## Background

A Robot is any machine which is completely automatic, i.e. it starts on its own, decides its own way of work and stops on its own. It is actually a replica of human being, which has been designed to ease human burden. Robots can be fixed robots or mobile robots. Mobile Robots are robots with a mobile base which makes the robot move freely in the environment. One of the advanced mobile robots is the Lane Follower Robot. It is basically a robot which follows a particular path or trajectory and decides its own course of action which interacts with obstacle.

As technology becomes increasingly important in today's world, it is invaluable to not only learn how to use technology, but also to understand how to create it.

* It gives visual grasp of math and science.
* It builds logical thinking.
* It brings out innovation and creativity.
* It enhances problem solving skills.

## Motivation

## Ants always travel in a line, following an invisible route in search of food, or back home.

## A robot that runs on line also. Which is a perfect or near the artificial nature. The purpose of this project is to recreate in terms of machines what one sees around to solve a problem or fulfill a requirement.

The area will be benefitted from the project:

* Industrial automated equipment carriers.
* Entertainment and small household applications.
* Tour guides in museums and other similar applications.

# CHAPTER 4

**4.1 Proposed System Overview**

Robotics is an interesting subject to discuss about and in this advanced world robots are becoming a part of our life. In this project, we have discussed about a robot which is capable of following a lane without the help of any external source. The Lane following Mobile Robot uses four DC motors to control rear and front wheels. It has Raspberry Pi Camera Sensor for detection of white lines. The proposed system detects the drivable region and road lines drawn on the floor from Camera data. The Path consists of a white line on a gray surface. The control system senses these lines and manoeuvre the robot to stay on course.

**Computer Vision**

Computer vision is an interdisciplinary field, in which we are trying to describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions. In other words, computer vision provides us the understanding of the scene.

**Digital Image Processing**

Image processing is a type of signal processing, which has an image, series of images or video frames as the input, while the output is processed images, maybe in a different format. Image processing is often exploited in computer vision to manipulate digital images. Vision can be considered the most important part of human perception and it plays the same role in machine sensing. However, unlike human, machines "see" the world as digital images of 2D or 3D scenes produced by sensors. The 2D digital image, which, in many cases, represents a projection of a 3D scene, may be defined as a two-dimensional array of intensity samples called pixels. Pixel values typically represent gray levels, colors, opacities, etc. 2D digital image can be processed and manipulated by computer programs to produce useful information about the world. Several techniques in digital image processing are noise removal, image blur, image sharpening, object recognition, etc.

**OpenCV Library**

Most of the image processing techniques in this thesis, for example image reading, gray scaling, thresholding, etc., are realized by using Open-Source Computer Vision Library (OpenCV), which is the most popular open-source library in computer vision. OpenCV (Open-Source Computer Vision Library) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. OpenCV is written in optimized C/C++ language and has a modular structure.

Listed below are several examples of using OpenCV in image processing:

**Gaussian Smoothing or Gaussian Blurring**: the most commonly used blurring method which blurs an image using a Gaussian filter.

**Grayscaling:** converts an image from RGB color space to grayscale.

**Thresholding:** transforms a grayscale image to a binary image.

**Region of Interest**

A Region of Interest (ROI) is a portion of an image that you want to filter or operate on in some way.

**Gaussian Filter**

In this paper, we have applied Gaussian filter for pre-processing. Gaussian filter can remove high frequency noise in image and it can make the lane detection accuracy

**Canny Edge Detection**

Canny edge detector is based on Gaussian blur and sobel masking method. Sobel edge masking is a kind of 1st order derivation operator, it can detect edge of every direction and normalization of every pixel data.

**Hough Transform**

The Hough Transform is a popular and relatively fast method for finding simple mathematical forms such as lines and circles in an image. It was originally developed by Hough (1962) to recognize straight lines in a binary image.

When implementing the project using camera with raspberry pi for image processing, we analyze the frame in seconds for finding the lane and there are few steps at first that are executed.

**Step 1:** Captures the image and then that image is converted from BGR to RGB as image with RGB format works best for analyzing as RGB image is simply a composite of three independent grayscale image.

**Step 2:** Region of Interest (ROI) is created for every image. As region of interest focuses on some portion of image. The Region of Interest covers most of lane portion of the frame.

**Step 3:** Another frame is created from that Region of Interest area of the frame to analyze the lane and the surroundings which basically called as perspective transformation.

**Step 4:** Created frame is converted to grayscale image using threshold operations where the gray portion of lane is given more threshold value and the noise is given less threshold value and as we see the lane clearly from the frame the distance between the lane center and right, left lane is determined. The distance is then stored and the data for Arduino to control the motors is evaluated using if-else conditions.

**Step 5:** The data from Raspberry pi is given to Arduino using four digital pins and by making a whole binary format from those pins for Arduino to control the motors. The binary format value is initialized to the variable and using the variable, left, right and stop functions are implemented.

## Logical Flow Chart

Figure 4.2.2 Logical Flow Chart

## Circuit Diagram

## 



Figure 4.2.2 Circuit diagram

# CHAPTER 5

# 5.1 Implementation

# 5.1.1 Hardware Implementation

# 

Figure 5.1.2 Back View of Protoype

# 

Figure 5.1.1 Front View of Prototype



Figure 5.1.3 Design of Lane

## 5.1.2 Raspberry Pi Program

## #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(80, 160), Point2f(300, 160), Point2f(40, 210), Point2f(340, 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 > 5 && diff < 15)

## {

## 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");

## }

## 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 << "Diretion: " << direction;

## putText(Frame, ss.str(), Point2f(110, 20), 0, 0.7, Scalar(230, 216, 173), 2);

## }

## 5.1.3 Arduino Program

//Left side of Motors

const int EnL = 5;

const int HighL = 6;

const int LowL = 7;

//Right side of Motors

const int EnR = 10;

const int HighR = 8;

const int LowR = 9;

//Raspberry Pi connection

const int D0 = 11; // Raspberry Pi pin 21

const int D1 = 12; // Raspberry Pi pin 22

const int D2 = 2; // Raspberry Pi pin 23

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

int a,b,c,d,binary;

void setup() {

pinMode(EnL, OUTPUT);

pinMode(HighL, OUTPUT);

pinMode(LowL, OUTPUT);

pinMode(EnR, 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);

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

}

//Forward

void Forward(){

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, HIGH);

digitalWrite(LowR, LOW);

analogWrite(EnL, 250);

analogWrite(EnR, 250);

}

//Backward

void Backward(){

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnL, 250);

analogWrite(EnR, 250);

}

//Left

void Right(){

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, HIGH);

digitalWrite(LowR, LOW);

analogWrite(EnL,230);

analogWrite(EnR, 0);

}

void Left(){

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnL, 0);

analogWrite(EnR,230);

}

void Stop(){

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, HIGH);

digitalWrite(LowR, LOW);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

}

void Right\_soft(){

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnR,255);

analogWrite(EnL,255);

delay(1000);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, HIGH);

digitalWrite(LowR, LOW);

analogWrite(EnR,255);

analogWrite(EnL,255);

delay(200);

}

void Right\_medium(){

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnR,170);

analogWrite(EnL,255);

}

void Left\_soft(){

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnR,255);

analogWrite(EnL,255);

delay(1000);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnR,255);

analogWrite(EnL,255);

delay(200);

}

void Left\_medium(){

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW);

digitalWrite(LowR, HIGH);

analogWrite(EnR,255);

analogWrite(EnL,170);

}

void Object(){

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW); //left

digitalWrite(LowR, HIGH);

analogWrite(EnL, 255);

analogWrite(EnR,255);

delay(1100);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); //Forward

digitalWrite(LowR, HIGH);

analogWrite(EnL, 250);

analogWrite(EnR, 250);

delay(1000);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, HIGH); //Right

digitalWrite(LowR, LOW);

analogWrite(EnL, 255);

analogWrite(EnR,255);

delay(1000);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); //Forward

digitalWrite(LowR, HIGH);

analogWrite(EnL, 250);

analogWrite(EnR, 250);

delay(1400);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, HIGH); //Right

digitalWrite(LowR, LOW);

analogWrite(EnL, 255);

analogWrite(EnR,255);

delay(1400);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); //Forward

digitalWrite(LowR, HIGH);

analogWrite(EnL, 250);

analogWrite(EnR, 250);

delay(800);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW); //left

digitalWrite(LowR, HIGH);

analogWrite(EnL, 255);

analogWrite(EnR,255);

delay(1000);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); //Forward

digitalWrite(LowR, HIGH);

analogWrite(EnL, 255);

analogWrite(EnR, 255);

delay(200);

}

void UTurn(){

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

analogWrite(EnL, 250);

analogWrite(EnR, 250);

delay(1700);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW); //left

digitalWrite(LowR, HIGH);

analogWrite(EnL, 255);

analogWrite(EnR,255);

delay(1800);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); //Forward

digitalWrite(LowR, HIGH);

analogWrite(EnL, 250);

analogWrite(EnR, 250);

delay(900);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, HIGH);

digitalWrite(LowL, LOW);

digitalWrite(HighR, LOW); //left

digitalWrite(LowR, HIGH);

analogWrite(EnL, 255);

analogWrite(EnR,255);

delay(1000);

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(400);

digitalWrite(HighL, LOW);

digitalWrite(LowL, HIGH);

digitalWrite(HighR, LOW); //Forward

digitalWrite(LowR, HIGH);

analogWrite(EnL, 255);

analogWrite(EnR, 255);

delay(200);

}

void StopS(){

analogWrite(EnL, 0);

analogWrite(EnR, 0);

delay(4500);

analogWrite(EnL, 250);

analogWrite(EnR, 250);

delay(1000);

}

void loop() {

Data();

if (binary == 0){

Forward();

}

else if(binary == 1){

Right\_medium();

}

else if(binary == 2){

Right();

}

else if(binary == 3){

Right();

}

else if(binary == 4){

Left\_medium();

}

else if(binary == 5){

Left();

}

else if(binary == 6){

Left();

}

else if(binary == 7){

UTurn();

}

else if(binary == 8){

StopS();

}

else if(binary == 9){

Object();

}

else if(binary > 9){

Stop();

}

}

## Results

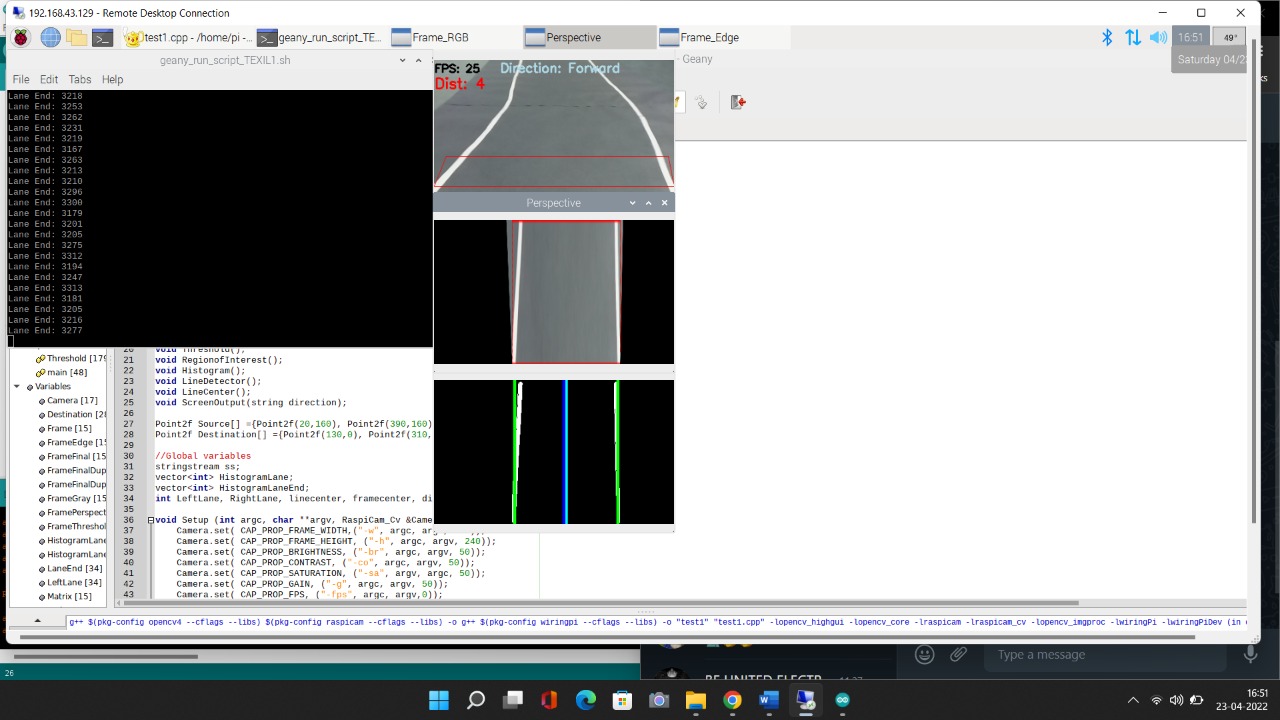


Figure 7.1.1 Forward Direction

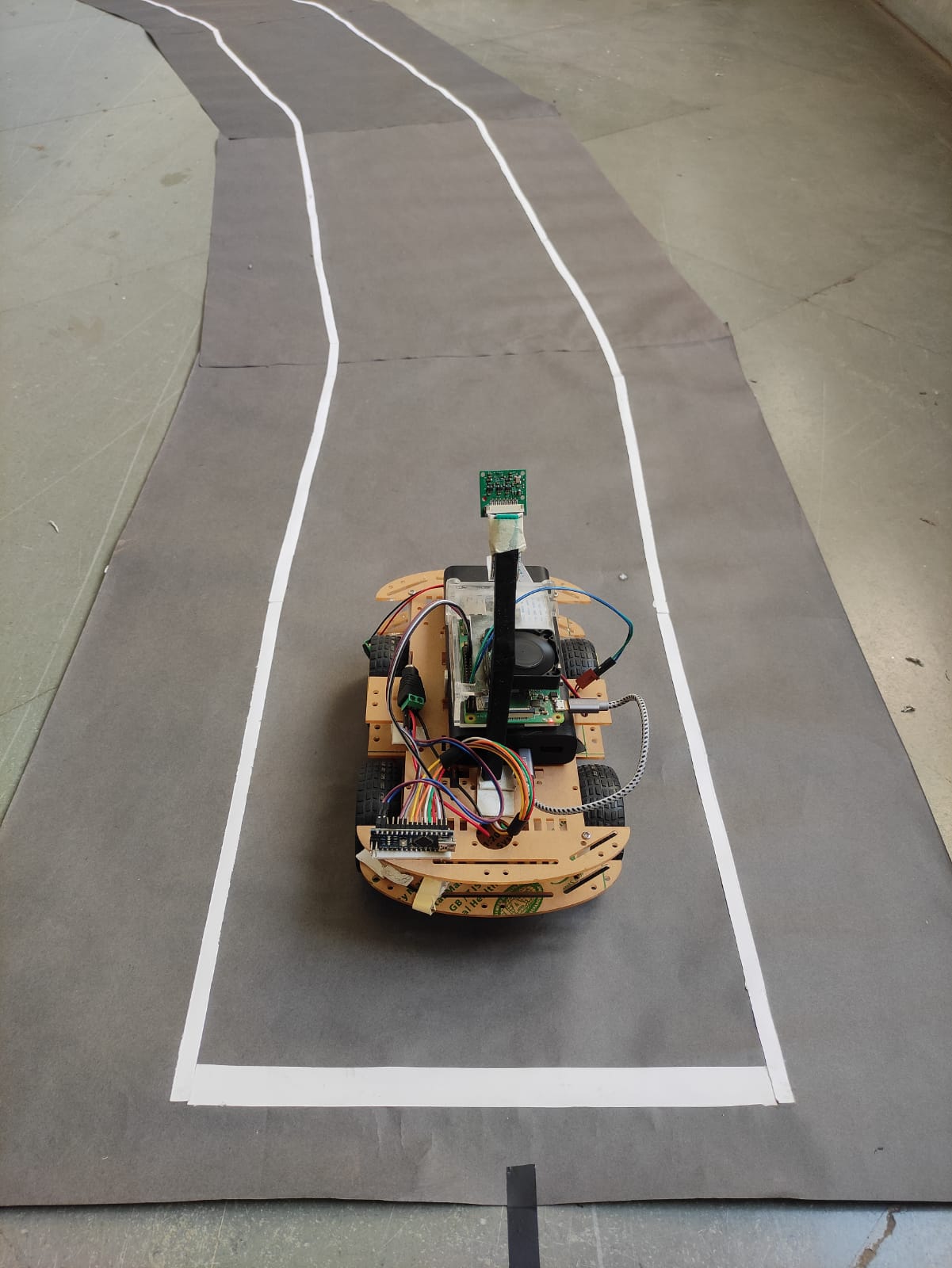


Figure 7.1.2 Forward Direction

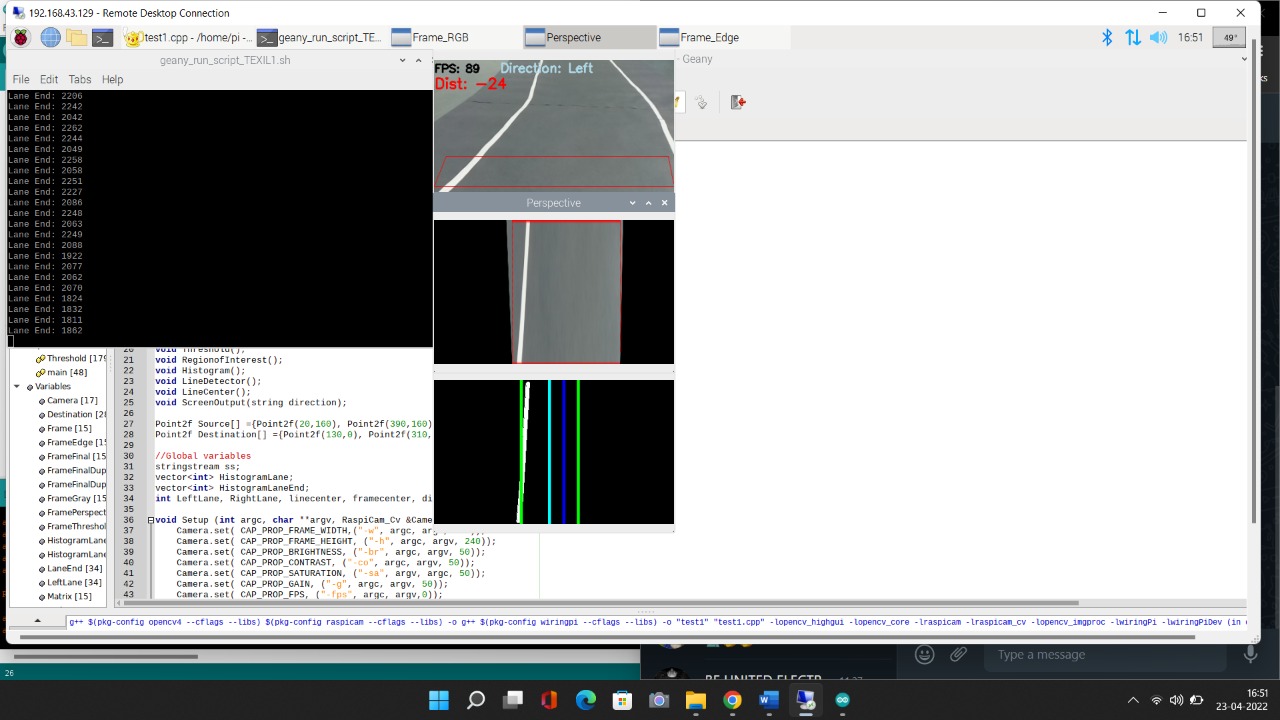


Figure 7.1.3 Left Direction

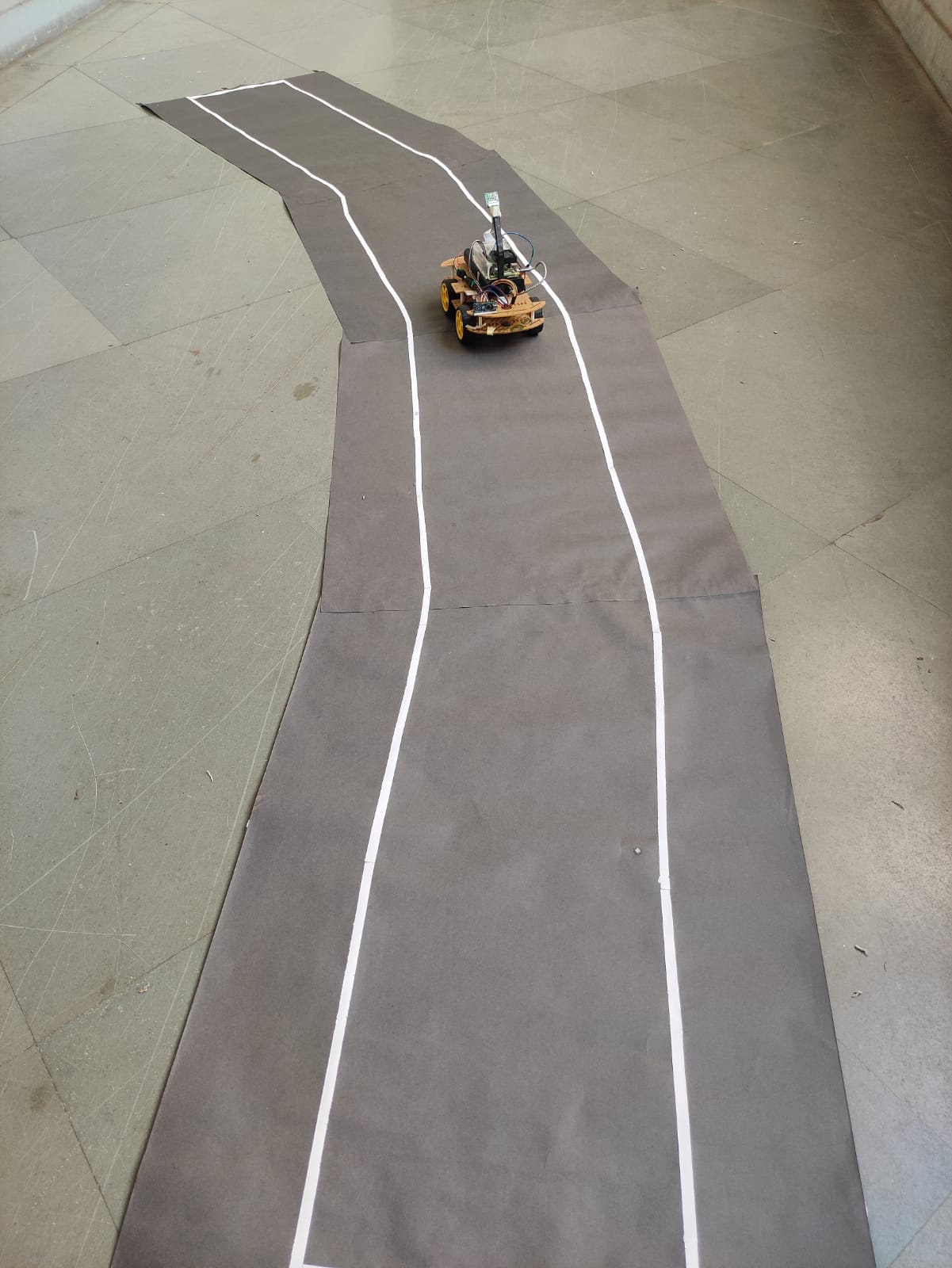


Figure 7.1.4 Left Direction

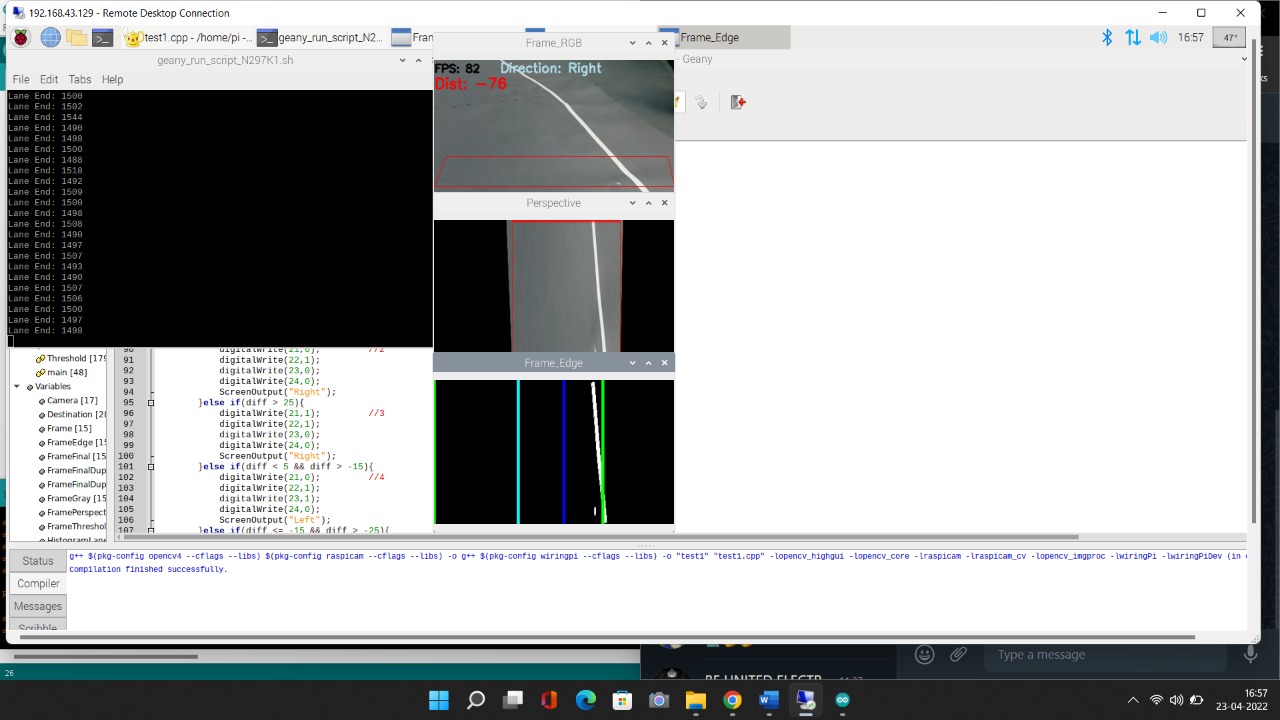


Figure 7.1.5 Right Direction

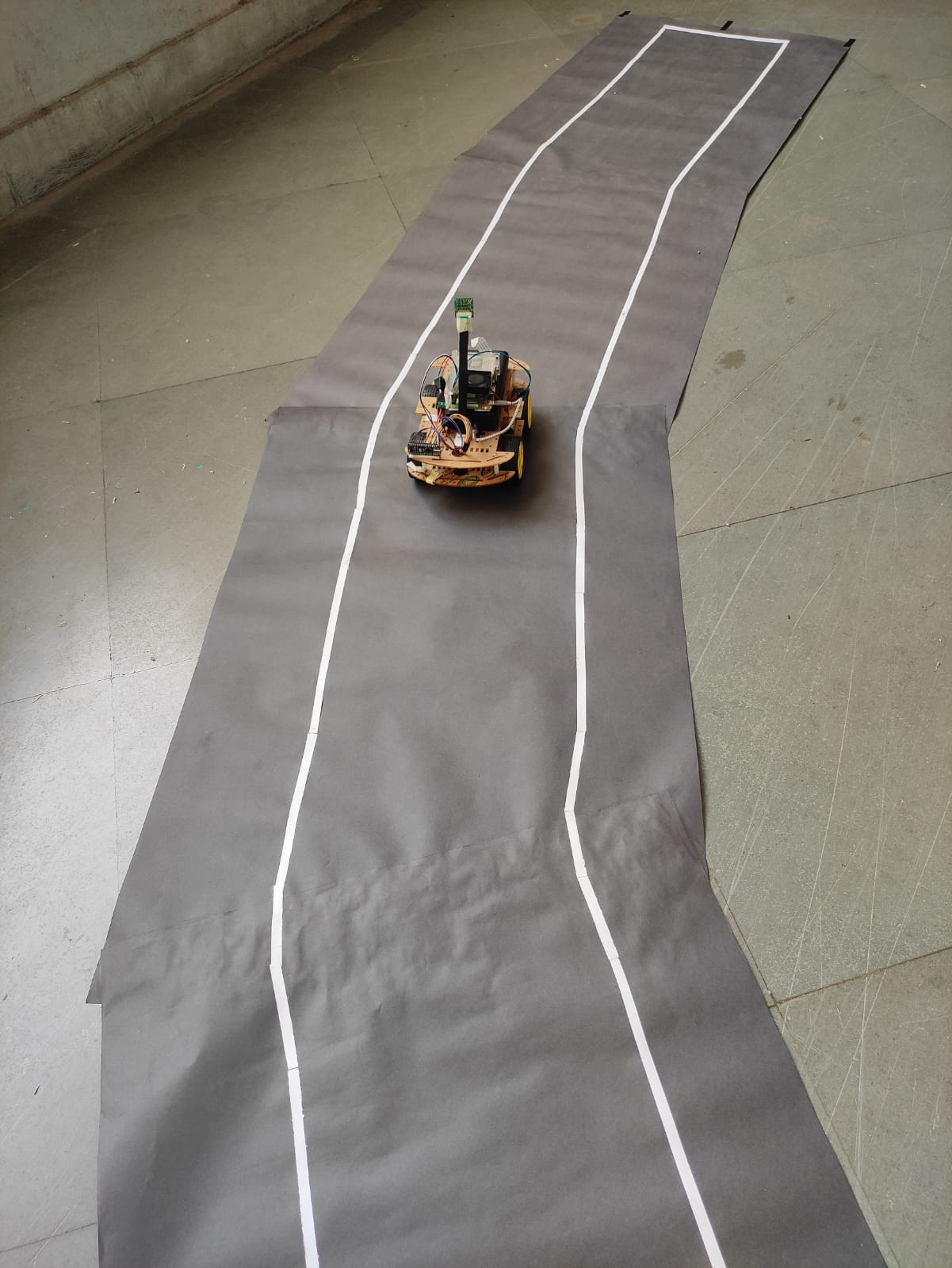


Figure 7.1.6 Right Direction

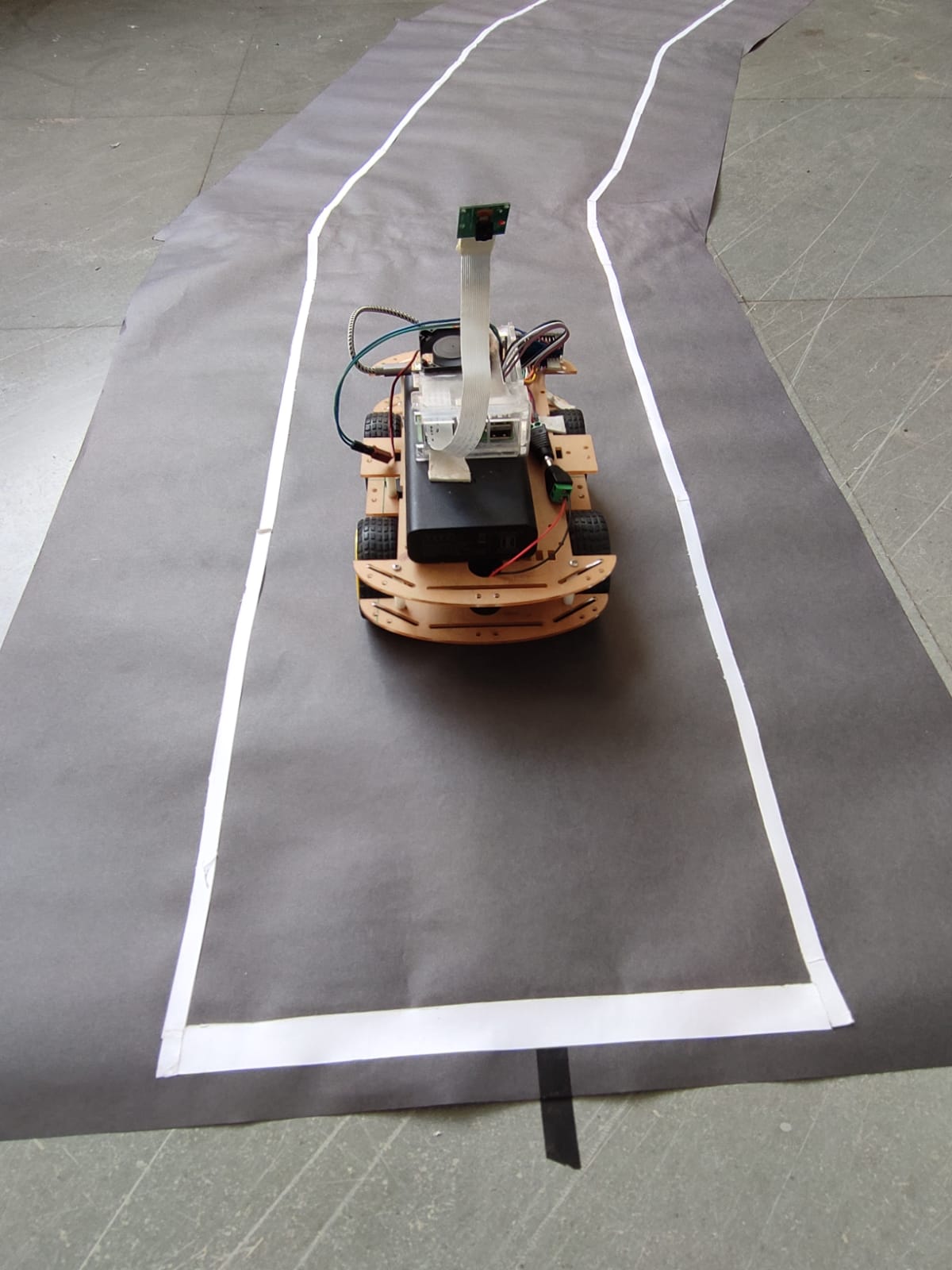


Figure 7.1.5 Lane End

## Conclusion

The Lane Following Robot is automobile system that has ability to recognize its path, move and change the robot's position toward the lane in the best way to remain in track. This project report presents a camera sensor-based lane following robot design which always directs along the white line on gray surface. The robot is able to detect its path in case it is out of path. The robot also detects obstacles coming between the path and responds accordingly. Thus the “LANE FOLLOWING MOBILE ROBOT” has been designed and tested successfully. The Lane following Mobile Robot project challenged the group to cooperate, communicate, and expand understanding of electronics, mechanical systems, and their integration with programming.