)/2 +LeftLanePos;

frameCenter = 188;

line(frameFinal, Point2f(laneCenter,0), Point2f(laneCenter,240), Scalar(0,255,0), 3);

line(frameFinal, Point2f(frameCenter,0), Point2f(frameCenter,240), Scalar(255,0,0), 3);

Result = laneCenter-frameCenter;

}

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

{

wiringPiSetup();

pinMode(21, OUTPUT);

pinMode(22, OUTPUT);

pinMode(23, OUTPUT);

pinMode(24, OUTPUT);

Setup(argc, argv, Camera);

cout<<"Connecting to camera"<<endl;

if (!Camera.open())

{

cout<<"Failed to Connect"<<endl;

}

cout<<"Camera Id = "<<Camera.getId()<<endl;

while(1)

{

auto start = std::chrono::system\_clock::now();

Capture();

Perspective();

Threshold();

Histrogram();

LaneFinder();

LaneCenter();

if (laneEnd > 3000)

{

digitalWrite(21, 1);

digitalWrite(22, 1); //decimal = 7

digitalWrite(23, 1);

digitalWrite(24, 0);

cout<<"Lane End"<<endl;

}

if (Result == 0)

{

digitalWrite(21, 0);

digitalWrite(22, 0); //decimal = 0

digitalWrite(23, 0);

digitalWrite(24, 0);

cout<<"Forward"<<endl;

}

else if (Result >0 && Result <10)

{

digitalWrite(21, 1);

digitalWrite(22, 0); //decimal = 1

digitalWrite(23, 0);

digitalWrite(24, 0);

cout<<"Right1"<<endl;

}

else if (Result >=10 && Result <20)

{

digitalWrite(21, 0);

digitalWrite(22, 1); //decimal = 2

digitalWrite(23, 0);

digitalWrite(24, 0);

cout<<"Right2"<<endl;

}

else if (Result >20)

{

digitalWrite(21, 1);

digitalWrite(22, 1); //decimal = 3

digitalWrite(23, 0);

digitalWrite(24, 0);

cout<<"Right3"<<endl;

}

else if (Result <0 && Result >-10)

{

digitalWrite(21, 0);

digitalWrite(22, 0); //decimal = 4

digitalWrite(23, 1);

digitalWrite(24, 0);

cout<<"Left1"<<endl;

}

else if (Result <=-10 && Result >-20)

{

digitalWrite(21, 1);

digitalWrite(22, 0); //decimal = 5

digitalWrite(23, 1);

digitalWrite(24, 0);

cout<<"Left2"<<endl;

}

else if (Result <-20)

{

digitalWrite(21, 0);

digitalWrite(22, 1); //decimal = 6

digitalWrite(23, 1);

digitalWrite(24, 0);

cout<<"Left3"<<endl;

}

if (laneEnd > 3000)

{

ss.str(" ");

ss.clear();

ss<<" Lane End";

putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(255,0,0), 2);

}

else if (Result == 0)

{

ss.str(" ");

ss.clear();

ss<<"Result = "<<Result<<" Move Forward";

putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(0,0,255), 2);

}

else if (Result > 0)

{

ss.str(" ");

ss.clear();

ss<<"Result = "<<Result<<"bMove Right";

putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(0,0,255), 2);

}

else if (Result < 0)

{

ss.str(" ");

ss.clear();

ss<<"Result = "<<Result<<" Move Left";

putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(0,0,255), 2);

}

namedWindow("orignal", WINDOW\_KEEPRATIO);

moveWindow("orignal", 0, 100);

resizeWindow("orignal", 640, 480);

imshow("orignal", frame);

namedWindow("Perspective", WINDOW\_KEEPRATIO);

moveWindow("Perspective", 640, 100);

resizeWindow("Perspective", 640, 480);

imshow("Perspective", framePers);

namedWindow("Final", WINDOW\_KEEPRATIO);

moveWindow("Final", 1280, 100);

resizeWindow("Final", 640, 480);

imshow("Final", frameFinal);

waitKey(1);