Self-Driving RC Car

Presented by Group AE-05

Pattarapon Buathong	(62070504012)
Sirakis Ngamwongwan	(62070504013)
Khun Angkharuengrattana	(62070504018)
Yiqi Chen	(62070504029)

Teacher Assistant

Chattriya Jariyavajee

Submitted on December 16th, 2019

as a part of the CPE101: Engineering Exploration course

Academic Year 2019

Summary

This project intends to make the RC toy car capable of driving by itself. Nowadays, self-driving car trends are flourishing with many famous companies, i.e., Tesla, which is our motivation for doing this project. Our expectation for the final product is the car could manage to drive in a lane and able to steer in both normal and sharp turns.

We chose the algorithm, after that, designed and created the project. Then, we built a mockup track and test the car. We encountered many problems such as algorithm error or circuit problems. After all the works, our final project which mainly depends on a camera, DC motors and a Raspberry Pi board could able to achieve our goal.

The outcome is decent. Yet, we planned to improve our final by using various methods such as add more ability to the car, or, improve the algorithm. Therefore, the car works correctly only in the prepared environment. If there are changes, we must configure the car to the new environment. Thus, it is better to use machine learning for the best outcome. However, with the success of this project, we concluded the project to be successful.

Table of Contents

Table of Figures

Introduction

Situation Analysis:

Nowadays, there is a lot of accident on the road. We could not argue that most of the accidents are caused by human rather than the car's mechanism. Many people got their driving license easily even though their skills are not great. Also, some people are either sleepy or drunk. In consequence, this could be dangerous to other people using the road.

It is difficult to improve a person's skill in driving. Thus, it will be fantastic if the car could be able to drive by itself. Either full-pilot or driving assistance. Many of the world's famous companies have developed their way to make the car self-driving. However, our motivation for doing this project is the "Tesla" company. Because they use cameras around their car to receive the input. After that, the input is processed in the car's processing units. Finally, the motors are being controlled, and, the car now drives by itself. We will mimic this idea by using one camera, one board, one motor driver, and two DC motors to make our RC car self-drives.

Statement of Problem

Only human skills to control a car may sometimes too low. Driving assistance may help a person to drive a car, or, take over the driver's seat. Yet, building the project in a real car is expensive and dangerous. Thus, we decided to use an RC car instead.

Main Purpose of Outcome (Primary Goal)

An RC car could be able to drive in a lane and steer by itself without human help.

Concept

- Pi Camera faces down to the ground about 30 degrees, and, receives the video stream input
- Raspberry Pi 3 board will process the input and determine whether it should steer left, right, or, go straight. Then, Raspberry Pi 3 board will send the pulse width modulation to L298N motor driver.
- L298N motor driver will control both DC motors to make the car drive according to Raspberry Pi 3's order.

Pre-Design Boundaries

1. Different light environment may affect the image processing algorithm. Since the color is very dependent to the light.
2. If the battery is low on power, it might affect both processing power and driving speed.
3. Poor internet connection will affect the remoting between Raspberry Pi 3 board and the laptop.

Background Knowledge and Related Work

Background Knowledge

Gaussian Blur

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function (named after mathematician and scientist Carl Friedrich Gauss). It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen, distinctly different from the bokeh effect produced by an out-of-focus lens or the shadow of an object under usual illumination. Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales—see scale space representation and scale space implementation.

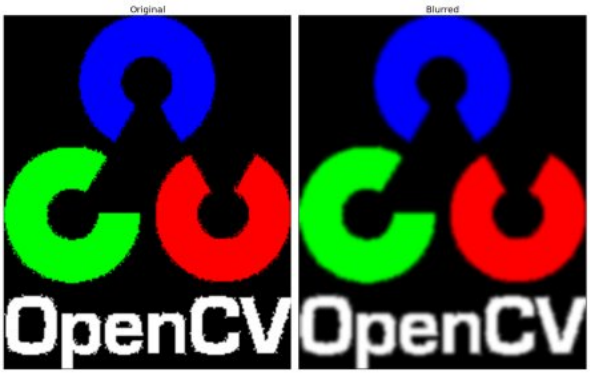

Figure 1 An image on the left is the original image. Gaussian blur of a kernel size of 5 is applied, as the right image shown.

HSV Color

The RGB (red, green, blue) color model is the most well-known way to mix and create colors. If you deal with commercial printers, you know about CMYK (cyan, magenta, yellow, key). You might have noticed HSV (hue, saturation, value) in the color picker of your graphics software. These are all schemes that describe the way colors combine to create the spectrum we see.

Unlike RGB and CMYK, which use primary colors, HSV is closer to how humans perceive color. It has three components: hue, saturation, and value. This color space describes colors (hue or tint) in terms of their shade (saturation or amount of gray) and their brightness value. Some color pickers, like the one in Adobe Photoshop, use the acronym HSB, which substitutes the term "brightness" for "value," but HSV and HSB refer to the same color model.

Hue is the color portion of the model, expressed as a number from 0 to 360 degrees. Saturation describes the amount of gray in a particular color, from 0 to 100 percent. Reducing this component toward zero introduces more gray and produces a faded effect. Sometimes, saturation appears as a range from just 0-1, where 0 is gray, and 1 is a primary color. Value works in conjunction with saturation and describes the brightness or intensity of the color, from 0-100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color.

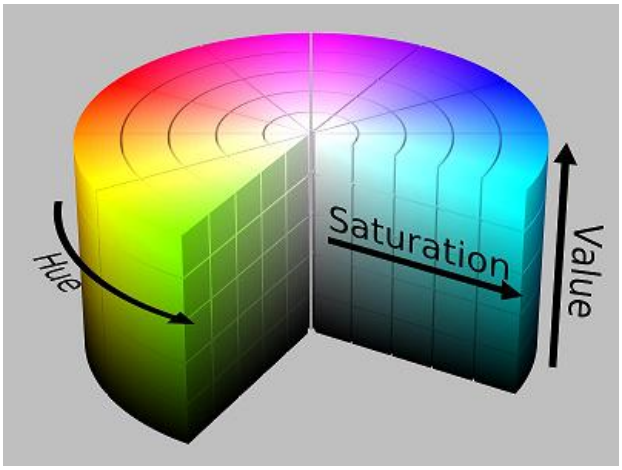

Figure 2 HSV color pie chart

Canny Edge Detection

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stages. Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. Yet, we already done this before. Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y) in equation (1). If we define **A** as the source image. From these two images, we can find edge gradient (2) and direction (3) for each pixel as follows.

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A} \tag{1}$$

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2} \tag{2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{3}$$

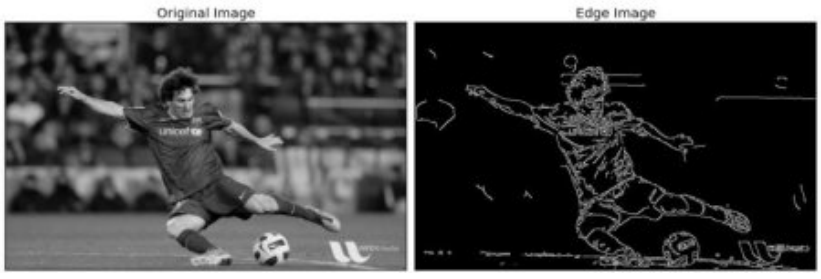

Figure 3 The left one is the original image. The right one is image after applying Canny edge detection.

Probabilistic Hough Transform

Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

In the Hough transform, you can see that even for a line with two arguments, it takes a lot of computation. Probabilistic Hough Transform is an optimization of Hough Transform we saw. It doesn't take all the points into consideration, instead take only a random subset of points and that is sufficient for line detection. We have to decrease the threshold. And, this method would return values of position in Cartesian coordinate (x, y). See below image which compare Hough Transform and Probabilistic Hough Transform in Hough space.

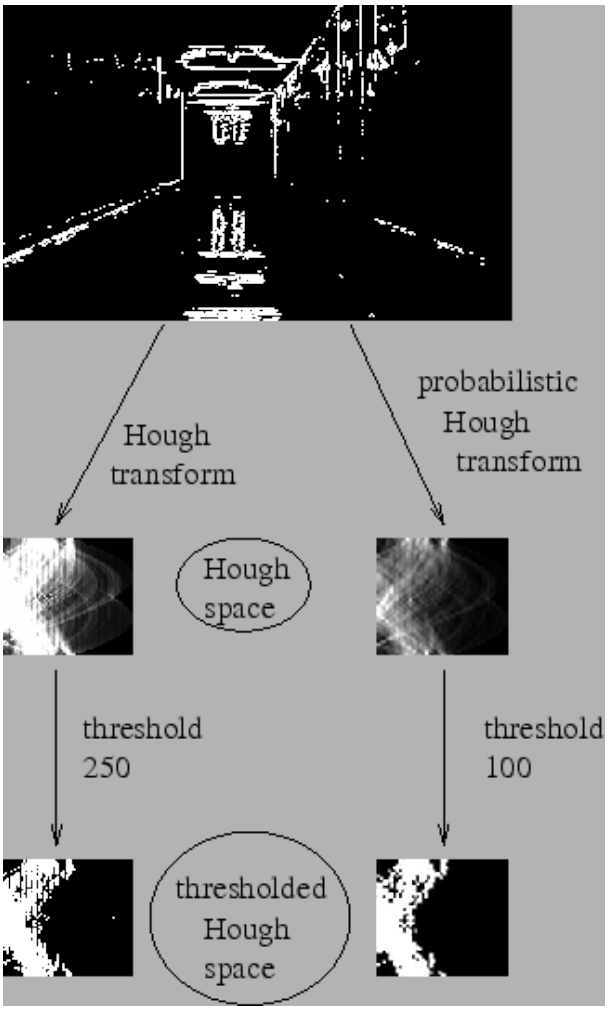

Figure 4 Comparison between ordinary Hough transform and probabilistic Hough transform.

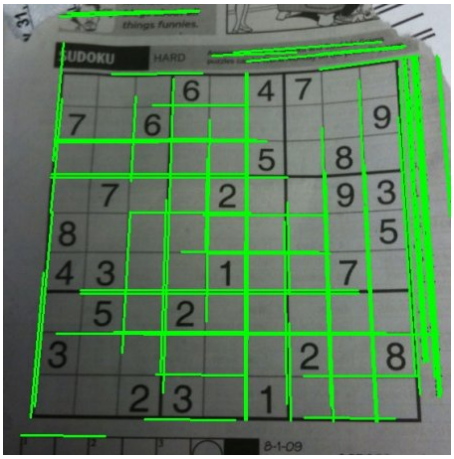

Figure 5 Example of probabilistic Hough transform applied to the image.

Atan2 function

Atan takes single argument and Atan2 equation (4) takes two arguments. The purpose of using two arguments instead of one is to gather information on the signs of the inputs in order to return the appropriate quadrant of the computed angle, which is not possible for the single-argument Atan. We use Atan2 function because it could convert Cartesian coordinate (x, y) into Polar coordinate (r, θ).

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases} \tag{4}$$

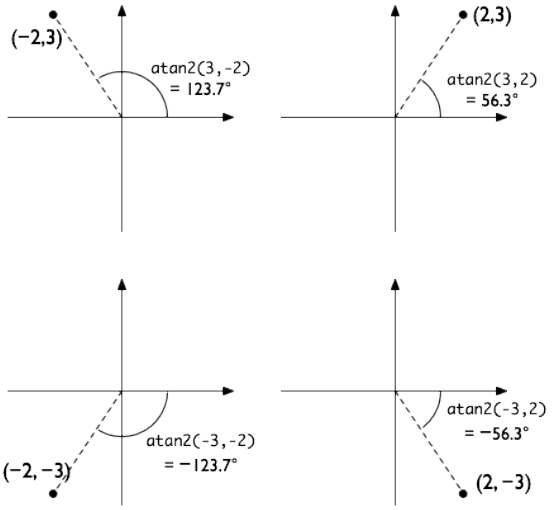

Figure 6 Graphs show angle calculated by using atan2 function.

Related Works

OpenCV Python Neural Network Autonomous RC Car

This project builds a self-driving RC car using Raspberry Pi, Arduino and open source software. Raspberry Pi collects inputs from a camera module and an ultrasonic sensor, and sends data to a computer wirelessly. The computer processes input images and sensor data for object detection (stop sign and traffic light) and collision avoidance respectively. A neural network model runs on computer and makes predictions for steering based on input images. Predictions are then sent to the Arduino for RC car control.

For the collection of input data, a Raspberry Pi board (model B+) with a pi camera module and an HC-SR04 ultrasonic sensor is used. To stream color video and ultrasonic sensor data to the device via local Wi-Fi link, two client programs run on Raspberry Pi. To achieve low latency transmission of content, video is scaled down to the resolution of QVGA (320 x240). The processing unit (computer) handles several tasks: receiving data from Raspberry Pi, training and prediction of the neural network, identification of objects (stop sign and traffic light), calculation of distance (monocular vision) and sending instructions to Arduino via USB connection. A multithread TCP server program runs on the computer to receive streamed image frames and ultrasonic data from the Raspberry Pi. Image frames are converted to gray scale and are decoded into numpy arrays.

One benefit of using neural network is that it only needs to load trained parameters afterwards once the network is trained, so prediction can be very fast. For training and prediction purposes, only the lower half of the input image is used. There are 38,400 (320×120) nodes in the input layer and 32 nodes in the hidden layer. The number of nodes in the hidden layer is chosen fairly arbitrary. In the output layer there are four nodes where each node corresponds to the instructions for steering control: left, right, forward and reverse respectively (although reverse is not used anywhere in this project, it is still included in the output layer).

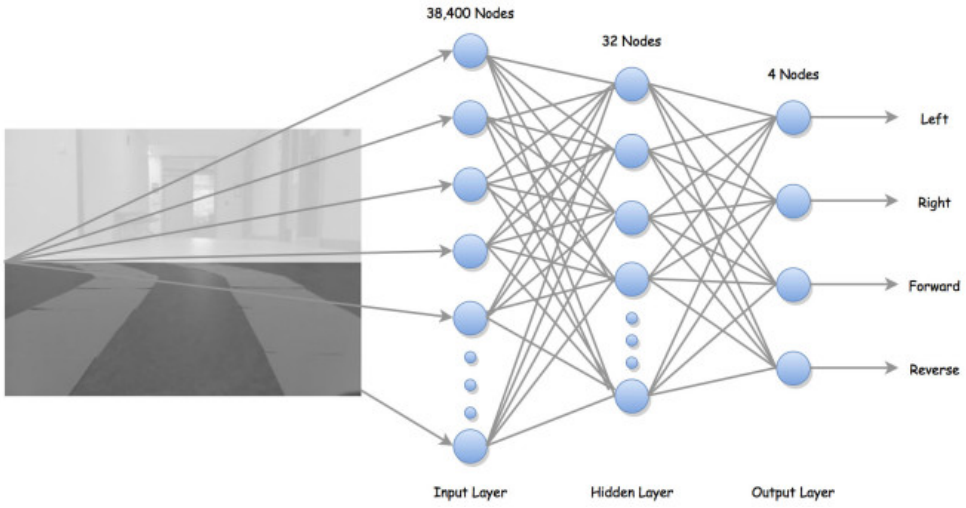

Figure 7 Layers in Neural Network

Below shows the training data collection process. First each frame is cropped and converted to a numpy array. Then the train image is paired with train label (human input). Finally, all paired image data and labels are saved into a npz file. The neural network is trained in OpenCV using back propagation method. Once training is done, weights are saved into a xml file. To generate predictions, the same neural network is constructed and loaded with the trained xml file.

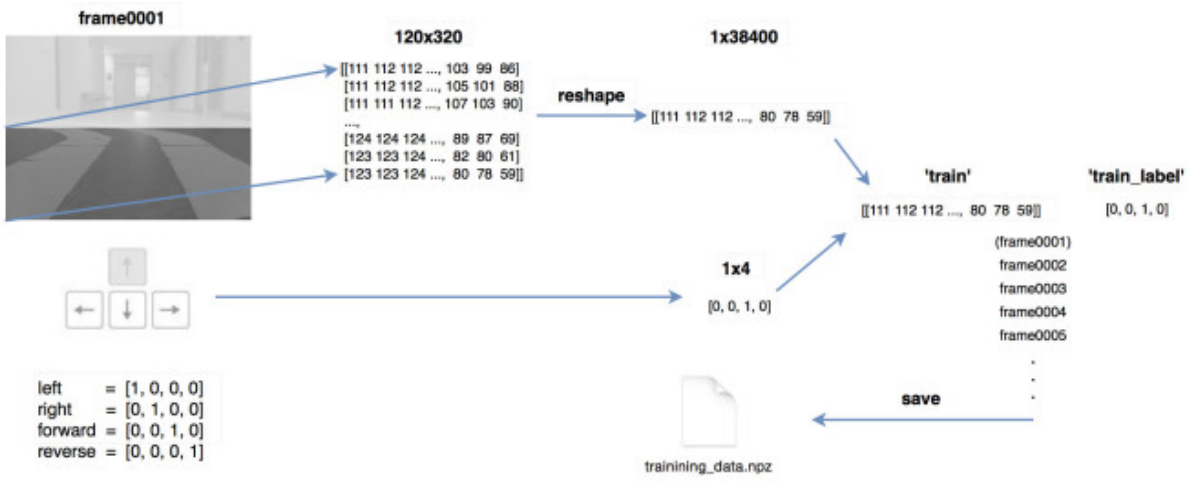

Figure 8 The training data collection process.

The RC car used in this project has an on/off switch type controller. When a button is pressed, the resistance between the relevant chip pin and ground is zero. Thus, an Arduino board is used to simulate button-press actions. Four Arduino pins are chosen to connect four chip pins on the controller, corresponding to forward, reverse, left and right actions respectively. Arduino pins sending LOW signal indicates grounding the chip pins of the controller; on the other hand sending HIGH signal indicates the resistance between chip pins and ground remain unchanged. The Arduino is connected to the computer via USB. The computer outputs commands to Arduino using serial interface, and then the Arduino reads the commands and writes out LOW or HIGH signals, simulating button-press actions to drive the RC car.

Road Lane Detection with Raspberry Pi

Self-driving cars are one of the new trends in the modern world. They use very sophisticated control systems and engineering techniques to maneuver the vehicle. Road lane detection is one of the important things in the vehicle navigation. Here I'm describing a simple and fast lane detection using Raspberry pi 3 and computer vision techniques. For fast calculation I just avoided linear regression method. This method gives the good result for low noise environment but for complex scenes advanced statistical and image processing techniques are needed.

First, receive a video input from Pi Camera every 0.1 second, and, store it in an array. After that, convert the image frame into grayscale and apply Gaussian blur in order to reduce the noise from camera captures. Next, apply edge detection (Canny) on the image. Following by, applying probabilistic Hough line transform on the edge we got. After applying, we should get an output of positions in the form of Cartesian coordinate. Thus, we could find the angle in

radius (like, slope). Finally, we compare the angle value with our defined threshold value to determine whether the car needs to turn left, right or go straight. This controls the servo that used to steer the car's wheels.

Figure 9 The program output in direction

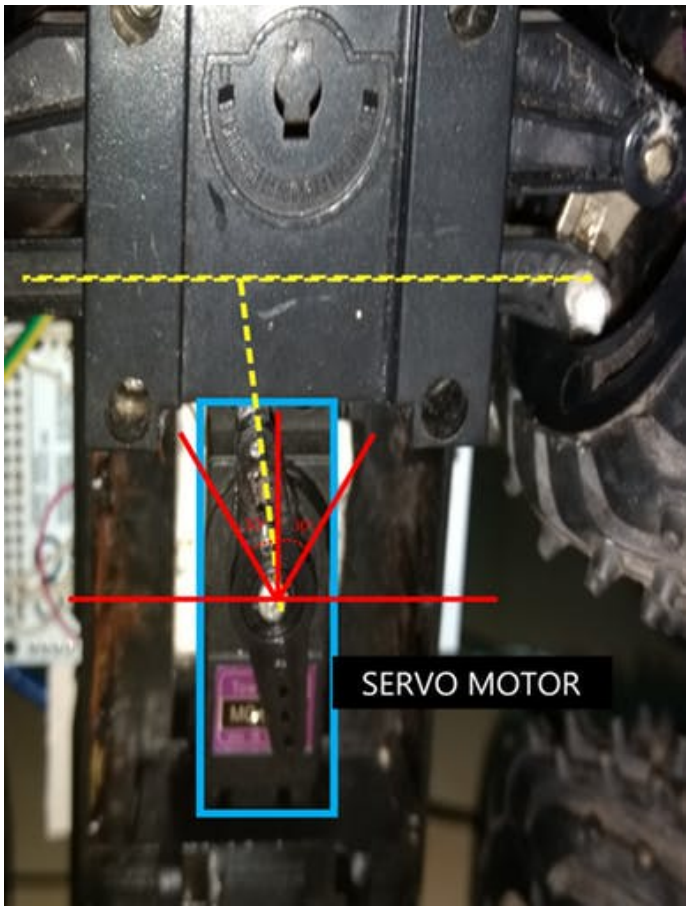

Figure 10 The servo that used to steer the wheels.

Tools and Methodology

Tasks and Schedule

Figure 11 Our Project Tasks and Schedule

List of Main Components

1. Camera: Pi Camera
2. Processing Board: Raspberry Pi 3 B+ board
3. Controlling Board: L298N motor driver
4. Driver: Two DC motors

Other equipment used: Batteries, Jump wires, Breadboards, 3D printed plastic case, Type-B charger, An old RC car toy, and duct tape.

Software/Package Used

- Python
    1. cv2 (OpenCV)
    2. numpy
    3. math
    4. picamera
    5. RPi.GPIO
- Raspberry Pi
    1. NOOBS
    2. Raspbian
    3. tightvncserver
    4. xrdp
- Laptop
    1. PuTTY
    2. Remote Desktop Connection

First Design

We tried an algorithm that connects wirelessly between the camera and the laptop. The camera receives video input and sends it to the laptop. Then, the laptop processes the video using a neural network that depends on a prepared dataset. After that, the laptop sends some value to Arduino Uno and it will control the wireless remote to drives an RC car.

Problems

1. This algorithm covers the topic “AI/Machine Learning” not the “Image Processing” topic that we chose.
2. Our components were not enough. Because the remote controller was lost.
3. It would have a delay connection if performance in a poor condition.

Second Design

The first design algorithm was too advance, yet, not satisfies our project's topic. We decided to use the Raspberry Pi 3 to process the video input from the camera. Then, send a logic to Arduino Uno and let it control the L298N motor driver. Lastly, L298N motor driver will control both DC motors to make the car self-drive.

Problems

1. We could not communicate between Raspberry Pi 3 board and Arduino Uno.
2. Need more power source for Arduino Uno board.
3. The jump wires look very messy.

Algorithm Overview

1. Pi Camera captures video input.
2. Raspberry Pi 3 board receives video input from Pi Camera. After that, image processing algorithm would be done on the video input. Therefore, it will send PWM to L298N motor driver.
3. The L298N motor driver will control both DC motors according to the value Raspberry Pi 3 had sent.

(See Flowchart)

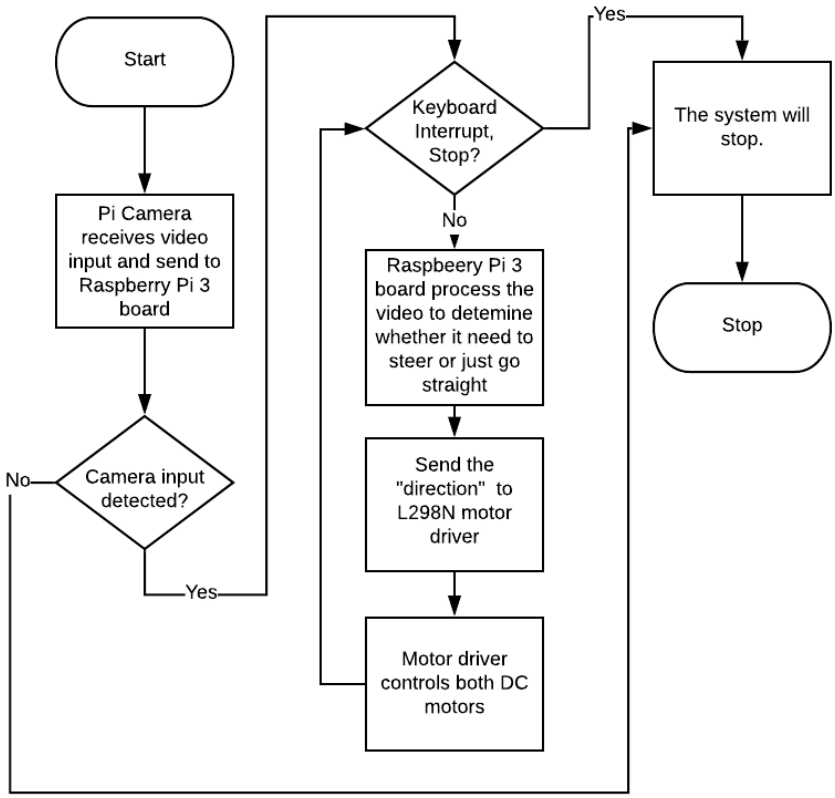

Figure 12 The flowchart describing algorithm overview.

System Overview

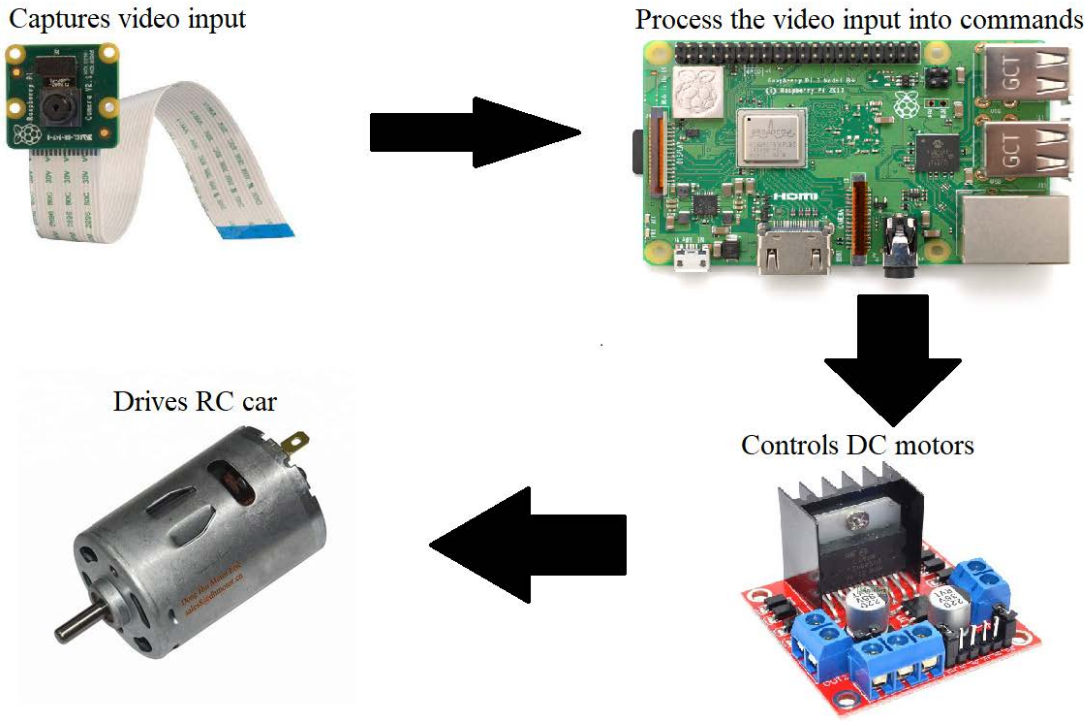

Figure 13 The workflow of the components in our project.

Algorithm in Detail

1. The Pi Camera captures video. Then, we rotate the video by 180 degrees. We have to rotate the video input because of the camera place on our car.
2. Apply Gaussian blur to the arrays that contain each video frame in order to reduce the noise slightly.
3. Since our lane is red colored. Thus, we apply the red color range into the array, and, filter it to detect only red.
4. Edge detection is applied in the frame array.
5. Probabilistic Hough transform is applied on the edge we detected. This process output a position of the line in Cartesian coordinate system.
6. The position x1, y1, x2, y2 we got from probabilistic Hough transform are used to calculate the angle using atan2 function.
7. Compare the value we got from 6) with our pre-defined threshold. For example, four. Then, determine whether we should turn left, turn right, or, go straight.
8. If we need to turn left, for example. Raspberry Pi board will send a pulse width modulation control to L298N motor driver. With the command that reduce duty cycles on the left DC motor. Hence, the car will turn left.

Driving and Steering Threshold

We had specific the threshold that used to determine whether the car need to turn left, turn right, or, go straight. This depends on the angle outputted from atan2 function. For example, angle should not exceed 30 degrees when the car is going forward. The below figure shows how we set the threshold.

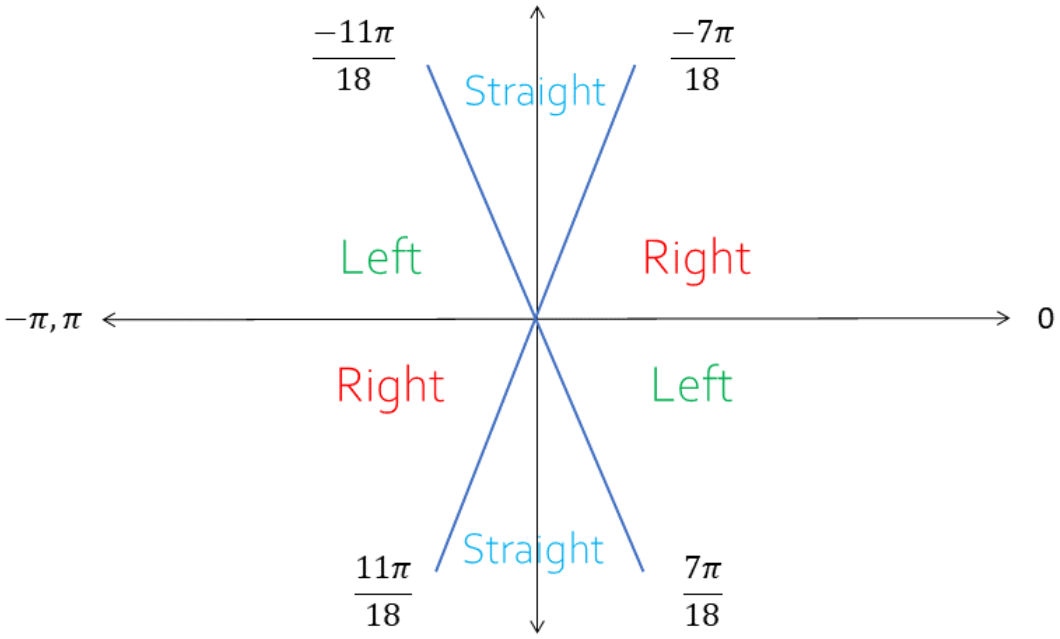

Figure 14 Threshold that detemines steering or go straight

From this figure, you could see that the values vary from $-\pi$ to π. This is the reason that why we are using atan2 function, instead of using the ordinary atan function.

Controlling DC motors

First, we need to find and connect the pulse width modulation (PWM) GPIO pin of the Raspberry Pi board into L298N motor driver. The following image shows all the pins which are available to be configured and used as PWM pins.

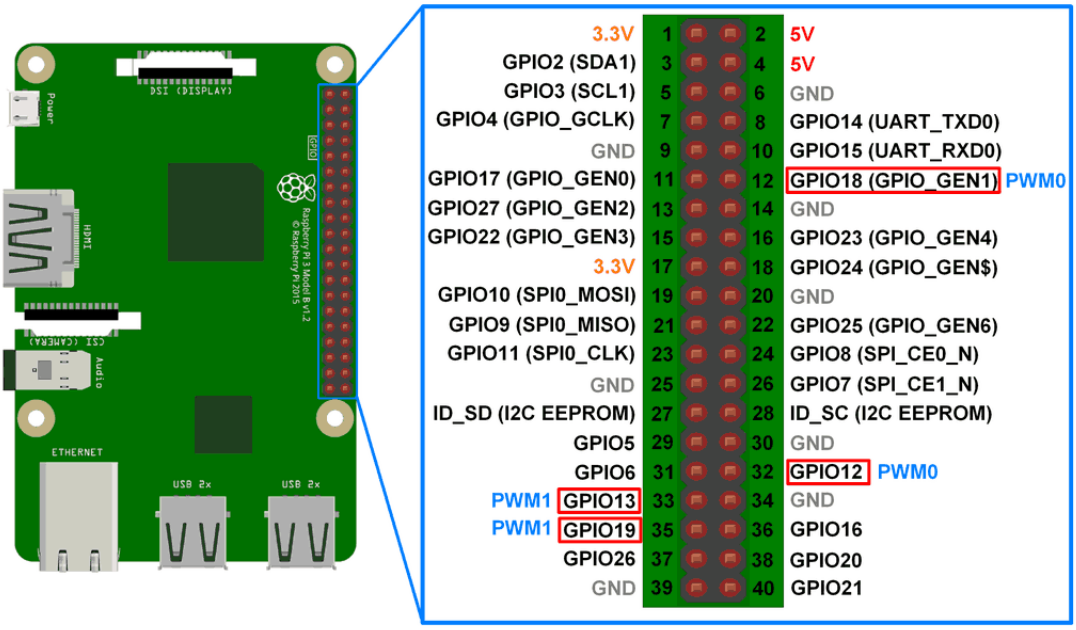

Figure 15 GPIO pins that able to control PWM.

In order to control the speed of the DC motors. We control the duty cycles, or it could be said that, this is a pulse width modulation. Duty cycle refers to the ratio of on time of the signal to the total time period of the signal. As with any signal, we first define a base frequency of the signal. On this signal, the time for which it is high and low is modulated to change the duty cycle. The figure shown below are three different signals with duty cycles of 75%, 50% and 25%. The value of duty cycle of a PWM signal decides the equivalent DC level of the signal.

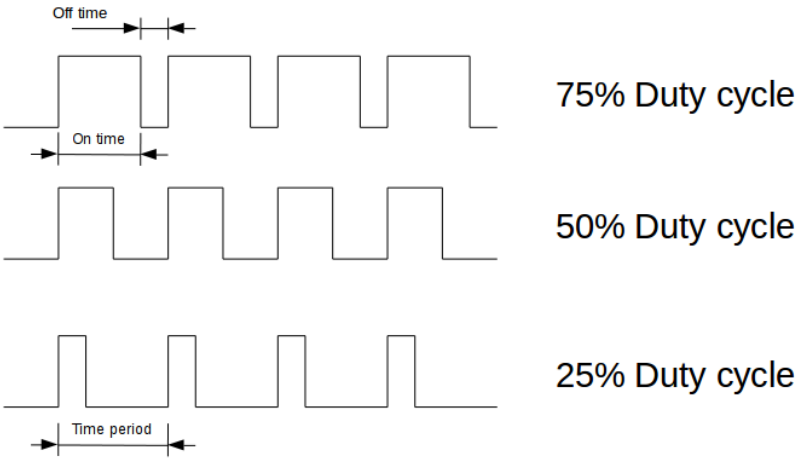

Figure 16 PWM signal according to percentage of duty cycle.

Circuit Design

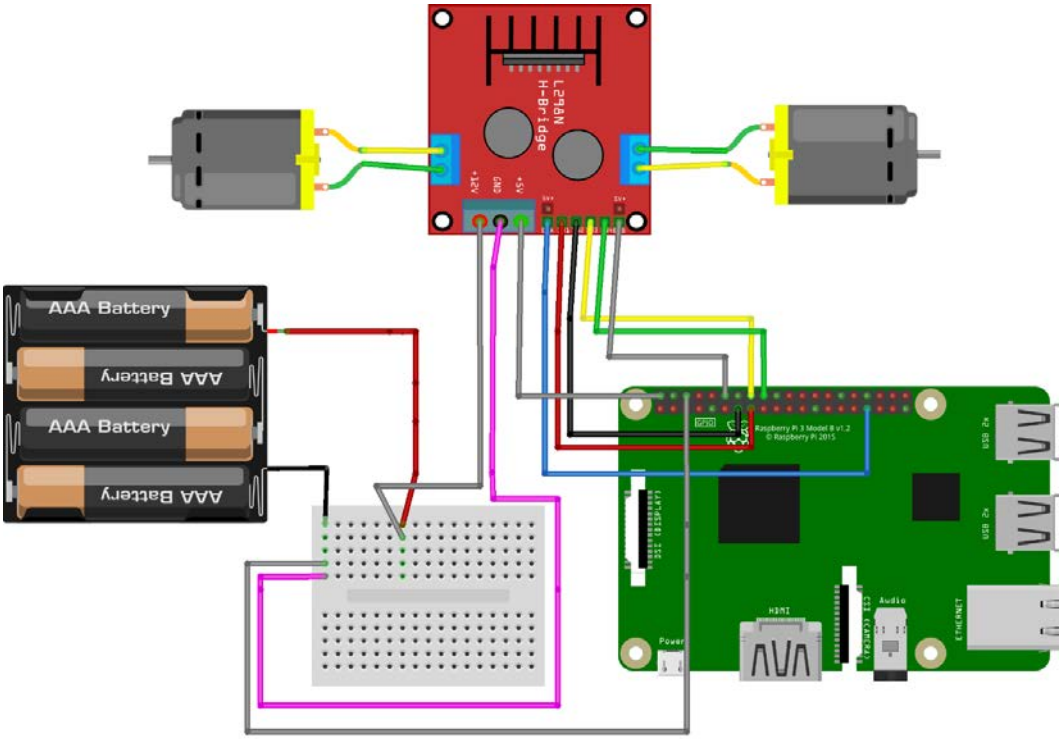

Figure 17 Our project circuit chart.

Prototype Testing

We ran various test before releasing the final design.

Algorithm Test

Problem 1: The framerates per second that Raspberry Pi 3 board could process is too low. In consequence, this would affect the car driving because it could not handle the input and process it fast enough.

Correction: We improved the algorithm by making the code run faster. For example, if we reserve the array for red color every time it loops. It would make the program slow down, or, editing the nested loop. Nested loop could affect the processing performance much, according to big-O-notation. Difference between an ordinary loop and a nested loop is high.

Problem 2: Sometimes, when the car approaches the sharp turn. It could not detect that they should turn left or turn right. Instead, they decided to output straight and go towards the curve.

Correction: This problem could be solved by having many ranges of threshold. Since the angles we got from the function atan2 are between $-\pi$ to π.

Problem 3: The car could not able to handle very sharp turn such as 90 degrees turn.

Correction: With the solution from problem 2. This problem could be solved.

Results and Discussion

Results

Final Design

From the second design, we solve the problem by not using Arduino Uno board anymore. Thus, we directly connect Raspberry Pi 3 board to L298N motor driver. Using duty cycles to control pulse width modulation, it could communicate via PWM pin on Raspberry Pi 3 board.

Features

1. Raspberry Pi 3 board would detect the lane and send the commands to L298N motor driver directly. Thus, the delay is very low and only depends on the algorithm.
2. This method satisfies the project's topic. Because, it use a lot of fundamental knowledge in image processing.

After all the commands applied on the input video, or, a frame of image in the video. Now, the car could be able to drives and steer in the lane. The figure below shows how the car sees the lane, and able to know which direction it needs to go.

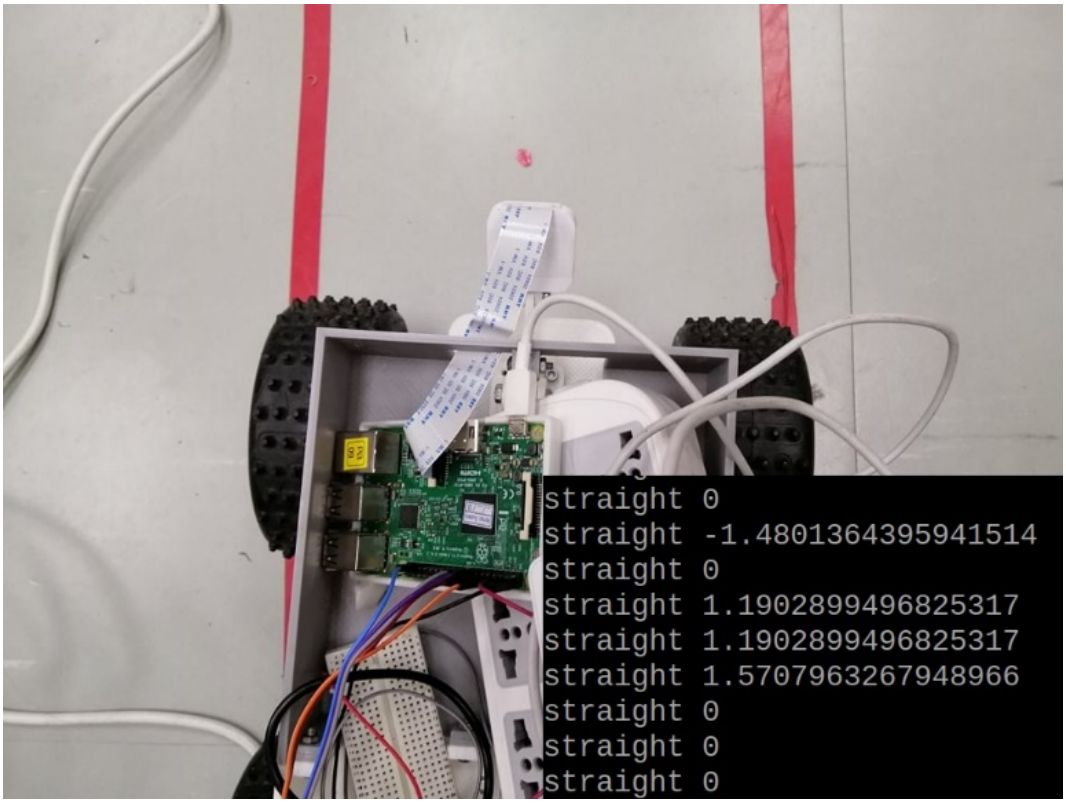

Figure 18 Go straight

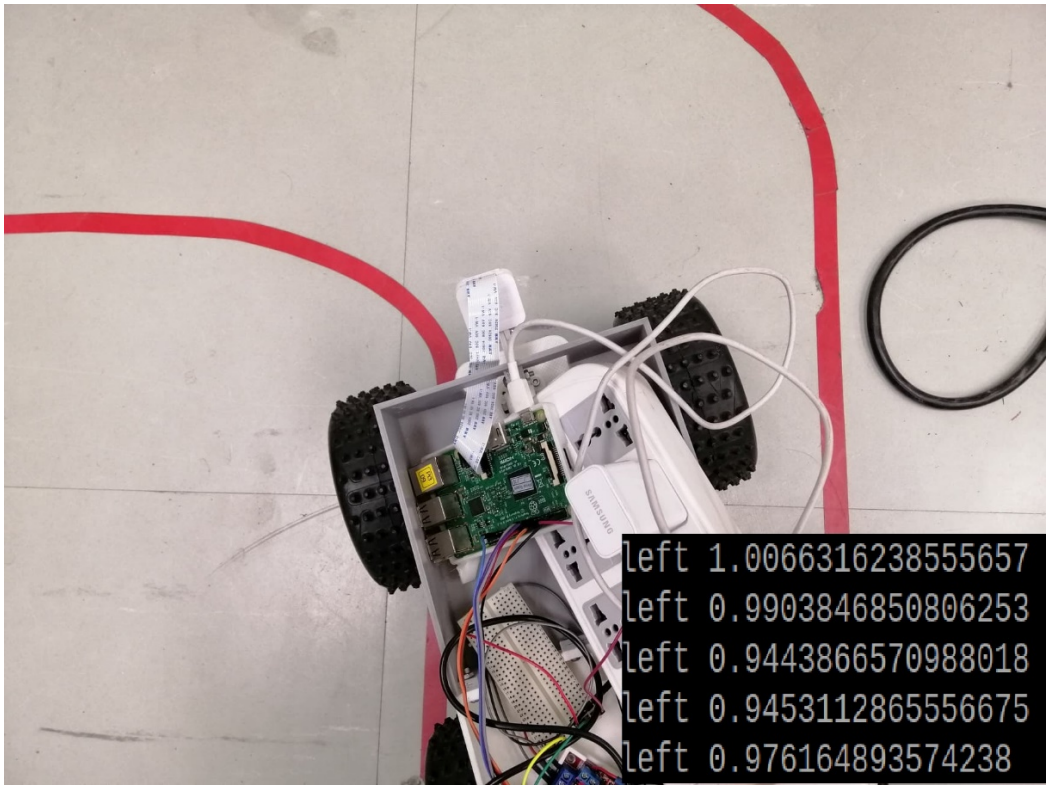

Figure 19 Turn left

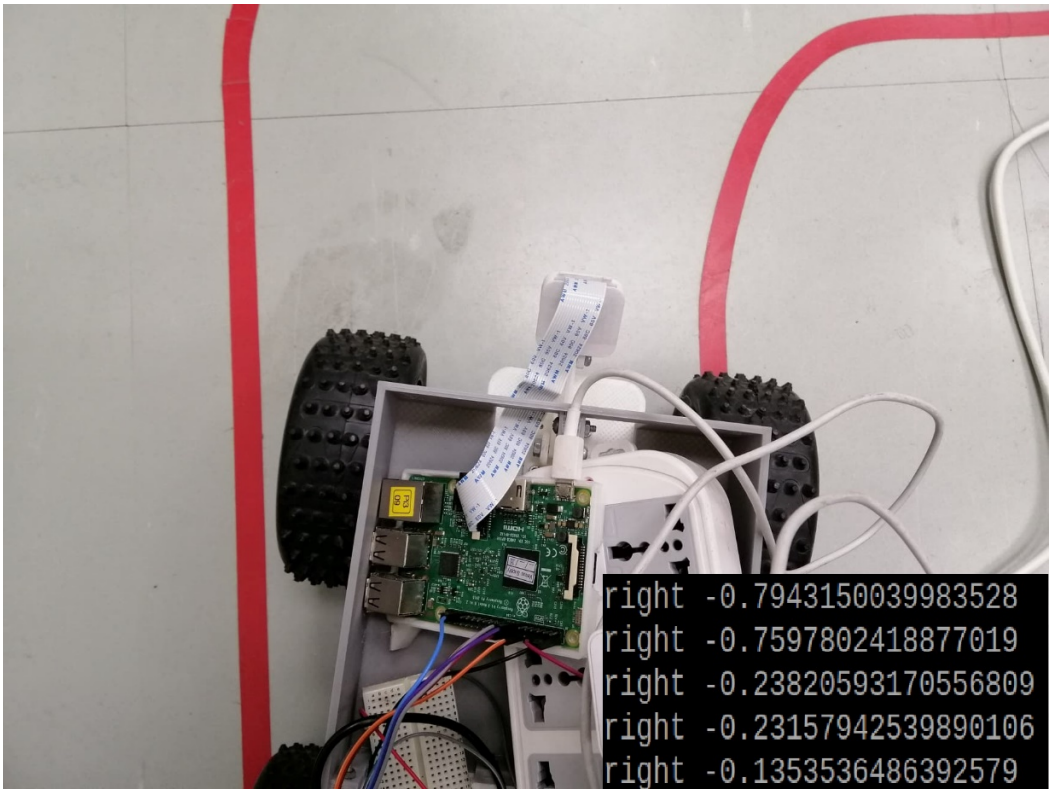

Figure 20 Turn right

Figure 21 Final Design

Discussion

1. In the difference light environment, the color the camera see would change slightly, or, drastically. Thus, we need to specify the red color range as an array for each new light environment.

2. The project is possible to improve more than just driving in a lane and steering. For example, if it saw the stop sign, it will eventually stop 5 seconds.

3. The car is dependent on starting position. If the starting position is bad, sometimes the car will drive out of the lane.

Conclusion

Currently, there is a skyrocket trend in self-driving car. This is due to the fact that, many accidents on the road are caused by human careless when driving a car. Either full self-driving car or driving assistance could help human drives. Yet, using a real car for our project is too expensive and dangerous. Thus, we decided to use an old RC car toy to be our model. We use the concept of image processing instead of IR sensor, to implement the way a camera could see in the real traffic.

We use Pi Camera for capturing video as an input, being reversed, and, store it frame by frame in an array. After that, Gaussian blur, color detection, Canny edge detection, and, probabilistic Hough transform is applied respectively. We could use the output from probabilistic Hough transform, position in Cartesian coordinate, and transfer the values into polar form, to determine whether we need to turn left, turn right, or, go straight. These values are going to get compared with our pre-defined threshold. Then, Raspberry Pi board would send PWM to L298N motor driver to control both of the DC motors. Combining all of the works, the car is now able to driving and steering in the lane. It will keep driving under the keyboard got interrupted. The program would terminate and stop. Lastly, we should stimulate the awareness of using a computer-assisted driving car. Because it could make commuting around more safely.

Appendix

The code below is our algorithm for self-driving car.

```
import cv2
import numpy as np
import math
import time
from picamera.array import PiRGBArray
import RPi.GPIO as GPIO
from picamera import PiCamera       # All necessary package used for detecting lane

def nothing(x):                     # if nothing
    pass                            # continue the program

camera = PiCamera()                 # starts Pi Camera
rawCapture = PiRGBArray(camera)     # captures video in RGB
camera.rotation = 180               # rotates camera 180 degrees

in1 = 23                            # GPIO pin23 = in1 in L298N
in2 = 24                            # GPIO pin24 = in2 in L298N
ena = 18                            # GPIO pin18 = ena in L298N
in3 = 22                            # GPIO pin22 = in3 in L298N
in4 = 27                            # GPIO pin27 = in4 in L298N
enb = 13                            # GPIO pin13 = enb in L298N

GPIO.setmode(GPIO.BCM)              # mode set to use the "GPIO PIN NUMBER"
GPIO.setup(in1,GPIO.OUT)            # set in1 to be output
GPIO.setup(in2,GPIO.OUT)            # set in2 to be output
GPIO.setup(ena,GPIO.OUT)            # set ena to be output
GPIO.setup(in3,GPIO.OUT)            # set in3 to be output
GPIO.setup(in4,GPIO.OUT)            # set in4 to be output
```

```
GPIO.setup(enb,GPIO.OUT)            # set enb to be output

p1=GPIO.PWM(ena,1000)           # set p1 to be pulse width mod. of ena, frequency = 1000
p2=GPIO.PWM(enb,1000)           # set p2 to be pulse width mod. of enb, frequency = 1000
p1.start(5)                     # starts p1 with duty cycle of 5
p2.start(5)                     # starts p2 with duty cycle of 5

def forward():                      # a function for drive the motor forward
    p1.ChangeDutyCycle(14)          # change duty cycle to 12 for p1
    p2.ChangeDutyCycle(14)          # change duty cycle to 12 for p2
    GPIO.output(in1,GPIO.LOW)       # stop in1
    GPIO.output(in2,GPIO.HIGH)      # start in2
    GPIO.output(in3,GPIO.LOW)       # stop in3
    GPIO.output(in4,GPIO.HIGH)      # start in4

def left():                         # a function to turn left
    p1.ChangeDutyCycle(48)          # change duty cycle to 40 for p1
    p2.ChangeDutyCycle(21)          # change duty cycle to 20 for p2
    GPIO.output(in1,GPIO.LOW)       # stop in1
    GPIO.output(in2,GPIO.HIGH)      # start in2
    GPIO.output(in3,GPIO.HIGH)      # start in3
    GPIO.output(in4,GPIO.LOW)       # stop in4

def right():                        # a function to turn right
    p1.ChangeDutyCycle(21)          # change duty cycle to 20 for p1
    p2.ChangeDutyCycle(48)          # change duty cycle to 40 for p2
    GPIO.output(in1,GPIO.HIGH)      # start in1
    GPIO.output(in2,GPIO.LOW)       # stop in2
    GPIO.output(in3,GPIO.LOW)       # stop in3
    GPIO.output(in4,GPIO.HIGH)      # start in4
```

```
theta=0                             # initial theta (angle) value = 0
minLineLength = 10                  # max line length in HoughLinesP
maxLineGap = 10                     # max line gap in HoughLinesP

lower_red = np.array([150,70,50])       # array range of red color
upper_red = np.array([180,255,255])     # array range of red color

try:                                    # force to do
    Size_crop = [0,0]                   # crop size to default
    # enter the loop if camera detected
    for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image2 = frame.array                                    # image2 store image as frame get stored in an array
        image = cv2.GaussianBlur(image2, (5, 5), 0)             # apply a gaussian blur (kernel size of 5x5) to reduce noise
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)            # red color detection
        mask = cv2.inRange(hsv, lower_red, upper_red)           # mask the image frame
        res = cv2.bitwise_and(image,image, mask=mask)           # filter the frame to show only red color
        edged = cv2.Canny(res, 85, 85)                          # apply Canny edge detection on the frame
        # apply probabilistic Hough transform on the frame
        lines = cv2.HoughLinesP(edged,1,np.pi/180,10,minLineLength,maxLineGap)
        rawCapture.truncate(0)                                  # truncate captured frame
        if(lines is not None):                                  # check if the line is not none value
            for line in lines:                                  # loop in lines
                for x1,y1,x2,y2 in line:                        # use the Cartesian coordinate fromm lines
                    theta=theta+math.atan2((y2-y1),(x2-x1)) # find the angle using function atan2
            theta /= len(lines)                                 # theta divided by length of lines

        if(-3.142<theta<-1.920 or 0<theta<1.222):               # define length for turning left
            left()                                              # turn left
            print("left", theta)                                # print left + value of theta on the monitor

        elif(1.920<theta<3.142 or -1.222<theta<0):              # define length for turning right
```

```
            right()                                 # turn right
            print("right", theta)                   # print right + value of theta on the monitor

        else:                                       # else, go straight
            forward()                               # go straight
            print("straight", theta)                # print straight + value of theta on the monitor

        theta = 0                                   # reset the theta value to 0
        cv2.imshow("Res",res)                       # show the stream video "res"
        k = cv2.waitKey(5) & 0xFF                   # k = looks for wait key
        if k == 27:                                 # if key press
            break                                   # break the loop
    cv2.destroyAllWindows()                         # close all of the windows shown
    cap.release()                                   # stop capturing video

except KeyboardInterrupt:                           # unless keyboard interrupt
    GPIO.cleanup()                                  # cleanup the GPIO pin
```

Bibliography

Abhi, A. (2018, April 20). *Road Lane Detection with Raspberry Pi*. Retrieved from https://create.arduino.cc/projecthub/Abhinav_Abhi/road-lane-detection-with-raspberry-pi-a4711f?use_route=project&fbclid=IwAR3ra8AIqmJC879BDhyQ3_EsTNnNkzEBCodEmxewl5dcrY0wYftqKo80I7Q

Alexander M. & Abid K. R. (2013). *Canny Edge Detection*. Retrieved from https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html#canny

Alexander M. & Abid K. R. (2013). *Hough Line Transform*. Retrieved from https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

Ashish. (2018, September 22). *Understanding Edge Detection (Sobel Operator)*. Retrieved from https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900

Bear, J. H. (2019, November 11). *The HSV Color Model in Graphic Design*. Retrieved from https://www.lifewire.com/what-is-hsv-in-design-1078068

croston. (2019). *Using PWM in RPi.GPIO*. Retrieved from https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/

Fernando, S. (2018). *Gaussian Blur*. Retrieved from https://www.opencv-srf.com/2018/03/gaussian-blur.html

hamuchiwa. (2019, May 3). *OpenCV Python Neural Network Autonomous RC Car*. Retrieved from https://github.com/hamuchiwa/AutoRCCar?fbclid=IwAR1v0nVlh9TsTDXEMI3waFPR6X_nzpBR1psgCzjUONXkM-0vctSrkHzuSoA

ndrplz. (2017, November 28). *Udacity Self-Driving Car Engineer Nanodegree projects.* Retrieved from https://github.com/ndrplz/self-driving-car?fbclid=IwAR1aGaHX_l6k2frUMjXIDQv22rAWmnhe8RpeVqA0iRKEXAElv1l_Co78xlI

Rauf, A. (2016, March 2). *Python atan or atan2, what should I use?* Retrieved from https://stackoverflow.com/questions/35749246/python-atan-or-atan2-what-should-i-use?fbclid=IwAR26hWN69c4SjVEsgKbDWfIMtHm48NHXum-JN5XYhHYUEA5uI5wEzm7Xakg

Robocraze. (2018, November 30). *RASPBERRY PI: CONTROLLING SPEED AND LED BRIGHTNESS*. Retrieved from https://www.robocraze.com/blog/raspberry-pi-controlling-speed-and-led-brightness.html

Rosebrock, A. (2014, August 4). *OpenCV and Python Color Detection*. Retrieved from https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/?fbclid=IwAR2E1QcI5s-lvPXQ0jP_6QFB0jmNyNRB1o5cwYmrGEGbCqqNvFHSg2lRlwU

Wang, Z. (2018). *Self Driving RC Car*. Retrieved from https://zhengludwig.wordpress.com/projects/self-driving-rc-car/?fbclid=IwAR2cuh2bbVrRyDuKh7yNdGl3FSWNtzvCtXRr7lN3i6_iP1o8YI6ZhdMB0kQ