**CODES**

**Raspberry Code:**

#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;

// Image Processing variables

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)};

//Machine Learning variables

CascadeClassifier Stop\_Cascade, Object\_Cascade, Traffic\_Cascade;

Mat frame\_Stop, RoI\_Stop, gray\_Stop, frame\_Object, RoI\_Object, gray\_Object, frame\_Traffic, RoI\_Traffic, gray\_Traffic;

vector<Rect> Stop, Object, Traffic;

int dist\_Stop, dist\_Object, dist\_Traffic;

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\_Stop, COLOR\_BGR2RGB);

cvtColor(frame, frame\_Object, COLOR\_BGR2RGB);

cvtColor(frame, frame\_Traffic, COLOR\_BGR2RGB);

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;

}

void Stop\_detection()

{

if(!Stop\_Cascade.load("//home//pi//Desktop//MACHINE LEARNING//Stop\_cascade.xml"))

{

printf("Unable to open stop cascade file");

}

RoI\_Stop = frame\_Stop(Rect(200,0,200,140));

cvtColor(RoI\_Stop, gray\_Stop, COLOR\_RGB2GRAY);

equalizeHist(gray\_Stop, gray\_Stop);

Stop\_Cascade.detectMultiScale(gray\_Stop, Stop);

for(int i=0; i<Stop.size(); i++)

{

Point P1(Stop[i].x, Stop[i].y);

Point P2(Stop[i].x + Stop[i].width, Stop[i].y + Stop[i].height);

rectangle(RoI\_Stop, P1, P2, Scalar(0, 0, 255), 2);

putText(RoI\_Stop, "Stop Sign", P1, FONT\_HERSHEY\_PLAIN, 1, Scalar(0, 0, 255, 255), 2);

dist\_Stop = (-1.07)\*(P2.x-P1.x) + 102.597;

ss.str(" ");

ss.clear();

ss<<"D = "<<dist\_Stop<<"cm";

putText(RoI\_Stop, ss.str(), Point2f(1,130), 0,1, Scalar(0,0,255), 2);

}

}

void Traffic\_detection()

{

if(!Traffic\_Cascade.load("//home//pi//Desktop//MACHINE LEARNING//Trafficc\_cascade.xml"))

{

printf("Unable to open traffic cascade file");

}

RoI\_Traffic = frame\_Traffic(Rect(200,0,200,140));

cvtColor(RoI\_Traffic, gray\_Traffic, COLOR\_RGB2GRAY);

equalizeHist(gray\_Traffic, gray\_Traffic);

Traffic\_Cascade.detectMultiScale(gray\_Traffic, Traffic);

for(int i=0; i<Traffic.size(); i++)

{

Point P1(Traffic[i].x, Traffic[i].y);

Point P2(Traffic[i].x + Traffic[i].width, Traffic[i].y + Traffic[i].height);

rectangle(RoI\_Traffic, P1, P2, Scalar(0, 0, 255), 2);

putText(RoI\_Traffic, "Traffic Light", P1, FONT\_HERSHEY\_PLAIN, 1, Scalar(0, 0, 255, 255), 2);

dist\_Traffic = (-1.07)\*(P2.x-P1.x) + 102.597;

ss.str(" ");

ss.clear();

ss<<"D = "<<P2.x-P1.x<<"cm";

putText(RoI\_Traffic, ss.str(), Point2f(1,130), 0,1, Scalar(0,0,255), 2);

}

}

void Object\_detection()

{

if(!Object\_Cascade.load("//home//pi//Desktop//MACHINE LEARNING//Object\_cascade.xml"))

{

printf("Unable to open Object cascade file");

}

RoI\_Object = frame\_Object(Rect(100,50,200,190));

cvtColor(RoI\_Object, gray\_Object, COLOR\_RGB2GRAY);

equalizeHist(gray\_Object, gray\_Object);

Object\_Cascade.detectMultiScale(gray\_Object, Object);

for(int i=0; i<Object.size(); i++)

{

Point P1(Object[i].x, Object[i].y);

Point P2(Object[i].x + Object[i].width, Object[i].y + Object[i].height);

rectangle(RoI\_Object, P1, P2, Scalar(0, 0, 255), 2);

putText(RoI\_Object, "Object", P1, FONT\_HERSHEY\_PLAIN, 1, Scalar(0, 0, 255, 255), 2);

dist\_Object = (-0.48)\*(P2.x-P1.x) + 56.6;

ss.str(" ");

ss.clear();

ss<<"D = "<<dist\_Object<<"cm";

putText(RoI\_Object, ss.str(), Point2f(1,130), 0,1, Scalar(0,0,255), 2);

}

}

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();

Stop\_detection();

Object\_detection();

Traffic\_detection();

if (dist\_Stop > 5 && dist\_Stop < 20)

{

digitalWrite(21, 0);

digitalWrite(22, 0); //decimal = 8

digitalWrite(23, 0);

digitalWrite(24, 1);

cout<<"Stop Sign"<<endl;

dist\_Stop = 0;

goto Stop\_Sign;

}

if (dist\_Object > 5 && dist\_Object < 30)

{

digitalWrite(21, 1);

digitalWrite(22, 0); //decimal = 9

digitalWrite(23, 0);

digitalWrite(24, 1);

cout<<"Object"<<endl;

dist\_Object = 0;

goto Object;

}

if (laneEnd > 4500)

{

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;

}

Stop\_Sign:

Object:

if (laneEnd > 4500)

{

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<<" (Move 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);

namedWindow("Stop Sign", WINDOW\_KEEPRATIO);

moveWindow("Stop Sign", 1280, 580);

resizeWindow("Stop Sign", 640, 480);

imshow("Stop Sign", RoI\_Stop);

namedWindow("Object", WINDOW\_KEEPRATIO);

moveWindow("Object", 640, 580);

resizeWindow("Object", 640, 480);

imshow("Object", RoI\_Object);

namedWindow("Traffic", WINDOW\_KEEPRATIO);

moveWindow("Traffic", 0, 580);

resizeWindow("Traffic", 640, 480);

imshow("Traffic", RoI\_Traffic);

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;

//cout<<"FPS = "<<FPS<<endl;

}

return 0;

}

**Arduino code:**

int i =0;

unsigned long int j =0;

const int EnableL = 5;

const int HighL = 6; // LEFT SIDE MOTOR

const int LowL =7;

const int EnableR = 10;

const int HighR = 8; //RIGHT SIDE MOTOR

const int LowR =9;

const int D0 = 0; //Raspberry pin 21 LSB

const int D1 = 1; //Raspberry pin 22

const int D2 = 2; //Raspberry pin 23

const int D3 = 3; //Raspberry pin 24 MSB

int a,b,c,d,data;

void setup() {

pinMode(EnableL, OUTPUT);

pinMode(HighL, OUTPUT);

pinMode(LowL, OUTPUT);

pinMode(EnableR, OUTPUT);

pinMode(HighR, OUTPUT);

pinMode(LowR, OUTPUT);

pinMode(D0, INPUT\_PULLUP);

pinMode(D1, INPUT\_PULLUP);

pinMode(D2, INPUT\_PULLUP);

pinMode(D3, INPUT\_PULLUP);

}

void Data()

{

a = digitalRead(D0);

b = digitalRead(D1);

c = digitalRead(D2);

d = digitalRead(D3);

data = 8\*d+4\*c+2\*b+a;

}

void Forward()

{

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL,255);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableR,255);

}

void Backward()

{

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

analogWrite(EnableL,255);

digitalWrite(HighR, HIGH);

digitalWrite(LowR, LOW);

analogWrite(EnableR,255);

}

void Stop()

{

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL,0);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableR,0);

}

void Left1()

{

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL,160);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableR,255);

}

void Left2()

{

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL,90);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableR,255);

}

void Left3()

{

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL,50);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableR,255);

}

void Right1()

{

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL,255);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableR,160); //200

}

void Right2()

{

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL,255);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableR,90); //160

}

void Right3()

{

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL,255);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableR,50); //100

}

void UTurn()

{

analogWrite(EnableL, 0);

analogWrite(EnableR, 0);

delay(400);

analogWrite(EnableL, 250);

analogWrite(EnableR, 250); //forward

delay(1000);

analogWrite(EnableL, 0);

analogWrite(EnableR, 0);

delay(400);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW); // left

digitalWrite(LowR, HIGH);

analogWrite(EnableL, 255);

analogWrite(EnableR, 255);

delay(700);

analogWrite(EnableL, 0);

analogWrite(EnableR, 0);

delay(400);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); // forward

digitalWrite(LowR, HIGH);

analogWrite(EnableL, 255);

analogWrite(EnableR, 255);

delay(900);

analogWrite(EnableL, 0);

analogWrite(EnableR, 0);

delay(400);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW); //left

digitalWrite(LowR, HIGH);

analogWrite(EnableL, 255);

analogWrite(EnableR, 255);

delay(700);

analogWrite(EnableL, 0);

analogWrite(EnableR, 0);

delay(1000);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW);

digitalWrite(LowL, HIGH);

analogWrite(EnableL, 150);

analogWrite(EnableR, 150);

delay(300);

}

void Object()

{

analogWrite(EnableL, 0);

analogWrite(EnableR, 0); //stop

delay(1000);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH); //left

analogWrite(EnableL, 250);

analogWrite(EnableR, 250);

delay(500);

analogWrite(EnableL, 0);

analogWrite(EnableR, 0); //stop

delay(200);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH); //forward

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableL, 255);

analogWrite(EnableR, 255);

delay(1000);

analogWrite(EnableL, 0); //stop

analogWrite(EnableR, 0);

delay(200);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, HIGH); //right

digitalWrite(LowR, LOW);

analogWrite(EnableL, 255);

analogWrite(EnableR, 255);

delay(500);

analogWrite(EnableL, 0); //stop

analogWrite(EnableR, 0);

delay(1000);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); // forward

digitalWrite(LowR, HIGH);

analogWrite(EnableL, 150);

analogWrite(EnableR, 150);

delay(500);

i = i+1;

}

void Lane\_Change()

{

analogWrite(EnableL, 0);

analogWrite(EnableR, 0); //stop

delay(1000);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, HIGH);

digitalWrite(LowR, LOW); //Right

analogWrite(EnableL, 250);

analogWrite(EnableR, 250);

delay(500);

analogWrite(EnableL, 0);

analogWrite(EnableR, 0); //stop

delay(200);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH); //forward

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnableL, 255);

analogWrite(EnableR, 255);

delay(800);

analogWrite(EnableL, 0); //stop

analogWrite(EnableR, 0);

delay(200);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW); //LEFT

digitalWrite(LowR, HIGH);

analogWrite(EnableL, 255);

analogWrite(EnableR, 255);

delay(500);

analogWrite(EnableL, 0); //stop

analogWrite(EnableR, 0);

delay(1000);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); // forward

digitalWrite(LowR, HIGH);

analogWrite(EnableL, 150);

analogWrite(EnableR, 150);

delay(500);

}

void loop()

{

if (j > 25000)

{

Lane\_Change();

i = 0;

j = 0;

}

Data();

if(data==0)

{

Forward();

if (i>0)

{

j = j+1;

}

}

else if(data==1)

{

Right1();

if (i>0)

{

j = j+1;

}

}

else if(data==2)

{

Right2();

if (i>0)

{

j = j+1;

}

}

else if(data==3)

{

Right3();

if (i>0)

{

j = j+1;

}

}

else if(data==4)

{

Left1();

if (i>0)

{

j = j+1;

}

}

else if(data==5)

{

Left2();

if (i>0)

{

j = j+1;

}

}

else if(data==6)

{

Left3();

if (i>0)

{

j = j+1;

}

}

else if(data==7)

{

UTurn();

}

else if (data==8)

{

analogWrite(EnableL, 0);

analogWrite(EnableR, 0);

delay(4000);

analogWrite(EnableL, 150);

analogWrite(EnableR, 150);

delay(1000);

}

else if(data==9)

{

Object();

}

else if(data==10)

{

analogWrite(EnableL, 0);

analogWrite(EnableR, 0);

delay(2000);

}

else if(data>10)

{

Stop();

}

}