

HELWAN UNIVERSITY

Faculty of Engineering

Communications & Information Engineering

Graduation project Title: Self-Driving Car

By:

Mohamed Osama Mohamed Ismail

Aya Mohamed Mustafa

Khaled Mustafa Murad

Sheriff Mohamed Morsy

Sara Shehata Thabet

Sara Hesham Mostafa

Supervised by: Dr Zaki Bassiony

ABSTRACT

The autonomous car or the driverless car can be referred to as a robotic car in simple language. This car is capable of sensing the environment, navigating and fulfilling the human transportation capabilities without any human input. It is a big step in the future technology. Autonomous cars sense their surroundings with cameras, radar, lidar, GPS and navigational paths. Advanced control systems interpret sensory information to keep track of their position even though the conditions change. The advantages of autonomous cars, such as fewer traffic collisions, increased reliability, increased roadway capacity, reduced traffic congestion as well as reduction of traffic police and care insurance, are compulsive for the development of autonomous car even though we have to overcome the issues of cyber security, software reliability, liability of damage and loss of driver related jobs. Autonomous cruise control or the Lane departure warning system and the Anti lock braking system (ABS) are the early steps. These steps though small are conclusive towards the progress in the direction of making the autonomous car. Companies such as Google, Volvo, Mercedes-Benz and Audi are the fore runners in making the autonomous car a reality. The development and expansion of the sector in Indian conditions is also worth considering. We strongly believe that the autonomous car will be a reality soon and be a necessity of life by overcoming the current obstacles, as human life needs to be secured by safe, efficient, cost effective and comfortable means of transport. The development of our prototype was completed through the use of two controllers; Raspberry pi and Arduino. The main parts of our model include the Raspberry pi, Arduino controller board, motors

Table of content

1. Introduction	1
1.1 History	2
1.2 Why autonomous car is important	4
1.2.1 Benefits of self-driving cars	4
1.3 What are autonomous and automated vehicles	5
1.4 Project description.....	7
1.4.1 image processing	7
1.4.2 Lane detection	7
1.4.3 Obstacle detection	8
1.4.4 traffic signs.....	8
2. Car Design	12
2.1 First car design	13
2.1.1 Four WD robot	13
2.1.1.1 specification	13
2.1.1.2 Real shots	14
2.1.1.3 Drawbacks.....	14
2.1.1.4 Overcoming Drawbacks.....	14
2.2 Motors	24
2.2.1 DC motor.....	25
2.2.1.1 DC motor fundamentals	25
2.2.1.2 DC motor principle	25
2.2.1.3 How DC Motors Work.....	31
2.3 Motor and its connection with the Arduino	32
2.3.1 H-Bridge.....	32
2.3.2 Features	34
2.4 Driver connection with Arduino	35
2.4.1 Left & Right Functions for Motors	37

2.4.1.1 Implementation & testing.....	37
2.4.1 Forward & Backward Functions for Motors	38
2.4.1.1 Implementation & testing	39

3. Microcontrollers and Hardware40

3.1 Microcontrollers	41
3.1.1 Arduino	42
3.1.1.1 Introduction	42
3.1.1.2 Arduino Uno	43
3.1.2 Raspberry pi	43
3.1.2.1 Design	44
3.1.2.2 ARM 1176 processor	44
3.1.2.3 Features	45
3.1.2.4 Performance	45
3.1.2.5 Function supported in Hardware.....	46
3.1.2.6 Comparison	47

4. Lane detection 54

4.1 Introduction	55
4.2 History.....	56
4.3 Limitations of Lane keeping Systems in real cars.....	56
4.4 Prototyping of Lane-detection.....	56
4.4.1 Basic components.....	57
4.4.2 Image Processing Techniques.....	57
4.4.3 Finding Region of Interest (ROI).....	57
4.4.3.1 implementation & testing.....	58
4.4.4 Canny edge detection.....	59
4.4.4.1 IMPLEMENTATION.....	60
4.4.5 lane end & U turn.....	60
4.5 final testing	61

5 Stop sign detection.....	73
5.1 Introduction	74
5.2 C++ code to Capture & Save Images	75
5.3 Slight Street Sign Modifications	76
5.4 HAAR CLASSIFICATION	76
5.4.1 Training of Haar Cascade Model for Stop Sign.....	77
5.4.2 Cascade Training Software and Image Cropping	78
5.5 Stop Sign Detection on Raspberry Pi3	82
5.5.1 Capturing Images.....	82
5.5.2 Multithreading in Capturing and Processing Images	83
5.5.3. Detecting Speed Signs.....	83
5.5.4 Open CV Contour Features and Edge Detection.....	84
5.5.5 Implementation of Linear Equations to Calculate Distance	85
 6 Traffic Signs Recognition	 86
6.1. overview	87
6.1.1 Traffic sign recognition.....	87
6.1.2 Recognizing Traffic Lights with Deep Learning.	87
6.1.3 Training Data.	88
6.1.4 Traffic Light Model	89
 7 Conclusion	 92
 References	 95

List of Figures

Figure 1 four WD robot (Page 20)

Figure 1.2 four WD real shot (page 21)

Figure 3 DS 4WD Robot Chassis with 4 TT Motor (page 23)

Figure 3.2 final car real shots (page 27)

Figure 3.3 final car inside view (page 28)

Figure 3.4 DC motor theory (page 32)

Figure 3.5 Dual H-bridge (page 35)

Figure 3.6 Powering L298N with Arduino (page 36)

Figure 3.7 L298N Block diagram (page 42)

Figure 4-1 Arduino types (page 44)

Figure 4-2 Arduino uno (Page 44)

Figure 3-3 Raspberry Pi 3 Model B 1GB Project Board (page 50)

Figure 4.5 Comparison of Raspberry Pi Models (Page 52)

Figure Raspberry Pi Gpio Access (page 55)

Fig4.1 road scene (page 60)

Figure 4.2 lane detection (page 60)

Fig 4.3 Number and rate of road traffic 2016 (page 60)

FIG 4.6Deteced interested lane (page 60)

fig4.7 ROI of path image (page 61)

Fig 4.8Original frame (page 61)

Fig 4.9canny edge frame (page 61)

Figure 5.3.1 Stop Sign (page 69)

Figure 5.3.2 An image of a panda, (page 69)

Figure 5.3.3 Top row shows legitimate sample images, (page 70)

Fig. 5.4: Various detection scenarios. (Page 70)

Fig 5.5.1Raspberry Pi 3 Model B Board (page 75)

Figure 5.5.2. Bounding rectangles for a set of speed signs (page 76)

Figure 6.1 – Shows non-identical traffic signs.(page 80)

Figure6. 2 -- (a) Partial occlusion, blurred traffic sign.

(c) destroyed traffic sign, (page 80)

Figure 6.3 Red yellow – green Traffic Lights (page 81)

Chapter 1

Introduction

1.1 History

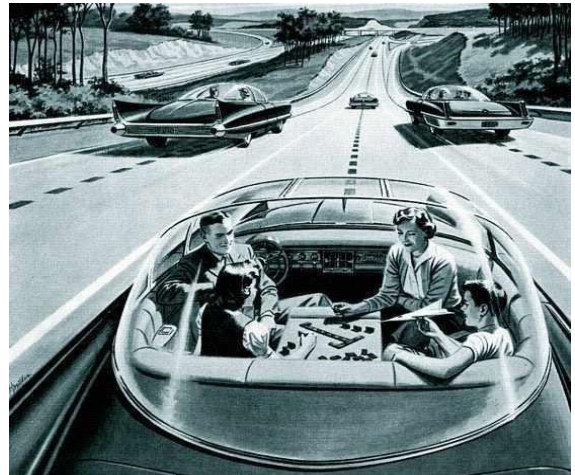
1930s

An early representation of the autonomous car was Norman Bell Geddes's Futurama exhibit sponsored by General Motors at the 1939 World's Fair, which depicted electric cars powered by circuits embedded in the roadway and controlled by radio.

1950s

In 1953, RCA Labs successfully built a miniature car that was guided and controlled by wires that were laid in a pattern on a laboratory floor. The system sparked the imagination of Leland M. Hancock, traffic engineer in the Nebraska Department of Roads, and of his director, L. N. Ress, state engineer. The decision was made to experiment with the system in actual highway

installations. In 1958, a full size system was successfully demonstrated by RCA Labs and the State of Nebraska on a 400-foot strip of public highway just outside Lincoln, Neb.



1980s

In the 1980s, a vision-guided Mercedes-Benz robotic van, designed by Ernst Dickmanns and his team at the Bundeswehr University Munich in Munich, Germany, achieved a speed of 39 miles per hour (63 km/h) on streets without traffic. Subsequently, EUREKA conducted the €749 million Prometheus Project on autonomous vehicles from 1987 to 1995.

1990s

In 1991, the United States Congress passed the ISTEA Transportation Authorization bill, which instructed USDOT to "demonstrate an automated vehicle and highway system by 1997." The Federal Highway Administration took on this task, first with a series of Precursor Systems Analyses and then by establishing the National Automated Highway System Consortium (NAHSC). This cost-shared project was led by FHWA and General Motors, with Caltrans, Delco, Parsons Brinkerhoff, Bechtel, UC-Berkeley, Carnegie Mellon University, and Lockheed Martin as additional partners. Extensive systems engineering work and research culminated in Demo '97 on I-15 in San Diego, California, in which about 20 automated vehicles, including cars, buses, and trucks, were demonstrated to thousands of onlookers, attracting extensive media coverage. The demonstrations involved close-headway platooning intended to operate in segregated traffic, as well as "free agent" vehicles intended to operate in mixed traffic.

2000s

The US Government funded three military efforts known as Demo I (US Army), Demo II (DARPA), and Demo III (US Army). Demo III (2001) demonstrated the ability of unmanned ground vehicles to navigate miles of difficult off-road terrain, avoiding obstacles such as rocks and trees. James Albus at the National Institute for Standards and Technology provided the Real-Time Control System which is a hierarchical control system. Not only were individual vehicles controlled (e.g. Throttle, steering, and brake), but groups of vehicles had their movements automatically coordinated in response to high level goals. The Park Shuttle, a driverless public road transport system, became operational in the Netherlands in the early 2000s. In January 2006, the United Kingdom's 'Foresight' think-tank revealed a report which predicts RFID-tagged driverless cars on UK's roads by 2056 and the Royal Academy of Engineering claimed that driverless trucks could be on Britain's motorways by 2019.

Autonomous vehicles have also been used in mining. Since December 2008, Rio Tinto Alcan has been testing the Komatsu Autonomous Haulage System – the world's first commercial autonomous mining haulage system – in the Pilbara iron ore mine in Western Australia. Rio Tinto has reported benefits in health, safety, and productivity. In November 2011, Rio Tinto signed a deal to greatly expand its fleet of driverless trucks. Other autonomous mining systems include Sandvik Automine's underground loaders and Caterpillar Inc.'s autonomous hauling.

In 2013, on July 12, VisLab conducted another pioneering test of autonomous vehicles, during which a robotic vehicle drove in downtown Parma with no human control, successfully navigating roundabouts, traffic lights, pedestrian crossings and other common hazards.

In 2011, the Freie Universität Berlin developed two autonomous cars to drive in the inner city traffic of Berlin in Germany. Led by the AUTONOMOS group, the two vehicles Spirit of Berlin and made in Germany handled intercity traffic, traffic lights and roundabouts between International Congress Centrum and Brandenburg Gate. It was the first car licensed for autonomous driving on the streets and highways in Germany and financed by the German Federal Ministry of Education and Research.

The 2014 Mercedes S-Class has options for autonomous steering, lane keeping, acceleration/braking, parking, accident avoidance, and driver fatigue detection, in both city traffic and highway speeds of up to 124 miles (200 km) per hour.

Released in 2013, the 2014 Infiniti Q50 uses cameras, radar and other technology to deliver various lane-keeping, collision avoidance and cruise control features. One reviewer remarked, "With the Q50 managing its own speed and adjusting course, I could sit back and simply watch, even on mildly curving highways, for three or more miles at a stretch adding that he wasn't touching the steering wheel or pedals.

Although as of 2013, fully autonomous vehicles are not yet available to the public, many contemporary car models have features offering limited autonomous functionality. These include adaptive cruise control, a system that monitors distances to adjacent vehicles in the same lane, adjusting the speed with the flow of traffic lane which monitors the vehicle's position in the lane, and either warns the driver when the vehicle is leaving its lane, or, less commonly, takes corrective actions, and parking assist, which assists the driver in the task of parallel parking

1.2 Why autonomous car is important

1.2.1 Benefits of Self-Driving Cars

1. Fewer accidents

The leading cause of most automobile accidents today is driver error. Alcohol, drugs, speeding, aggressive driving, over-compensation, inexperience, slow reaction time, inattentiveness, and ignoring road conditions are all contributing factors. Given some 40 percent of accidents can be traced to the abuse of drugs and or alcohol, self-driving cars would practically eliminate those accidents altogether.

2. Decreased (or Eliminated) Traffic Congestion

One of the leading causes of traffic jams is selfish behavior among drivers. It has been shown when drivers space out and allow each other to move freely between lanes on the highway, traffic continues to flow smoothly, regardless of the number of cars on the road.

3. Increased Highway Capacity

There is another benefit to cars traveling down the highway and communicating with one another at regularly spaced intervals. More cars could be on the highway simultaneously because they would need to occupy less space on the highway

4. Enhanced Human Productivity

Currently, the time spent in our cars is largely given over to simply getting the car and us from place to place. Interestingly though, even doing nothing at all would serve to increase human productivity. Studies have shown taking short breaks increase overall productivity. You can also finish up a project, type a letter, monitor the progress of your kid's schoolwork, return phone calls, take phone calls safely, text until your heart's content, read a book, or simply relax and enjoy the ride .

5. Hunting For Parking Eliminated

Self-driving cars can be programmed to let you off at the front door of your destination, park themselves, and come back to pick you up when you summon them. You're freed from the task of looking for a parking space, because the car can do it all

6. Improved Mobility For Children, The Elderly, And The Disabled

Programming the car to pick up people, drive them to their destination and Then Park by themselves, will change the lives of the elderly and disabled by providing them with critical mobility.

7. Elimination of Traffic Enforcement Personnel

If every car is “plugged” into the grid and driving itself, then speeding, along with stop sign and red light running will be eliminated. The cop on the side of the road measuring the speed of traffic for enforcement purposes? Yeah, they’re gone. Cars won’t speed anymore. So no need to Traffic Enforcement Personnel.

8. Higher Speed Limits

Since all cars are in communication with one another, and they’re all programmed to maintain a specific interval between one another, and they all know when to expect each other to stop and start, the need to accommodate human reflexes on the highway will be eliminated. Thus, cars can maintain higher average speeds.

9. Lighter, More Versatile Cars

The vast majority of the weight in today’s cars is there because of the need to incorporate safety equipment. Steel door beams, crumple zones and the need to build cars from steel in general relate to preparedness for accidents. Self-driving cars will crash less often, accidents will be all but eliminated, and so the need to build cars to withstand horrific crashes will be reduced. This means cars can be lighter, which will make them more fuel-efficient

1.3 What Are Autonomous and Automated Vehicles

Technological advancements are creating a continuum between conventional, fully human- driven vehicles and automated vehicles, which partially or fully drive themselves and which may ultimately require no driver at all. Within this continuum are technologies that enable a vehicle to assist and make decisions for a human driver.

.

1.4 Project description



1.4.1 Image processing

It is a feature which is used to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image.

Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

1.4.2 Lane detection

In this feature we perform a development of a self-driving car with the lane line detection system using images processing technique found that the system can be operated by taking images from the camera on the self-driving car and processed by the Raspberry Pi board using image processing from the car camera to detect and track the road. The system is 90 percent accurate in an environment where there are no objects or obstructions, the road is clearly seen and the light not too bright, so the system can detect the color code on the lane line as well as can great track of traffic lane.

1.4.3 Obstacle detection

The purpose of this project is to design a navigation system and an obstacle avoidance system for an autonomous wheelchair application. The obstacle detection system is done merely using the camera sensor with the knowledge of image processing. The techniques used are including Canny Edge Detection, Erosion and so on where algorithms are developed to identify and detect the obstacle.

1.4.4 Traffic signs

The main concern with traffic sign recognition system is not how to recognize or identify a traffic sign with a high reminder in a still image. In fact, it is how to achieve high accuracy in high-resolution live video streaming with real-time capability. In order to demonstrate the problem of false detection.

Chapter 2

Car design

2.1 First car design

in this section, we will talk about the trials that we passed by till reaching our prototype. This car performs some functions of our project such as the adaptive cruise control (ACC), lane keeping and lane departure.

2.1.1 Four WD Robot (Acrylic with 4 Motors and 4 Wheels)

The 4WD robot consists of four gear-motors with 65mm diameter wheels. The chassis plates contain numerous cuts and holes for mounting sensors, microcontrollers, and other hardware. The space between the plates is ideal for batteries or more components.

2.1.1.1 Specifications

1) Motors

- Suggested Voltage:4.5V DC (work well from 3-6V)
- No load Speed:90±10rpm
- No Load
Current:190mA(max.250mA)
- Torque:800gf.cm (Minimum)
- Stall current 1A

2) Wheels

- 65mm diameter, 30mm width
- Plastic rims with solid rubber tires

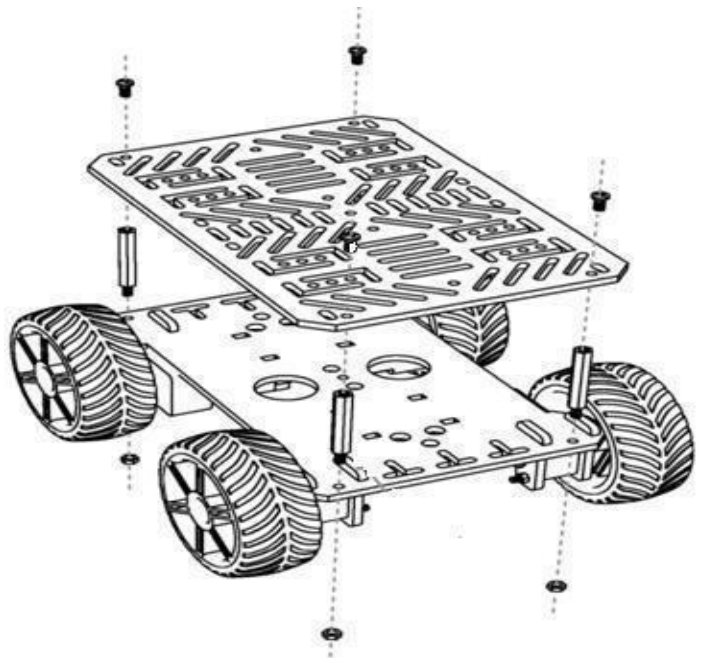


Fig 2-1 four WD robot

3) Chassis

- Laser cut acrylic
- Metal standoffs
- Top and bottom plates are both
110mm long x 174mm wide

2.1.1.2 Real shots



Figure 2-2 four WD real shot

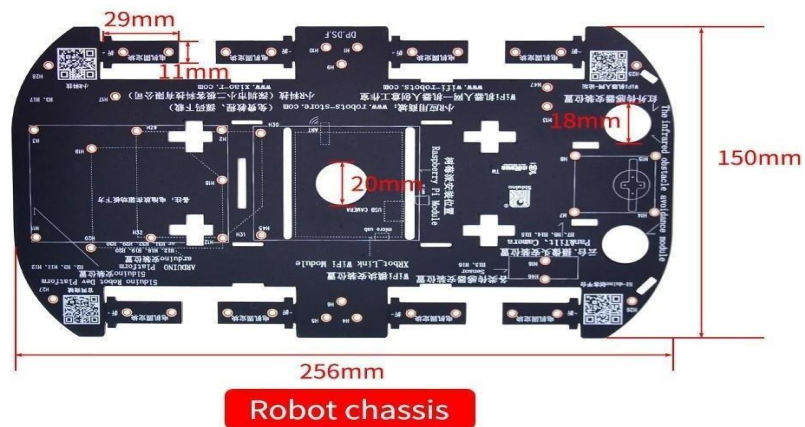
2.1.1.3 Drawbacks

- 1) The 4WD robot uses four wheels with four Dc motors which is not similar to a real car.
- 2) The steering mechanism is not accurate, which causes many problems while working.
- 3) The robot size is very small which is not stable and can be easily broken.

2.1.1.4 Overcoming the drawbacks

- 1) Implementing a mechanical design to serve our needs
- 2) Using motor with steering mechanism to give stable steering while turning left or right.

2.1.1.5 Specifications



Tire



Motor

Figure 2.3 DS 4WD Robot Chassis with 4 TT Motor

1)Motors

*DC motor with dual H-bridge.

1) Wheels

- * 4 rubber wheels.
- * Plastic rims with solid rubber tires.

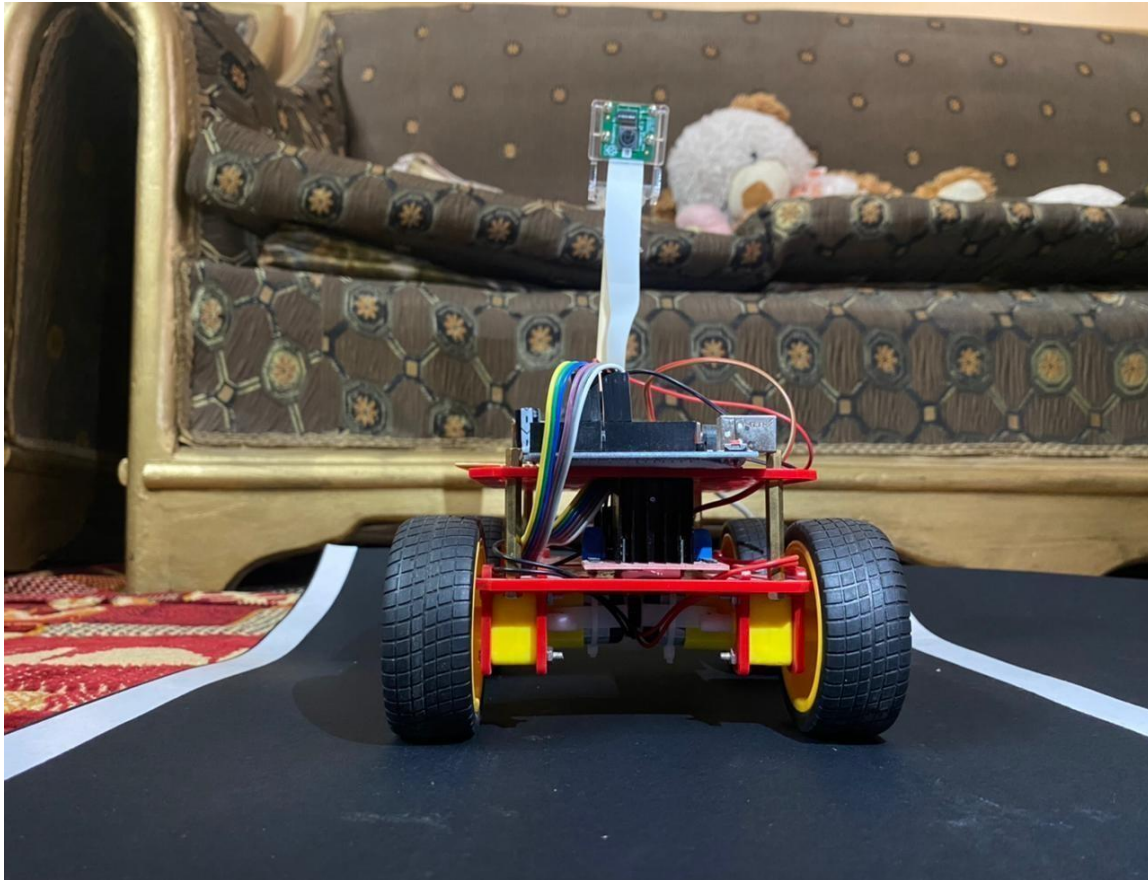
2) Chassis

- * Laser cut acrylic.

2.1.1.6 Real Shots

First shot shows the motor with the front wheel, Arduino card, and back wheel with the driving DC motor.

The car with all components on and connected.



2.1.3.2 Real shots



Figure 2-4 final car real shots

2.2 Motors

The motor is the primary tool for creating motion. At its simplest use, you can use it to make something spin. With a little more mechanical work, using gears and other mechanical devices, you can use a motor to make something move, vibrate, rise, fall, roll, creep, or perform almost any other type of motion that does not require precise positioning. There are several different kinds of motors: servos, stepper motors, or unidirectional DC motors. In this section, we will talk about DC and how it works.

2.2.1 DC Motor

D. C motors are seldom used in ordinary applications because all electric supply companies furnish alternating current. However, for special applications such as in steel mills, mines and electric trains, it is advantageous to convert alternating current into direct current in order to use DC motors. The reason is that speed/torque characteristics of DC motors are much more superior to that of AC motors. Therefore, it is not surprising to note that for industrial drives, DC motors are as popular as 3-phase induction motors. Like DC generators, dc motors are also of three types. (series-wound, shunt-wound and compound wound). The use of a particular motor depends upon the mechanical load it has to drive. Motors are all around us; just look inside moving toys, and you'll find a number of excellent motors and gears. Any electronics supplier will have a wide range of motors that will suit many purposes from spinning small objects to driving large loads

2.2.1.1 D.C Motor Fundamentals

DC motors consist of rotor (or armature), commutator, brushes, rotating shaft and bearings, stator with permanent magnet. The principle of operation with a simple two-pole dc motor: The torque is produced by the fact that like field poles attracts and unlike poles repel.

2.2.1.2 DC Motor Principle

It is a machine that converts DC power into mechanical power. Its operation is based on the principle that when a current carrying conductor is placed in a magnetic field, the conductor experiences a mechanical force. There is no constructional difference between a DC motor and a DC generator. The same DC machine can be run as generator motor.

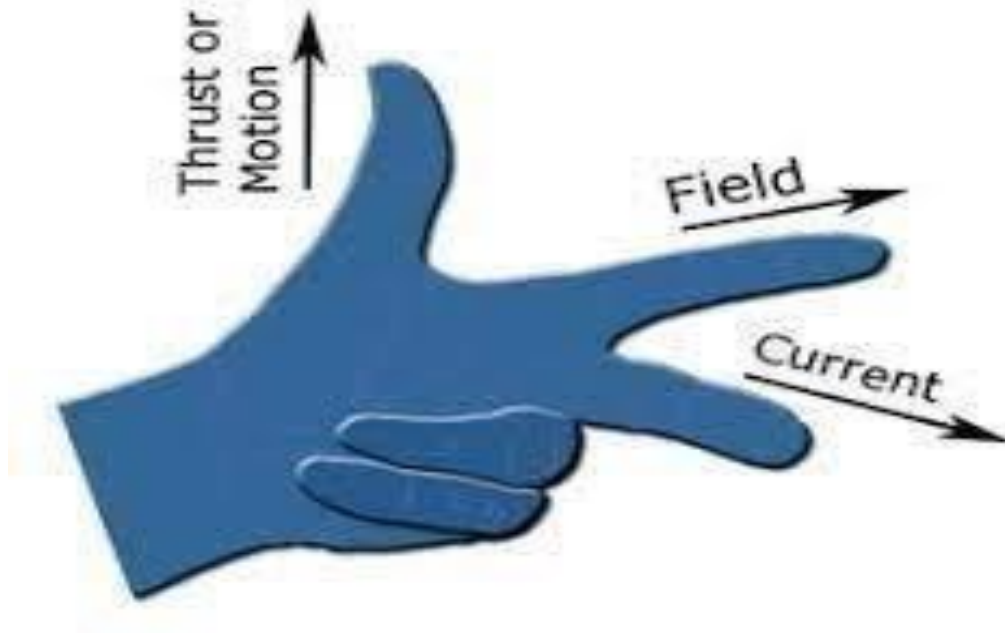


Figure 2-5 DC motor theory

2.2.1.3 How DC Motors Work

In a simple DC motor, there are two main components, the “stator” and the “armature”. The stator is a permanent magnet and provides a constant magnetic field. The armature, which is the rotating part, is a simple coil.

The armature is connected to a DC power source using a 2-piece ring installed around the motor shaft, these ring sections are called “commutator rings”. The two pieces of the commutator rings are connected to each end of the armature coil. Direct Current of a suitable voltage is applied to the commutator rings via two “brushes” that rub against the rings.

When DC is applied to the commutator rings it flows through the armature coil, producing a magnetic field. This field is attracted to the stator magnet (remember, opposite magnetic polarities attract, similar ones repel) and the motor shaft begins to spin

The motor shaft rotates until it arrives at the junction between the two halves of the commutator. At that point the brushes come into contact with the other half of the commutator rings, reversing the polarity of the armature coil (or coils, most modern DC motors have several). This is great because at this point the motor shaft has rotated 180 degrees and the magnetic field polarities need to be reversed for the motor to continue rotating. This process repeats itself indefinitely until the current is removed from the armature coils

2.3 Motor and its connection with the Arduino

2.3.1 H-Bridge

An H bridge is an electronic circuit that enables voltage to be supplied to the DC motor and control its direction. This circuit is often used in robotics and other applications. at our project we use l298n bridge

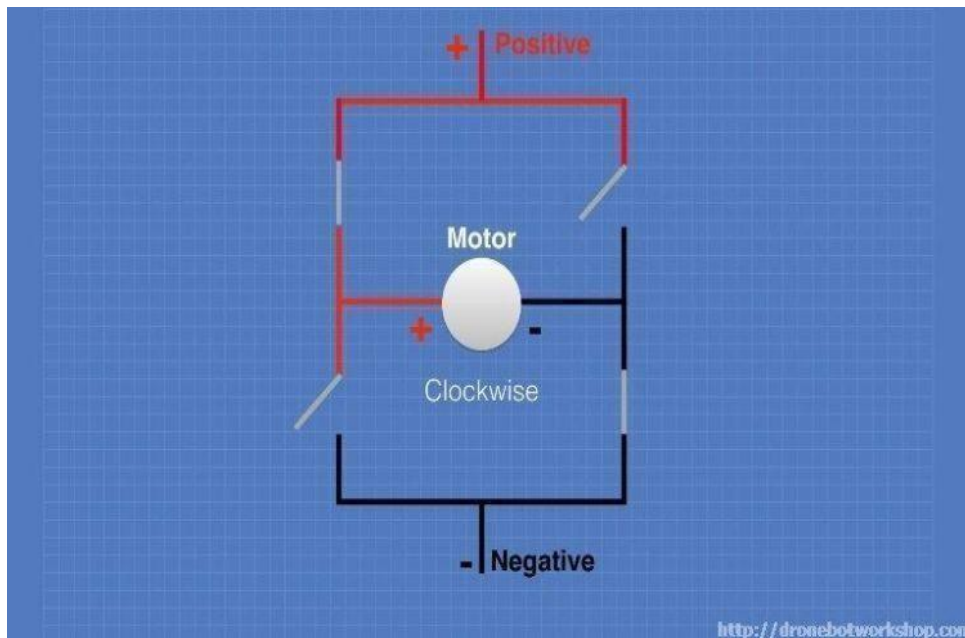
H- Bridge (L298N)

This is how DC motors work, how-to reverse their direction by changing polarity and how you can change their speed using pulse width modulation, let us examine an easy way to do this using a very common circuit configuration called an “H-Bridge.”

An “H-Bridge” is simply an arrangement of switching the polarity of the voltage applied to a DC motor, thus controlling its direction of rotation. To visualize how this all works I’ll use some switches, although in real life an H-Bridge is usually built using transistors. Using transistors also allows you to control the motor speed

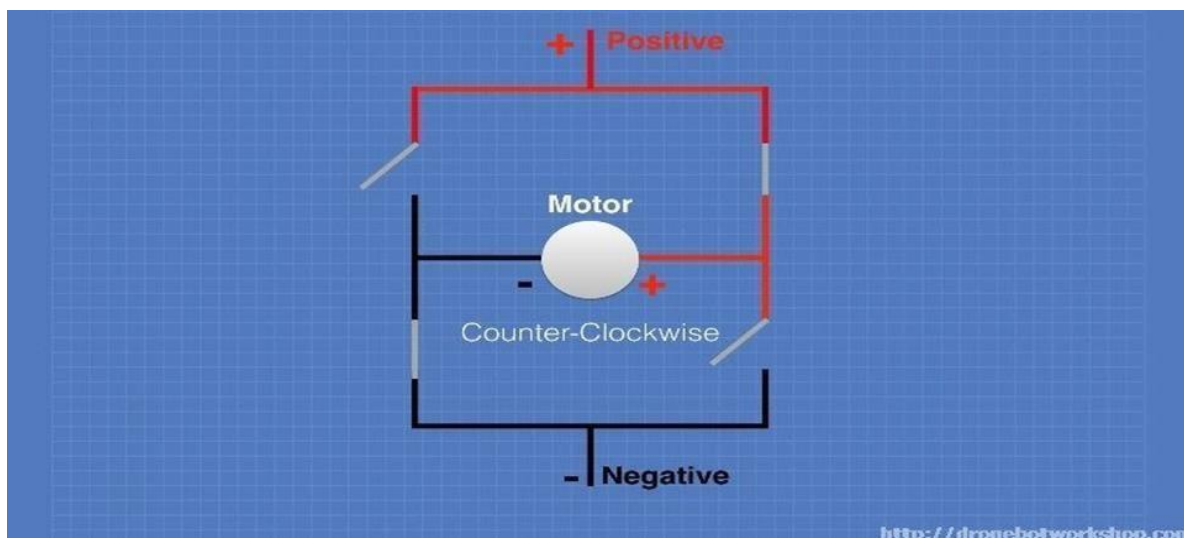
with PWM, as described above.

In the first diagram we can see four switches which are all in the open or “off” position. In the center of the circuit is a DC motor. If you look at the circuit as it is drawn here you can distinctly see a letter “H”, with the motor attached in the center or “bridge” section – thus the term “H-Bridge”



If we close (i.e., turn on) two of the switches you can see how the voltage is applied to the motor, causing it to turn clockwise.

Now we will open those switches and close the other two. As you can see this causes the polarity of the voltage applied to the motor to be reversed, resulting in our motor spinning counterclockwise



This is simple but effective. In fact, if all you need to do is design a circuit to drive the motor full-speed in either direction you could build this as shown here, using a 4PDT (4 Pole Double-Throw) center-off switch. But of course, we want to control the motor using an Arduino, so an electronic circuit where the switches are replaced by transistors is what we need.

2.3.1.1 Features

- Dual H bridge drive (can drive 2 DC motors).
- Chip L298N.
- Logical voltage 5V.
- Drive voltage 5V-35V.
- Logic current 0mA-36mA.
- Drive current 2A (For each DC motor)).
- Weight 30gm.
- Size: 43*43*27mm.

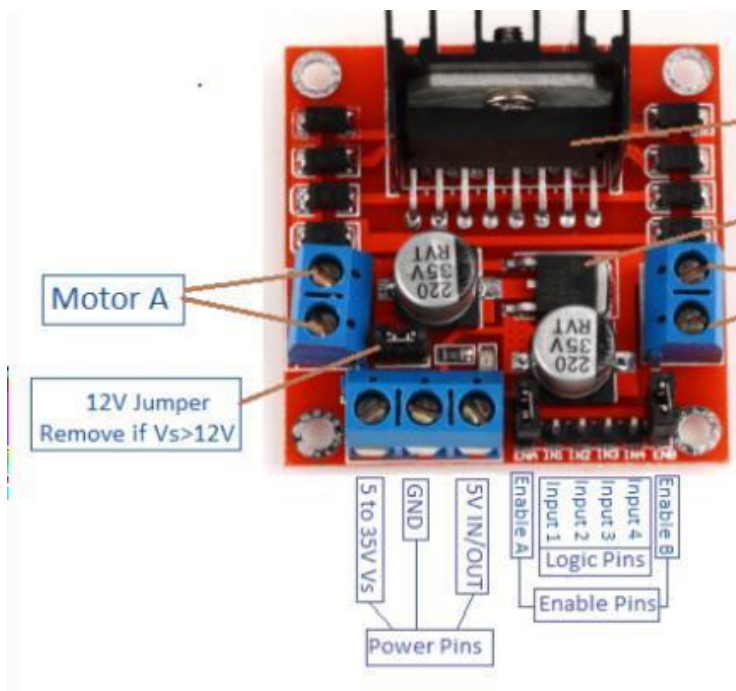
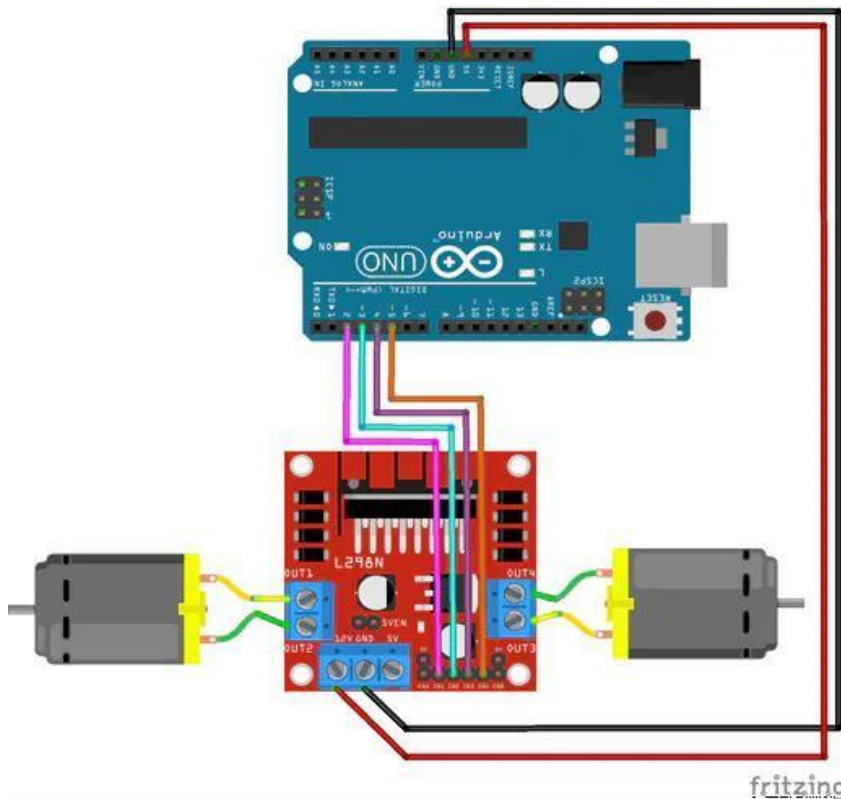


Figure 2-6 Dual H-bridge

2.4 Driver connection with Arduino

- 1- GND.
- 2- + 5 V (power for driver (not motor)).
- 3- ENA: Motor enable for Motor A (high/low).
- 4, 5- IN1, IN2: pins control Motor A direction of rotation (one is high and the other is low).
- 6- ENB: Motor enable for Motor B (high/low).
- 7, 8- IN3, IN4: These pins define Motor B direction of rotation (one is high and the other is low).

figPowering L298N with Arduino



2.4.1 Left & Right Functions for Motors:

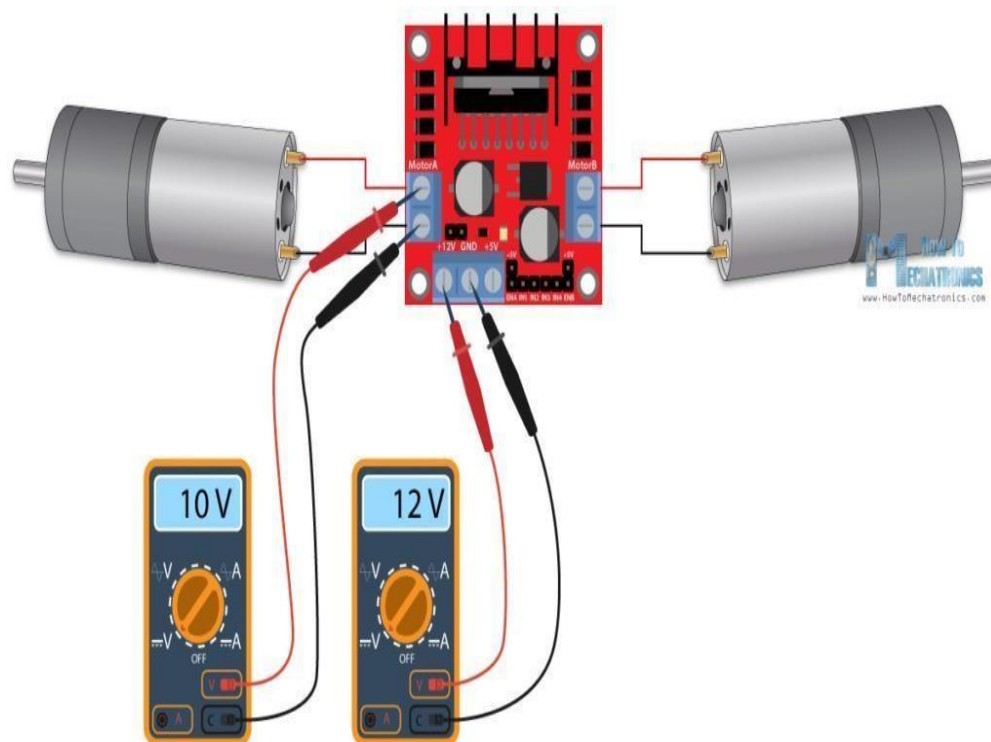
the code Setting IN1 to HIGH and IN2 to LOW will cause the left motor to turn a direction. Setting IN1 to LOW and IN2 to HIGH will cause the left motor to spin the other direction. The same applies to IN3 and IN4.

2.4.1.1 Implementation & testing

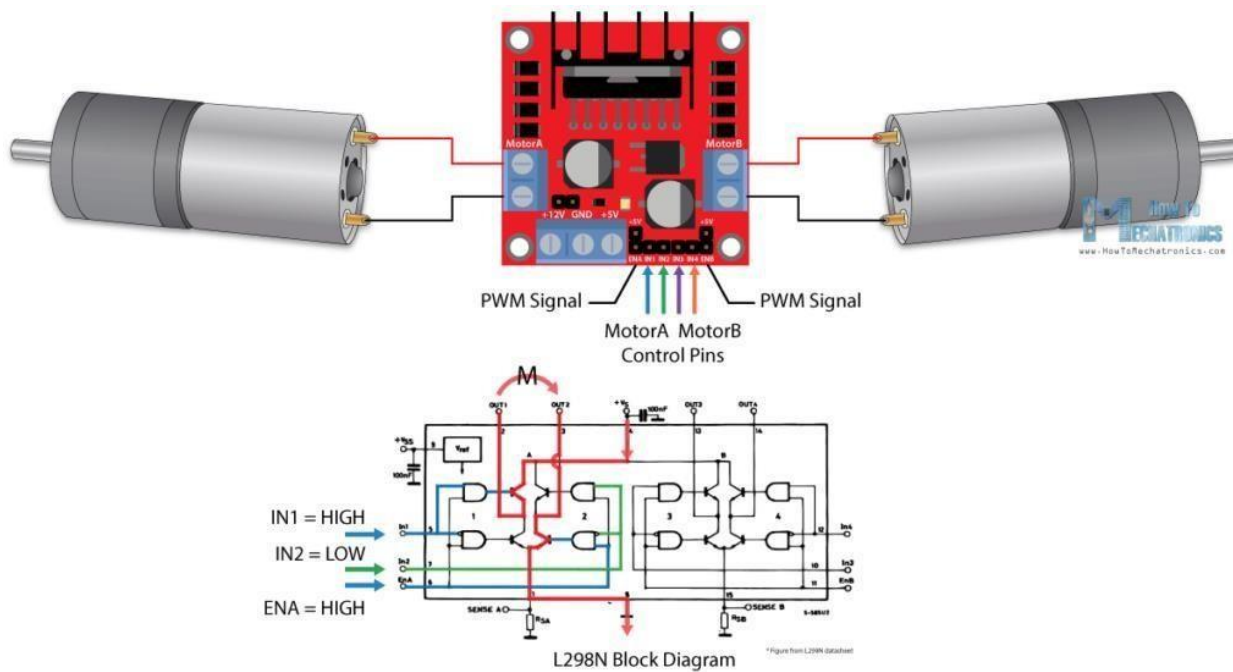
```
{  
    digitalWrite(HighL, LOW);  
    digitalWrite(LowL, HIGH);  
    analogWrite(EnableL, 100);  
  
    digitalWrite(HighR, LOW);  
    digitalWrite(LowR, HIGH);  
    analogWrite(EnableR, 255);  
}  
  
void Right()  
{  
    digitalWrite(HighL, LOW);  
    digitalWrite(LowL, HIGH);  
    analogWrite(EnableL, 255);  
  
    digitalWrite(HighR, LOW);  
    digitalWrite(LowR, HIGH);  
    analogWrite(EnableR, 100);  
}
```

2.4.2 Forward & Backward Functions for Motors:

IC makes a voltage drop of about 2V. So for example, if we use a 12V power supply, the voltage at motors terminals will be about 10V, which means that we won't be able to get the maximum speed out of our 12V DC motor.



the logic control inputs. The Enable A and Enable B pins are used for enabling and controlling the speed of the motor. If a jumper is present on this pin, the motor will be enabled and work at maximum speed, and if we remove the jumper we can connect a PWM input to this pin and in that way control the speed of the motor. If we connect this pin to a Ground the motor will be disabled.



the Input 1 and Input 2 pins are used for controlling the rotation direction of the motor A, and the inputs 3 and 4 for the motor B. Using these pins we actually control the switches of the H-Bridge inside the L298N IC. If input 1 is LOW and input 2 is HIGH the motor will move forward, and vice versa, if input 1 is HIGH and input 2 is LOW the motor will move backward. In case both inputs are same, either LOW or HIGH the motor will stop. The same applies for the inputs 3 and 4 and the motor B.

2.4.2.1 Implementation & testing

```
}  
  
void Forward()  
{  
    digitalWrite(HighL, LOW);  
    digitalWrite(LowL, HIGH);  
    analogWrite(EnableL, 255);  
  
    digitalWrite(HighR, LOW);  
    digitalWrite(LowR, HIGH);  
    analogWrite(EnableR, 255);  
}  
  
void Backward()  
{  
    digitalWrite(HighL, LOW);  
    digitalWrite(LowL, HIGH);  
    analogWrite(EnableL, 255);  
  
    digitalWrite(HighR, LOW);  
    digitalWrite(LowR, HIGH);  
    analogWrite(EnableR, 255);  
}
```

Chapter 3

Microcontrollers and hardware

3.1 Micro Controllers

A Microcontroller (sometimes abbreviated μC , or MCU) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general-purpose applications.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems. Some microcontrollers may use four-bit words and operate at clock rate frequencies as low as 4 kHz, for low power consumption (single-digit milliwatts or microwatts). They will have the ability to retain functionality while waiting for an event such as a button press or other interrupt, power consumption while sleeping (CPU clock and most peripherals off) may be just Nano watts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.

3.1.1 Arduino

3.1.1.1 Introduction

1. What is Arduino?

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can be communicating with software running on your computer (e.g. Flash, Processing, MaxMSP.) The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

2. Why Arduino?

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's

Handy board, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and +wrap it up in an easy-to-use package.

Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- **Inexpensive**
Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than \$50
- **Cross-platform**
The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment**
The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino
- **Open source and extensible software**
The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- **Open source and extensible hardware**
The Arduino is based on Atmel's ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

Types of Arduinos.

There are several types of Arduinos.

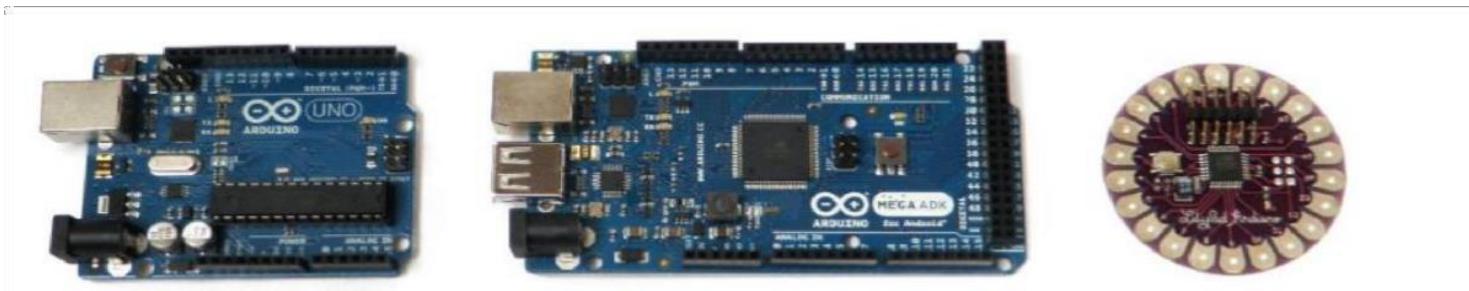


Figure 4-1 Arduino types

3.1.1.1 Arduino Uno

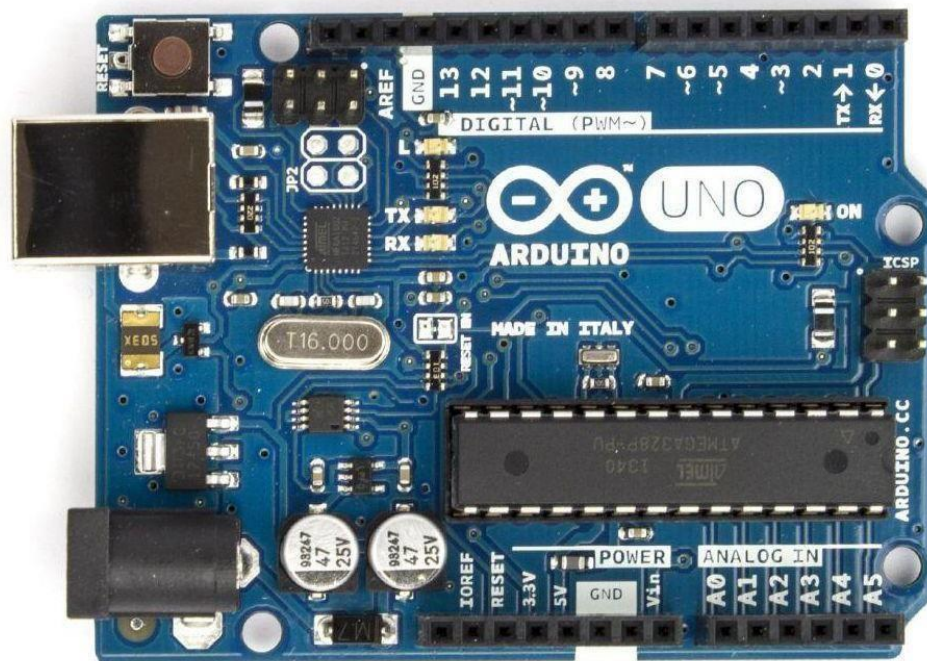


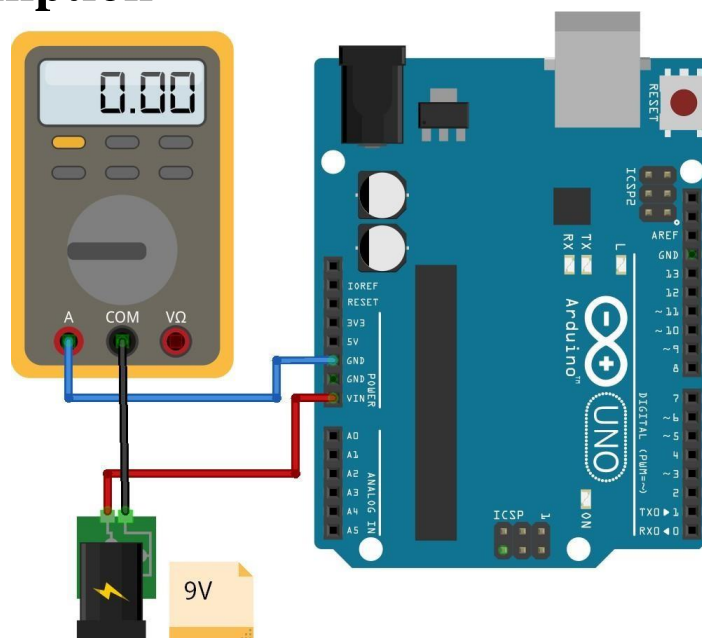
Figure 4-2 Arduino uno

1. Overview

The Arduino Uno is a microcontroller board based on the ATmega328. It has 20 digital input/output pins (of which 6 can be used as PWM outputs and 6 can be used as analog inputs), a 16 MHz resonator, a USB connection, a power jack, an in-circuit system programming (ICSP) header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a [AC-to-DC adapter](#) or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features an ATmega16U2 programmed as a USB-to-serial converter. This auxiliary microcontroller has its own USB bootloader, which allows advanced users to reprogram it.

2. Power consumption



The Arduino Uno has a lot of different pins and therefore we want to go over the various kinds of pins.

The Uno has in total three power pins of which one has a supply voltage of 3.3V and two pins provide 5V. All power pins have a maximum current of 50 mA. You can use the VIN pin to power the whole microcontroller with a voltage between 7V-12V, also perfect for a battery. Of course if you have power pins you also need some ground pins to close the electric circuit. The Arduino Uno has in total three ground pins which are all connected internally.

It has in total 14 digital pins which provide a maximum current of 20 mA. Six of the 14 digital I/O pins can produce a PWM signal.

If you want to communicate between multiple devices, you need communication pins which are also provided by the Arduino. The microcontroller has for each communication protocol.

1.1 How to Power Arduino Uno: 3 Possibilities

You can power your Arduino Uno in 3 save ways because a voltage regulator provides a regulated and stable voltage for the microprocessor:

1. **USB cable:** The most popular and also the easiest way to power the microcontroller is via USB cable. The standard USB connection delivers 5V and allows you to draw 500mA in total.
2. **DC Power Jack:** It is possible to use the DC power Jack as power supply. If you buy a DC power jack, make sure the power adapter of the plug supplies a voltage between 7V and 12V.
3. **VIN Pin:** If you use an external power supply like a battery, you can use the VIN pin. The voltage must be between 7V and 12V. Therefore, you can power the Uno with an external 9 Volt battery.

1.1.2 General pin functions

- **LED:** There is a built-in LED driven by digital pin 13. When the pin is high value, the LED is on, when the pin is low, it is off.
- **VIN:** The input voltage to the Arduino board when it is using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V:** This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 20V), the USB connector (5V), or the VIN pin of the board (7-20V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator and can damage the board.
- **3V3:** A 3.3-volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND:** Ground pins.
- **IOREF:** This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.
- **Reset:** Typically used to add a reset button to shields that block the one on the board.

1.1.3 Special pin functions

Each of the 14 digital pins and 6 analog pins on the Uno can be used as an input or output, under software control (using `pinMode()`, `digitalWrite()`, and `digitalRead()` functions). They operate at 5 volts. Each pin can provide or receive 20 mA as the recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50K ohm. A maximum of 40mA must not be exceeded on any I/O pin to avoid permanent damage to the

microcontroller. The Uno has 6 analog inputs, labeled A0 through A5; each provides 10 bits of resolution (i.e. 1024 different values). By default, they measure from ground to 5 volts, though it is possible to change the upper end of the range using the AREF pin and the `analogReference()` function.

In addition, **some pins have specialized functions:**

- **Serial / UART:** pins 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL serial chip.
- **External interrupts:** pins 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- **PWM** (Pulse-width modulation): pins 3, 5, 6, 9, 10, and 11. Can provide 8-bit PWM output with the `analogWrite()` function.
- **SPI** (Serial Peripheral Interface): pins 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK). These pins support SPI communication using the SPI library.
- **TWI** (two-wire interface) : pin SDA (A4) and pin SCL (A5). Support TWI communication using the Wire library.
- **AREF** (analog reference): Reference voltage for the analog inputs.

3. Communication

The Arduino Uno has a few facilities for communicating with a computer, another Arduino board, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The 16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. Arduino Software (IDE) includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A Software Serial library allows serial communication on any of the Uno's digital pins.

4. Memory

The Arduino UNO has only **32K bytes of Flash memory and 2K bytes of SRAM**. That is more than 100,000 times LESS physical memory than a low-end PC! And that is not even counting the disk drive! Working in this minimalist environment, you must use your resources wisely.

5. Technical specifications



- Microcontroller: Microchip ATmega328P
- Operating Voltage: 5 Volts
- Input Voltage: 7 to 20 Volts
- Digital I/O Pins: 14 (of which 6 can provide PWM output)
- UART: 1
- I2C: 1
- SPI: 1
- Analog Input Pins: 6
- DC Current per I/O Pin: 20 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB of which 0.5 KB used by bootloader
- SRAM: 2 KB
- EEPROM: 1 KB
- Clock Speed: 16 MHz
- Length: 68.6 mm
- Width: 53.4 mm
- Weight: 25 g

5. Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno board is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nano-farad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip

This setup has other implications. When the Uno is connected to a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e., anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened.

3.1.2 Raspberry Pi

3.1.2.1 Design

The Raspberry Pi is a single-board computer developed in the UK by the Raspberry Pi. The Raspberry Pi is a credit-card sized computer that plugs into your TV and a keyboard. It's a capable little PC which can be used for many of the things that your desktop PC does, like spreadsheets, word-processing and games.

The design is based around a Broadcom BCM2835 SoC, which includes an ARM1176JZF-S 700 MHz processor and 512 Megabytes of RAM.

The design does not include a built-in hard disk or solid-state drive, instead relying on an SD card for booting and long-term storage. This board is intended to run Linux kernel based operating systems cost designs.

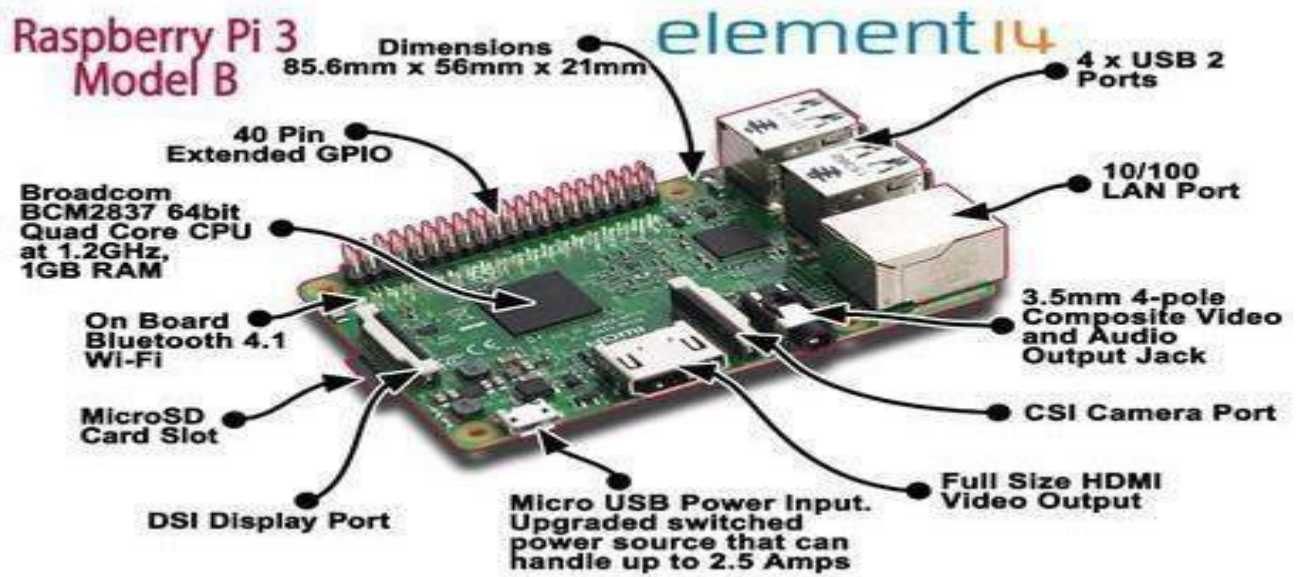


Figure 4-3 Raspberry Pi 3 Model B 1GB Project Board

3.1.2.1 Features

- Low risk and fast time to market
- High performance in low-cost designs
- Physically addressed caches for multi-tasking performance
- Broad OS support, multiple Linux distributions, amazing ARM ecosystem
- Full Internet experience
- Low Power Leadership
- 93% of flops are clock gated

3.1.2.2 Performance

The ARM1176 processor performance reaches up to 1GHz and beyond in 40G, and can reach 1GHz in 65nm with overdrive voltages.

PPA	ARM1176
Process Geometry	TSMC65LP
Performance (DMIPS)	603
Performance (Coremarks)	1002
Frequency * (MHz)	482
Total area (mm2)	1.75
Power efficiency (DMIPS/mW)	3

3.1.2.3 Functions supported in hardware

- Multiplication, addition, and multiply-accumulate (various variants)
- Division and square root operation (multi-cycle, not pipelined)
- Comparisons and format conversions
- Operations can be performed on short vectors (From assembler only)
- Separate pipelines allow load/store and MAC operations to occur simultaneously with divide/square root unit operation
- Clock gated and/or power completely removed

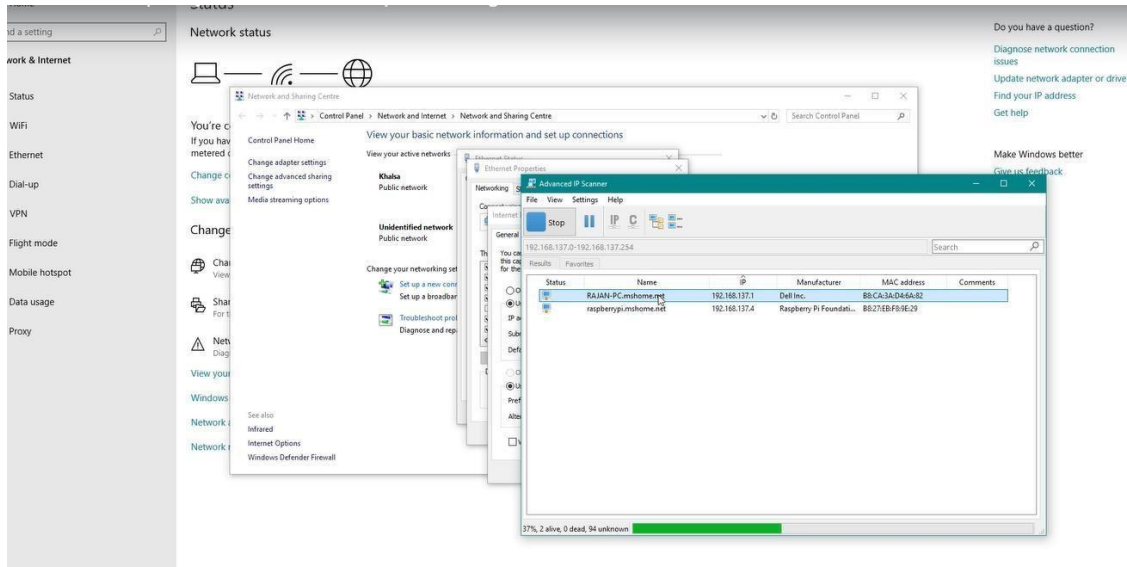
3.1.2.4 Comparison

	Raspberry Pi 3 Model B	Raspberry Pi 2 Model B	Raspberry Pi Model B+
Processor Chipset	Broadcom BCM2837 64Bit Quad Core ARM Cortex A53 at 1.2GHz	Broadcom BCM2836 32Bit Quad Core ARMv7 at 900MHz	Broadcom BCM2835 32Bit ARMv6k at 700MHz
GPU	Videocore IV @ 400MHz	Videocore IV @ 250MHz	Videocore IV @ 250MHz
Processor Speed	QUAD Core @1.2 GHz	QUAD Core @900 MHz	Single Core @700 MHz
RAM	1GB SDRAM @ 400 MHz	1GB SDRAM @ 400 MHz	512 MB SDRAM @ 400 MHz
Storage	MicroSD	MicroSD	MicroSD
USB 2.0	4x USB Ports	4x USB Ports	4x USB Ports
Max Power Draw/voltage	2.5A @ 5V	1.8A @ 5V	1.8A @ 5V
GPIO	40 pin	40 pin	40 pin
Ethernet Port	Yes	Yes	Yes
WiFi	Built in	No	No
Bluetooth LE	Built in	No	No
Video Output	HDMI/Composite via RCA Jack	HDMI/Composite via RCA Jack	HDMI/Composite via RCA Jack
Audio Output	3.5mm Jack	3.5mm Jack	3.5mm Jack

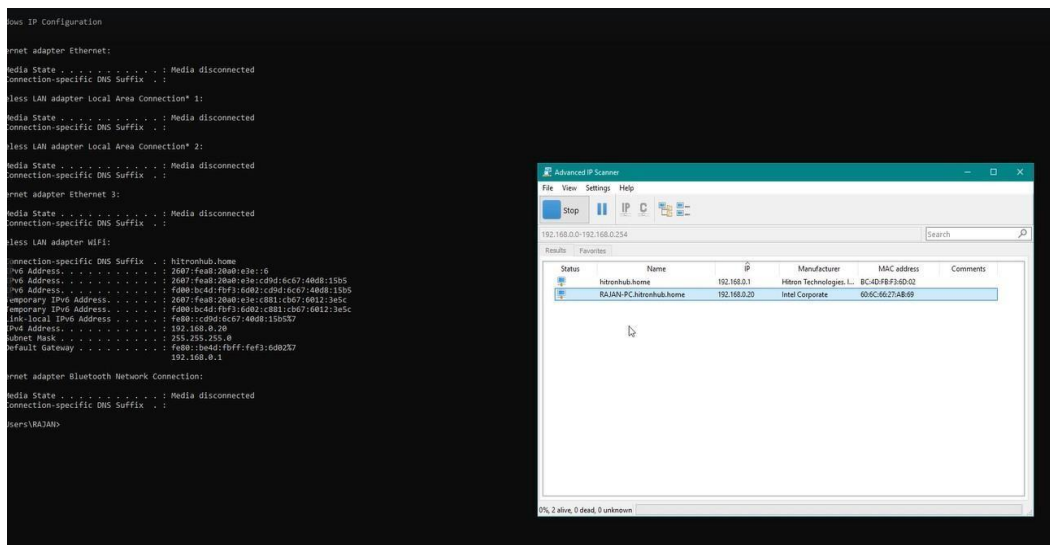
Figure 4.5 Comparison of Raspberry Pi Models

3.1.2.6 Connection of Raspberry Pi With personal PC

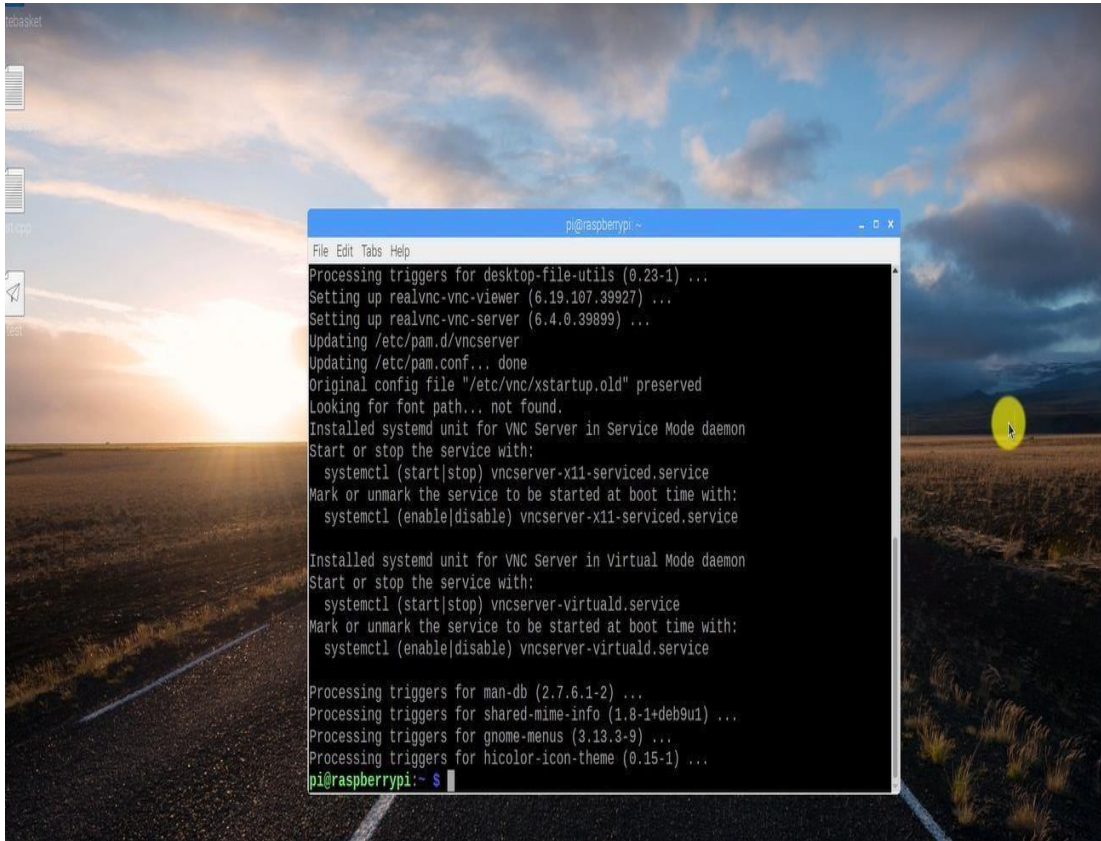
1.1 Connect Raspberry PI to Personal Computer through Ethernet



1.2 Connect Raspberry PI to Personal Computer through Wi-Fi



1.3 Connect Raspberry PI to Personal Computer through VNC Viewer



3.1.1.1 Arduino connection with Raspberry Pi

Here is we will use the raspberry Pi as master and Arduino as a slave, and we will use 4 different pins which can drive up to total 16 different digital combinations.

Setting out the pins on the Raspberry Pi.

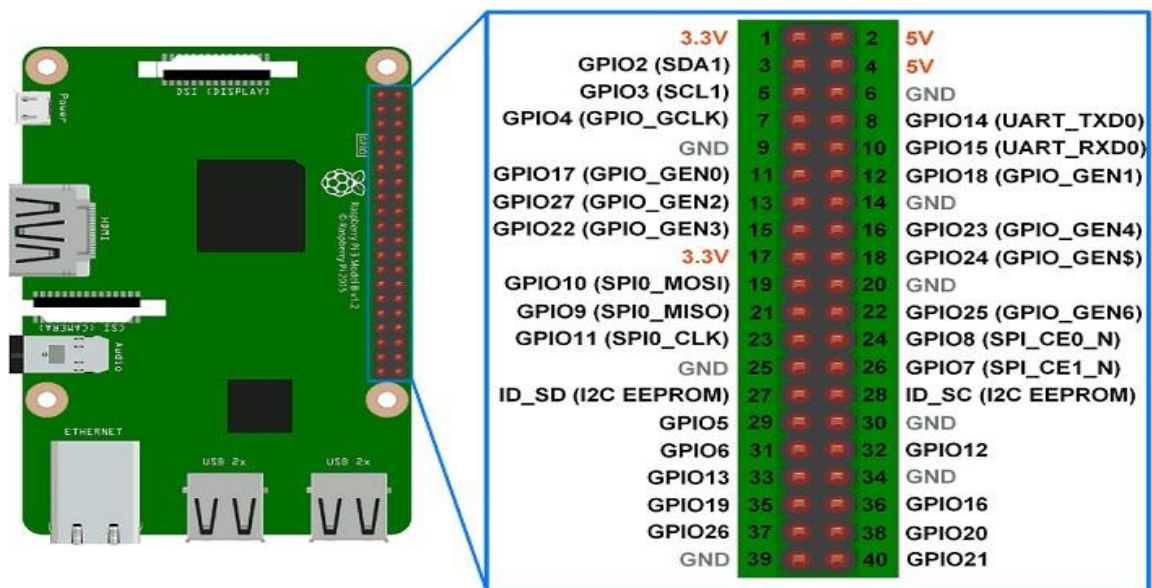


Figure Raspberry Pi Gpio Access

175	
176	else if (result <0 && result >-10)
177	{
178	digitalWrite(21, 0);
179	digitalwrite(22, 0); //decimal = 4
180	digitalwrite(23, 1);
181	digitalwrite(24, 0);
182	cout<<"Left1"<<endl;
183	}
184	
185	else if (result <=-10 && result >-20)
186	{
187	digitalWrite(21, 1);
188	digitalwrite(22, 0); //decimal = 5
189	digitalwrite(23, 1);
190	digitalwrite(24, 0);
191	cout<<"Left2"<<endl;
192	}
193	
194	else if (result <=-20)
195	{
196	digitalWrite(21, 0);
197	digitalwrite(22, 1); //decimal = 6
198	digitalwrite(23, 1);
199	digitalwrite(24, 0);
200	cout<<"Left3"<<endl;
201	}

```
>}mb0|5 )' ° @
```

```
"" "" • 139'
' Capture [38] 140
' Histogram [ 141
' LaneCenter [ 142
' LaneFinder [ 143
' Perspective [ 144
' Setup [26] 145
' Threshold [5 146
```

```
if (result == 0)
{
    digitalwrite(21, 0);
    digitalWrite(22, 0); //decimal = 0
    digitalWrite(23, 0);
    digitalWrite(24, 0);
    cout<< "!" << endl;
}
```

```
variables 149
*""""!"" 158.
Destination [ 151
LeftLanePos 152
Matrix [11] 153
ROI_Lane [12] 154
Result [13] 155
```

```
else If (result >0 It resvlt <10)
{
    digitalWrite(21, 1);
    digitalWrite(22, 0); //decimal= 1
    digitalWrite(23, 0);
    digitalWrite(24, 0);
    cout<< " " << endl;
}
```

```
frame [11] 158
frameCenter 166
frameEdge [ 161
frameFinal [ 162
frameFinalDu 163
frameGray [1 164
```

```
else If (result >=10 to result <20)
```

```
digitalwrite(21, 1);
digitalwrite(22, 1); //Accrual • I
digitalwrite(23, 0);
digitalwrite(24, 0);
cout<< "Forward" << endl;
```

To start Connect Raspberry Pi with Arduino after setup raspberry pi pins:

1-

```
4 const int LowL =7;
5
6 const int EnableR = 10;
7 const int HighR = 8;          //RIGHT SIDE MOTOR
8 const int LowR =9;
9
10 const int D0 = 0;           //Raspberry pin 21      LSB
11 const int D1 = 1;           //Raspberry pin 22
12 const int D2 = 2;           //Raspberry pin 23
13 const int D3 = 3;           //Raspberry pin 24      MSB
14
15 int a,b,c,d,data;
16
17 void Data()
18 {
19     a = digitalRead(D0);
20     b = digitalRead(D1);
21     c = digitalRead(D2);
22     d = digitalRead(D3);
23
24     data = 8*d+4*c+2*b+a;
25 }
26
27
28 void setup() {
29
30     pinMode(EnableL, OUTPUT);
31     pinMode(HighL, OUTPUT);
32     pinMode(LowL, OUTPUT);
33
34 }
```

```
34
35 pinMode(EnableR, OUTPUT);
36 pinMode(HighR, OUTPUT);
37 pinMode(LowR, OUTPUT);
38
39 pinMode(D0, INPUT_PULLUP);
40 pinMode(D1, INPUT_PULLUP);
41 pinMode(D2, INPUT_PULLUP);
42 pinMode(D3, INPUT_PULLUP);
43
44
```

Chapter 4

Lane detection

4.1 Introduction

Lane tracking is one of the most important tasks for autonomous vehicles. There are one or more lanes on each road, In a vehicle without lane tracking, there is no steering and this causes to accident



Fig4.1 road scene



fig4.2 lane detection

These road accidents can be reduced with the help of road lanes or white markers that assist the driver to identify the road area and non-road area. A lane is a part of the road marked which can be used by a single line of vehicles as to control and guide drivers so that the traffic conflicts can be reduced

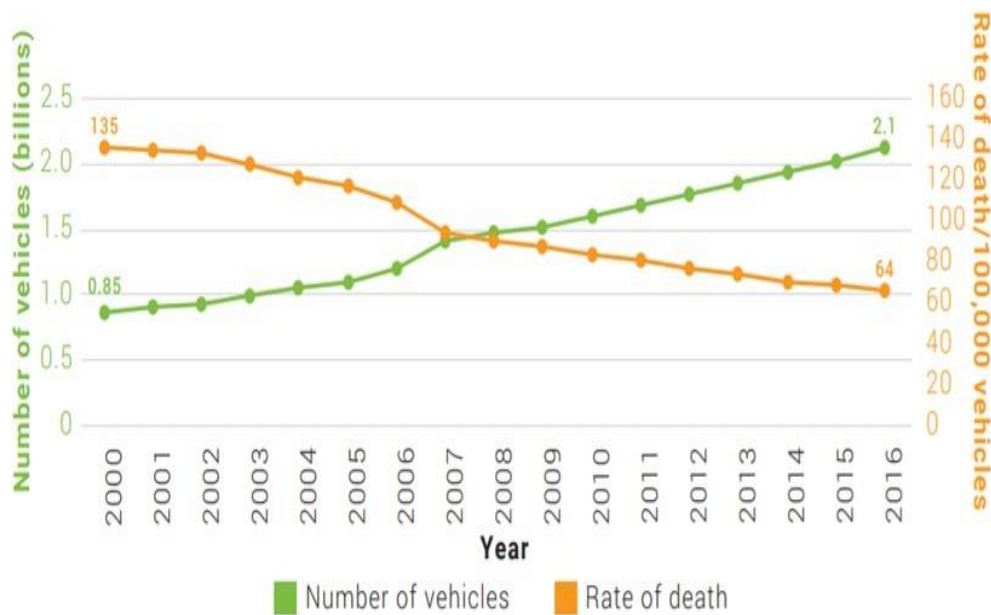


Fig 4.3 Number and rate of road traffic death per 100,000 population: 2000-2016

4.2 History

The first production lane departure warning system in Europe was developed by the United States company Iteris for Mercedes Actros commercial trucks. The system debuted in 2000, and is now available on most trucks sold in Europe.

In 2002, the Iteris system became available on Freightliner Trucks' North American vehicles. In both these systems, the driver is warned of unintentional lane departures by an audible rumble strip sound generated on the side of the vehicle drifting out of the lane. No warnings are generated if, before crossing the lane, an active turn signal is given by the driver.

4.3 Limitations of Lane keeping Systems in real cars

Modern lane departure warning systems are more reliable than earlier iterations of the technology, but even the most advanced examples have limitations. These systems often rely on visual information to track the relative position of a vehicle within its lane, so anything that obscures the lane markers will render the technology useless. That means you usually can't depend on your LDW or LKS in heavy rain, snow, or if there is excessive glare from the sun. Also lane keeping cannot be used in some other conditions because doing so may result in several accidents such as: roads with sharp edges, when entering a sharp curve into an interchange or junction, the stereo cameras' field of view is obstructed.

4.4 Prototyping of Lane-detection

Our prototype is a simulation of what happens in a real car by using some components.

4.4.1 Basic components:

1-Raspberry Pi

2- camera

Camera and raspberry pi are used as hardware.

car kit, breadboard, cables, battery and power bank were used for the operation of the car. to keep the power of the car uninterrupted, the battery was connected to the motors and the power bank was started to power the raspberry pi

4.4.2 Image Processing Techniques

Color images contain more information than gray level images. So canny edge detection process for color images has been examined. Grayscale method is one of these methods and the cost of detecting ROI in the image was reduced by the Grayscale image conversation method and the process was accelerated

While(1)

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

```
    Capture();
```

```
    Perspective();
```

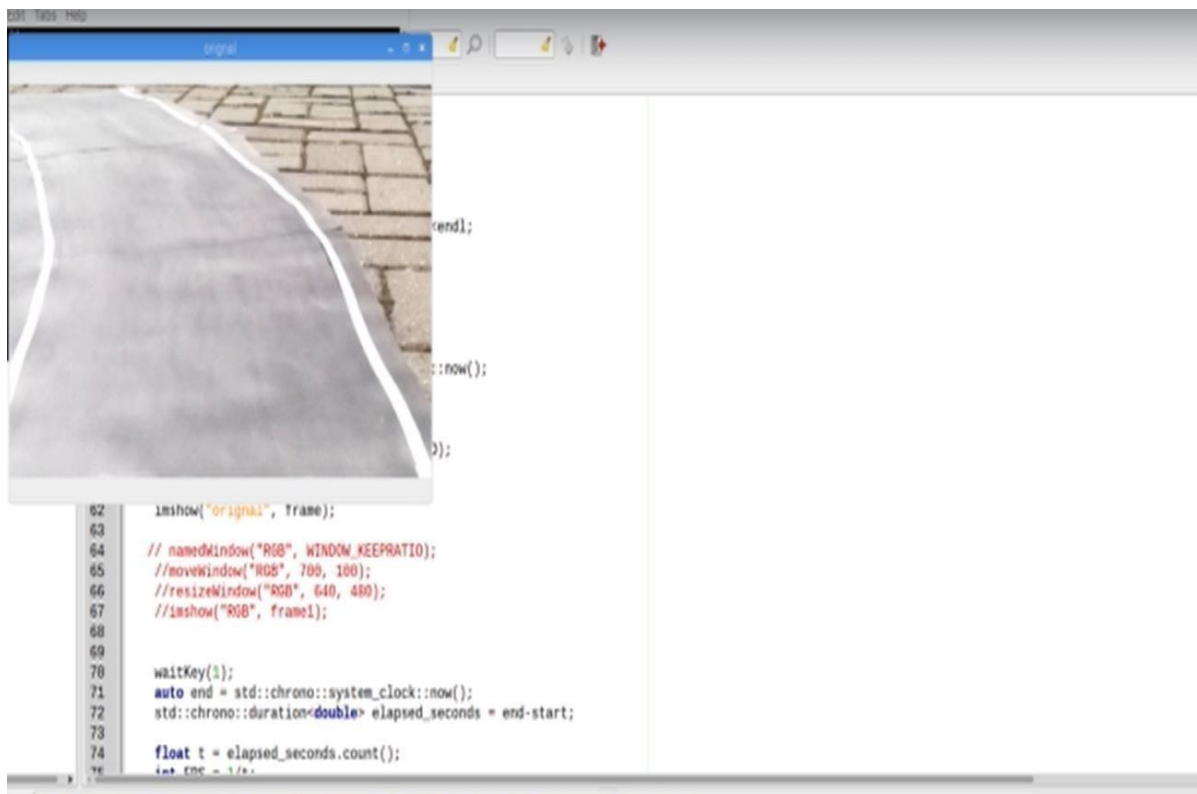
```
    Threshold();
```

```
    namedWindow("original", WINDOW_KEEPRATIO);
```

```
    moveWindow("original", 0, 100);
```

```
    resizeWindow("original", 640, 480);
```

```
    imshow("original", frame);
```



4.4.3 Finding Region of Interest (ROI)

The area of the lane is called the ROI. While some algorithms have an ROI up to the point where the lines can be detected by combining the lane lines in the image, some algorithms have methods for finding the ROI based on vanishing point detection Techniques. There are two common methods for the detection of ROI. Firstly, left and right lane lines are found. In the first method, the lane lines are stretched and intersect at one point. A triangular region is formed. The area within the triangular region up to the region where the lane is detected forms a rectangular region. This field returns the ROI. The deviation difference of the angles and the obtained region is calculated, and the direction of the path is calculated. Another method is to determine the ROI by determining the skyline. Horizon line is the line where the earth globe and sky intersect when viewed in nature. With the determination of the horizon line, the lane lines are also combined and the intersecting area shows the ROI

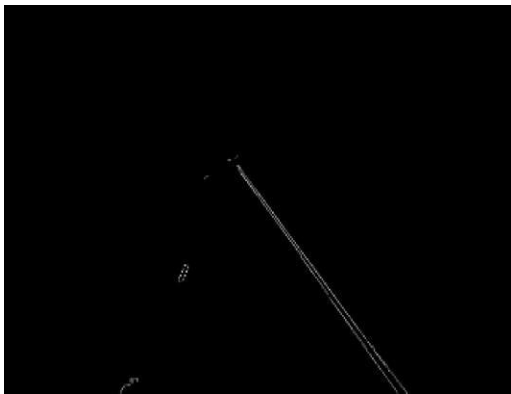


FIG 4.6 Detected interested lane

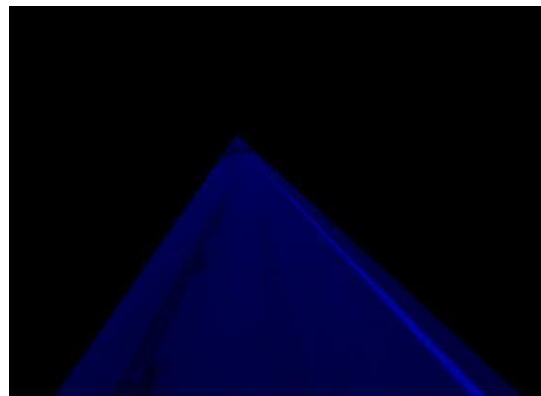
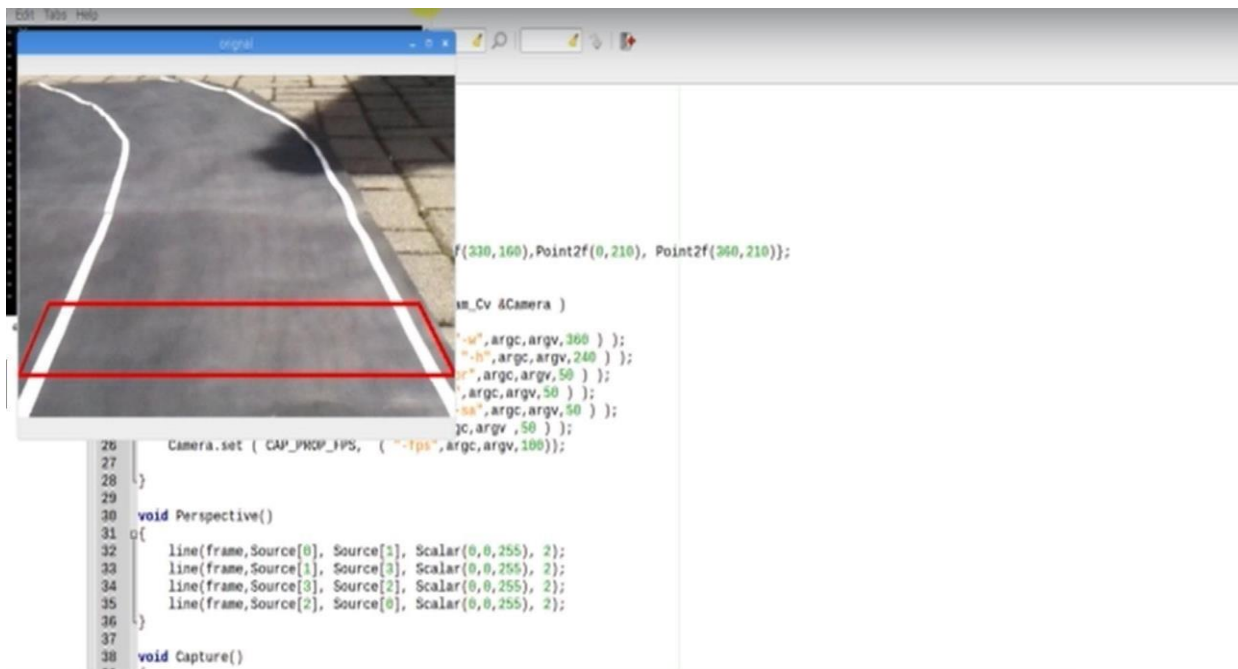


fig4.7 ROI of path image

4.4.3.1 Implementation and testing

```
Camera.cpp X
6
7 using namespace std;
8 using namespace cv;
9 using namespace raspicam;
10
11 Mat frame;
12
13 RaspiCam_Cv Camera;
14
15 Point2f Source[] = {Point2f(25,160),Point2f(230,160),Point2f(0,210), Point2f(360,210)};
16
17
18 void Setup ( int argc, char **argv, RaspiCam_Cv &Camera )
19 {
20     Camera.set ( CAP_PROP_FRAME_WIDTH, ( "-w",argc,argv,360 ) );
21     Camera.set ( CAP_PROP_FRAME_HEIGHT, ( "-h",argc,argv,240 ) );
22     Camera.set ( CAP_PROP_BRIGHTNESS, ( "-br",argc,argv,50 ) );
23     Camera.set ( CAP_PROP_CONTRAST, ( "-co",argc,argv,50 ) );
24     Camera.set ( CAP_PROP_SATURATION, ( "-sa",argc,argv,50 ) );
25     Camera.set ( CAP_PROP_GAIN, ( "-g",argc,argv,50 ) );
26     Camera.set ( CAP_PROP_FPS, ( "-fps",argc,argv,100) );
27 }
28
29
30 void Perspective()
31 {
32     line(frame,Source[0], Source[1], Scalar(0,0,255), 2);
33     line(frame,Source[1], Source[3], Scalar(0,0,255), 2);
34     line(frame,Source[3], Source[2], Scalar(0,0,255), 2);
35     line(frame,Source[2], Source[0], Scalar(0,0,255), 2);
36 }
37
38 void Capture()
```



4.4.4 Canny edge detection

is a method of edge detection. The lanes are lines and have edges. detection algorithm in images with a grayscale color space is closer to the edge information in the actual image.



Fig 4.8Original frame

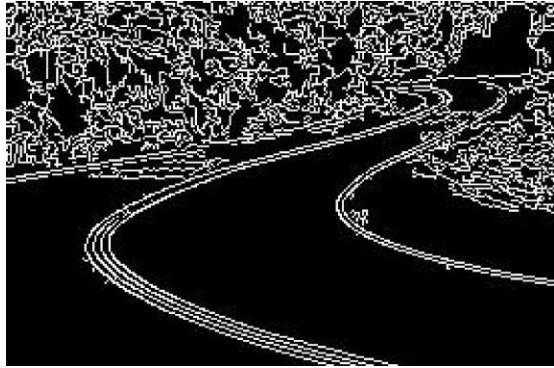
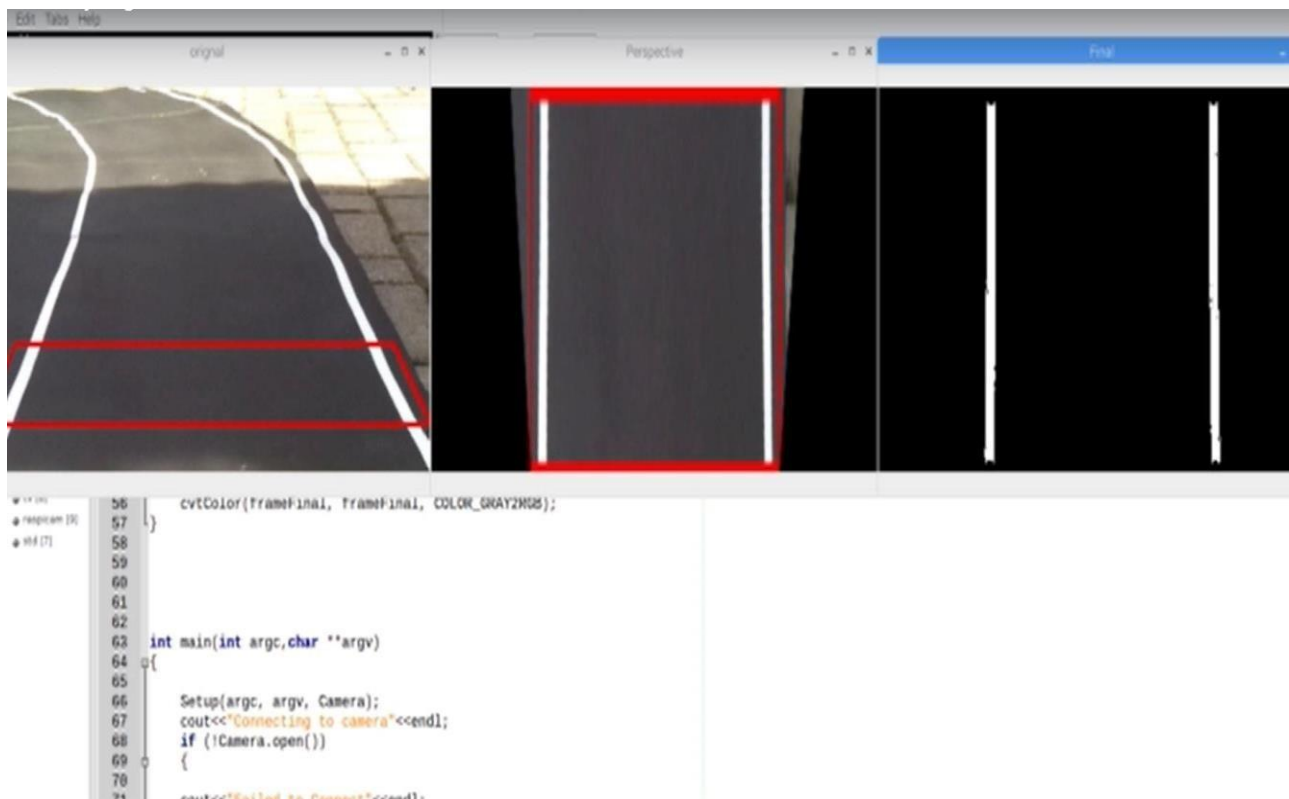


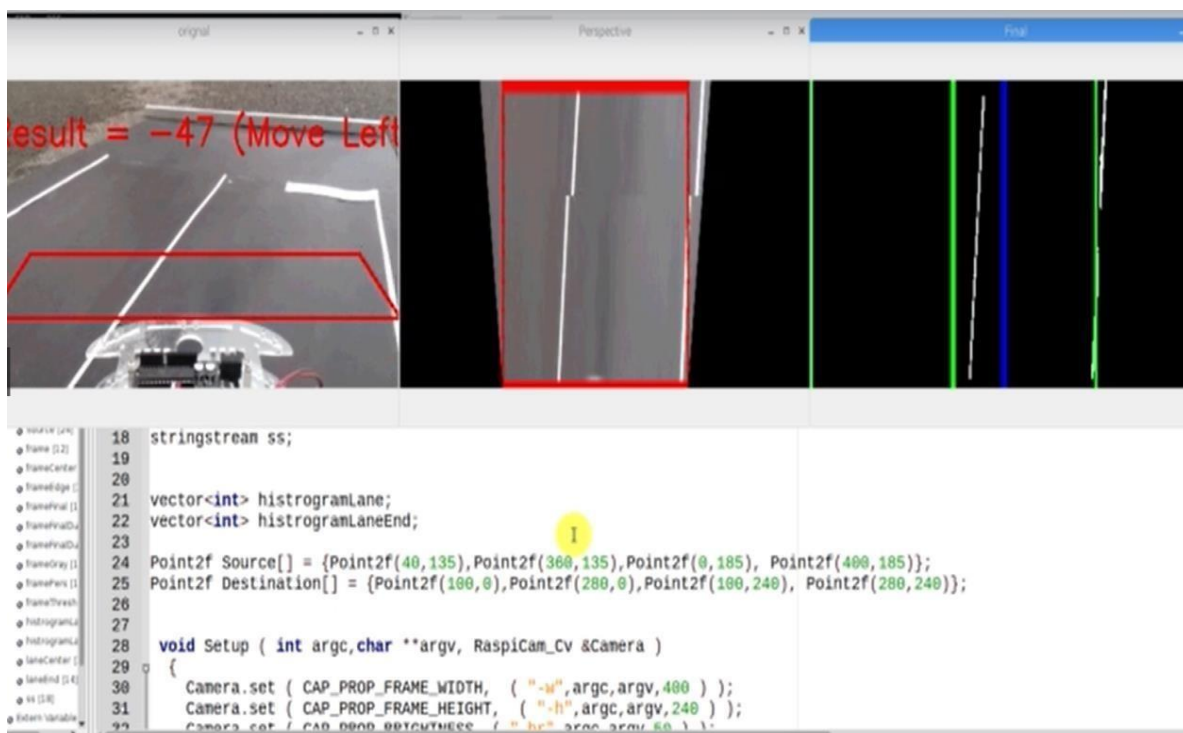
Fig 4.9canny edge frame

4.4.4.1 Implementation & testing

```
view document project build tools help
Camera.cpp X
36 }
37
38 void Perspective()
39 {
40     line(frame,Source[0], Source[1], Scalar(0,0,255), 2);
41     line(frame,Source[1], Source[3], Scalar(0,0,255), 2);
42     line(frame,Source[3], Source[2], Scalar(0,0,255), 2);
43     line(frame,Source[2], Source[0], Scalar(0,0,255), 2);
44
45     Matrix = getPerspectiveTransform(Source, Destination);
46     warpPerspective(frame, framePers, Matrix, Size(360,240));
47 }
48
49
50 void Threshold()
51 {
52     cvtColor(framePers, frameGray, COLOR_RGB2GRAY);
53     inRange(frameGray, 240, 255, frameThresh);
54     Canny(frameGray, frameEdge, 600, 700, 3, false);
55     add(frameThresh, frameEdge, frameFinal);
56     cvtColor(frameFinal, frameFinal, COLOR_GRAY2RGB);
57 }
58
59
60
61
62
63 int main(int argc, char **argv)
64 {
65     Setup(argc, argv, Camera);
66     cout<<"Connecting to camera"<<endl;
67     if (!Camera.open())
68     {
69
70
71         cout<<"Failed to connect"<<endl;
```



4.4.5 Lane end & U Turn



4.5 Final 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> histogramLane;
vector<int> histogramLaneEnd;
```

```
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 histogram function only  
    cvtColor(frameFinal, frameFinalDuplicate1,  
COLOR_RGB2BGR); //used in histogram function only  
}
```

```
void Histogram()
```

```
{  
    histogramLane.resize(400);  
    histogramLane.clear();  
  
    for(int i=0; i<400; i++)    //frame.size().width = 400  
    {  
        ROILane = frameFinalDuplicate(Rect(i,140,1,100));  
        divide(255, ROILane, ROILane);  
    }
```



```

    histogramLane.push_back((int)(sum(ROILane)[0]));
}

    histogramLaneEnd.resize(400);
    histogramLaneEnd.clear();
    for (int i = 0; i < 400; i++)
    {
        ROILaneEnd = frameFinalDuplicate1(Rect(i, 0, 1,
240));
        divide(255, ROILaneEnd, ROILaneEnd);

        histogramLaneEnd.push_back((int)(sum(ROILaneEnd)[0]
));

    }
    laneEnd = sum(histogramLaneEnd)[0];
    cout<<"Lane END = "<<laneEnd<<endl;
}

void LaneFinder()
{
    vector<int>:: iterator LeftPtr;
    LeftPtr = max_element(histogramLane.begin(),
histogramLane.begin() + 150);
    LeftLanePos = distance(histogramLane.begin(), LeftPtr);

    vector<int>:: iterator RightPtr;
    RightPtr = max_element(histogramLane.begin() +250,
histogramLane.end());
    RightLanePos = distance(histogramLane.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("original", WINDOW_KEEPRATIO);  
moveWindow("original", 0, 100);  
resizeWindow("original", 640, 480);  
imshow("original", 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);  
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;
```

```
}
```


Chapter 5

Stop sign detection

5.1 Introduction

The goal of this section is to detect and track traffic signs in video streams captured by a front-facing camera mounted on a moving car. Given a video stream captured by a car mounted, front-facing camera, recognize traffic signs such as the STOP sign that appear in the images and track them over a sequence of frames.

This is an important problem because it has great utility in automated or assisted driving applications. Traffic signs occupy an important position in the road traffic systems. The main function of the traffic signs is to display the contents that need to be noticed in the current road sections, to prompt the drivers in front of the road the danger and difficulty in the environment, to warn the driver to drive at the prescribed speed, to provide a favorable guarantee for safe driving. Therefore, the detection and identification of traffic signs is an especially important research direction, which is of great significance to prevent road traffic accidents and protect the personal safety of drivers

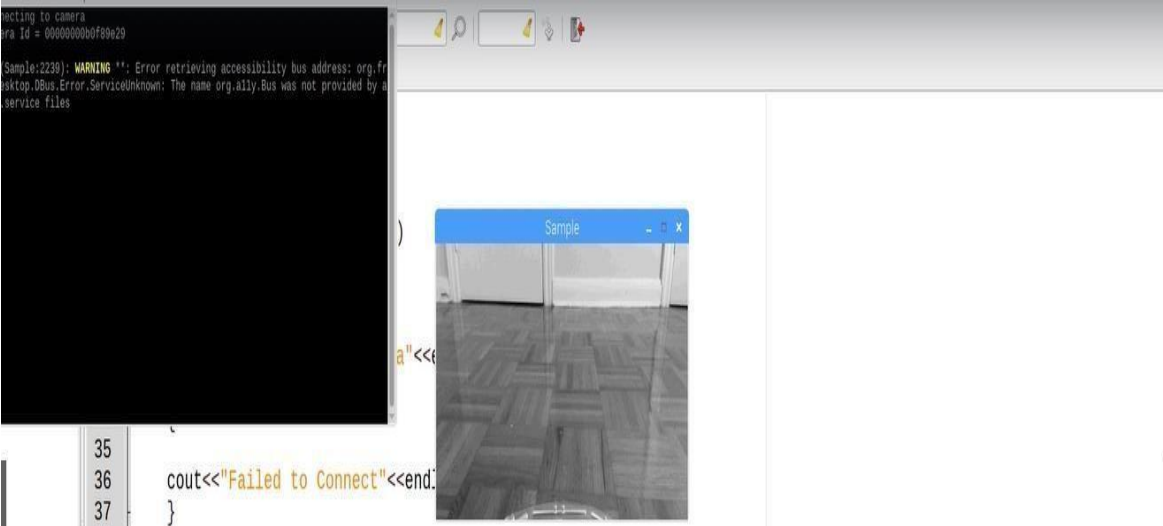
Road traffic signs are divided in to two major categories of main signs and auxiliary signs.

The main sign is divided into warning signs, prohibition signs, mandatory signs, guide signs, tourist signs and road construction and safety signs. Among them, prohibition signs mainly played a role in banning certain kinds of behavior, a total of 43 categories. Mandatory signs indicate the role of vehicles which is placed in the need to indicate vehicles, near the intersection. Warnings are to alert drivers, vehicles pedestrians to beware of dangerous targets, a total of 45 categories. They all play a key role in traffic signs. Among them, the most common speed limit signs, and the prohibition of left and right turning signs are of great significance for safe driving of drivers and therefore are the focus of the current research on traffic sign recognition.

The aim of this part is to develop a better system for the automatic detection and recognition of traffic signs with high accuracy and robustness under various complicated situations and disturbances.

5.2 C++ code to Capture & Save Images:

```
Connecting to camera  
Camera Id = 000000000f0e9e29  
  
Sample(2238): WARNING **: Error retrieving accessibility bus address: org.freedesktop.Obus.Error.ServiceUnknown: The name org.a11y.Bus was not provided by a service files  
  
a" <<  
  
35  
36 cout<<"Failed to Connect"<<endl;  
37 }  
38  
39 cout<<"Camera Id = "<<Camera.getId()<<endl;  
40  
41 for (int i=0; i<40; i++)  
42 {  
43     Camera.grab();  
44     Camera.retrieve(frame);  
45     cvtColor(frame,frame, COLOR_BGR2GRAY);  
46     imshow("Sample", frame);  
47     imwrite("Stop"+to_string(i)+".jpg" , frame);  
48     waitKey();  
49 }  
50  
51 return 0;  
52 }
```



5.3 Slight Street Sign Modifications Can Completely Fool Machine Learning Algorithms

Minor changes to street sign graphics can fool machine learning algorithms into thinking the signs say something completely different



Figure 5.3.1 Stop Sign

It is exceedingly difficult, if not impossible, for us humans to understand how robots see the world. Their cameras work like our eyes do, but the space between the image that a camera captures and actionable information about that image is filled with a black box of machine learning algorithms that are trying to translate patterns of features into something that they are familiar with. Training these algorithms usually involves showing them a set of different pictures of something (like a stop sign), and then seeing if they can extract enough common features from those pictures to reliably identify stop signs that are not in their training set.

This works well, but the common features that machine learning algorithms come up with generally are not “red octagons with the letters S-T-O-P on them.” Rather, they are looking features that all stop signs share, but would not be in the least bit comprehensible to a human looking at them. If this seems hard to visualize, that is because it reflects a fundamental disconnect between the way our brains and

artificial neural networks interpret the world.

The upshot here is that slight alterations to an image that are invisible to humans can result in wildly different (and sometimes bizarre) interpretations from a machine learning algorithm. These "adversarial images" Have required relatively complex analysis and image manipulation, but a group of researchers from the University of Washington, the University of Michigan, Stony Brook University, and the University of California Berkeley have just published a paper showing that it is also possible to trick visual classification algorithms by making slight alterations in the physical world. A little bit of spray paint or some stickers on a stop sign were able to fool a deep neural network-based classifier into thinking it was looking at a speed limit sign 100 percent of the time.

Here is an example of the kind of adversarial image we are used to seeing

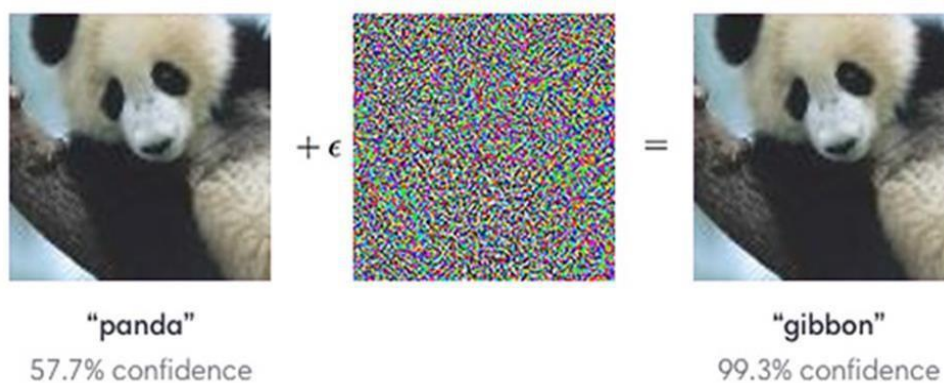


Figure 5.3.2 panda, when combined with an adversarial input, can convince a classifier that it is looking at a gibbon.

Obviously, it is totally, uh, obvious to us that both images feature a panda. The differences between the first and third images are invisible to us, and even when the alterations are shown explicitly, there is nothing in there that looks all that much like a gibbon. But to a neural network-based classifier, the first image is probably a panda while the third image is almost definitely a gibbon. This kind of thing also works with street signs, causing signs that look like one thing to us to look like something completely different to the vision system of an autonomous car, which could be extremely dangerous for obvious reasons.

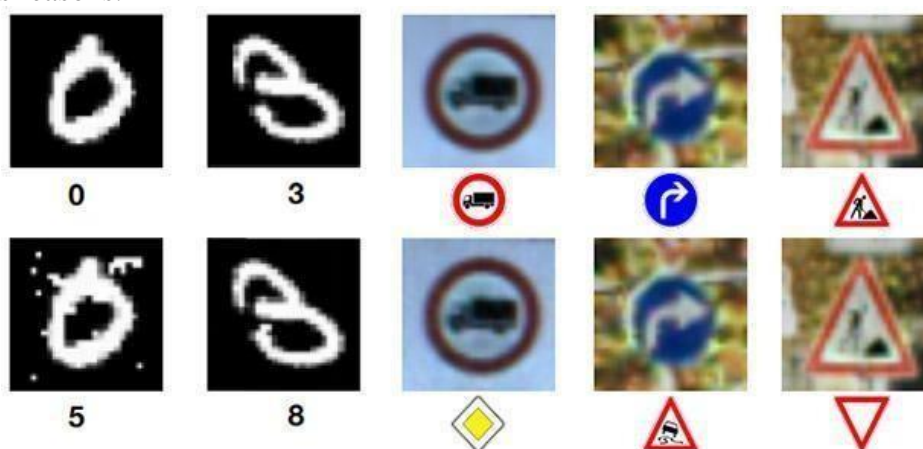


Figure 5.3.3 Top row shows legitimate sample images, while the bottom row shows adversarial sample images, along with the output of a deep neural network classifier below each image

Adversarial attacks like these, while effective, are much harder to do in practice, because you usually do not have direct digital access to the inputs of the neural network you are trying to mess with. Also, in the context of an autonomous car, the neural network has the opportunity to analyze a whole bunch of images of a sign at different distances and angles as it approaches

And here are two attacks that are easier to manage on a real-world sign, since they are stickers rather than posters:



Because the stickers have a much smaller area to work with than the posters, the perturbations they create have to be more significant, but it is certainly not obvious that they are not just some random graffiti. And they work almost as well. According to the researchers:

In order to develop these attacks, the researchers trained their own road sign classifier in TensorFlow using a publicly available, labeled dataset of road signs.

5.4 HAAR CLASSIFICATION

Haar feature-based Cascade Classifier initially proposed by Viola and Jones classifies objects using a series of edge, line, and center-surround features that scan throughout the image to construct ROI features. The name cascade means the resultant classifier consists of several simpler classifiers that are applied to the image one at a time in the order of their classification effectiveness; to speed up the detection process, when earlier classifiers fail to produce a match, the remaining classifiers are no longer applied. We implemented Haar cascade detection for stop signs. However, a problem is that because of the small size of the traffic signs, the window size has to be small to detect any stop signs. We used a window of 12×12 pixels. We observed that while an 8×8 window can only detect about 30%

of the signs, a 12×12 window with the picture magnified by $1.5 \times$ is able to achieve recognize over 80% of the signs. We suspect that this behavior may be due to the step sizes built into the OpenCV detect Multiscale function. In order to evaluate the efficiency of cascade classification, we evaluated the following types of detectors:

- Number of stages varying from 12 to 20
 - Miss probability per stage varying from 0.995 to 0.9995
 - LBP and Haar features
 - Initial magnification of $\{1, 1.1, 1.2, 1.3, 1.4\}$
- We use 500 positive images and 750 negative images at each stage of the detector.

After the training period, we evaluate the false alarm and the hit rates of all the detectors using a separate

Validation dataset. The miss rate vs false positives per frame (FPPF) scatter plot of the approaches.

We observed that the following combination works well.

- LBP features
 - 14 stages
 - miss probability per stage of 0.9975
 - initial magnification of 1.3
- This configuration achieves 78.7% accuracy over the full dataset.

The false positives per frame (FPPF) of this approach is 2.65, which is extremely high. Later, we evaluate how to bring the number of false positives down by using color information. Figure 3 shows

some of the example scenarios that arise in traffic sign recognition, and the performance of the detector in these scenarios.

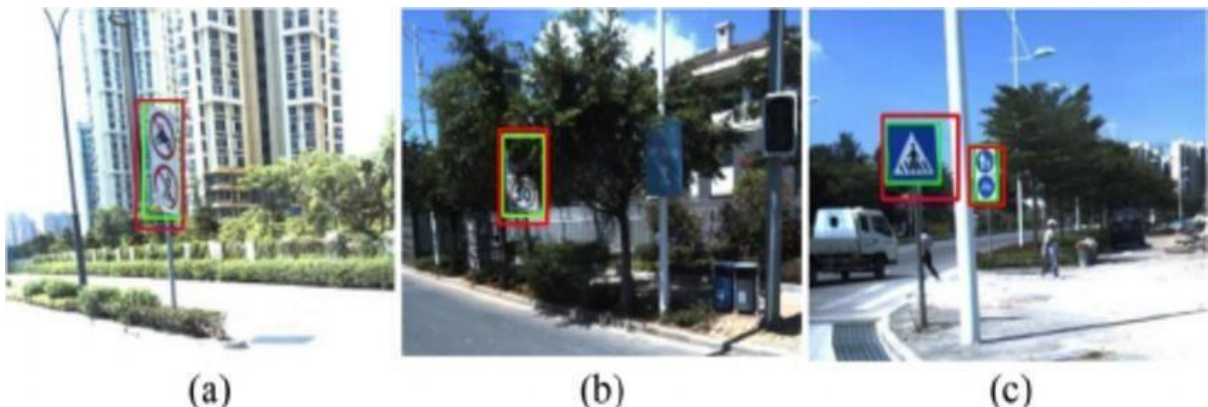
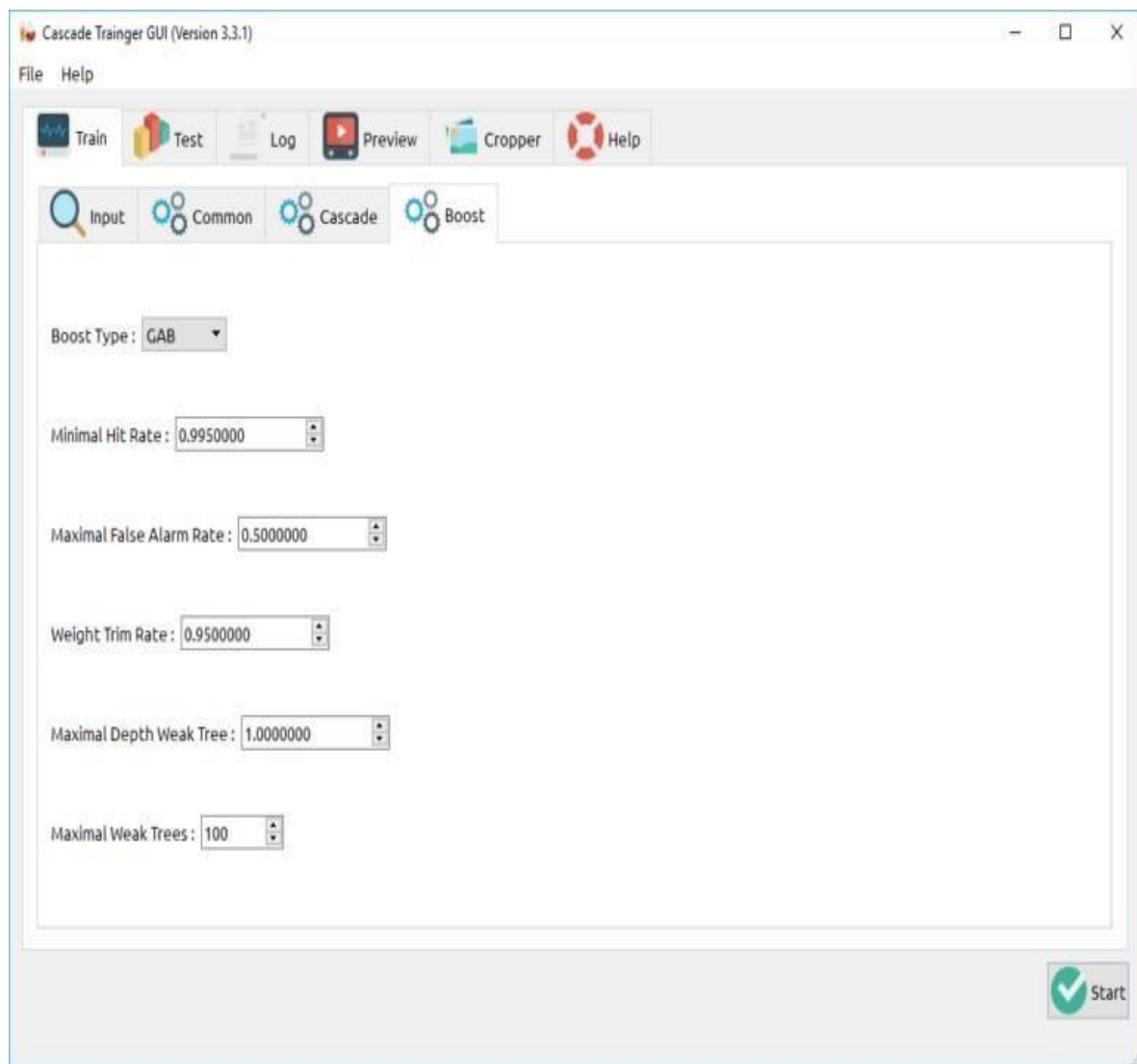
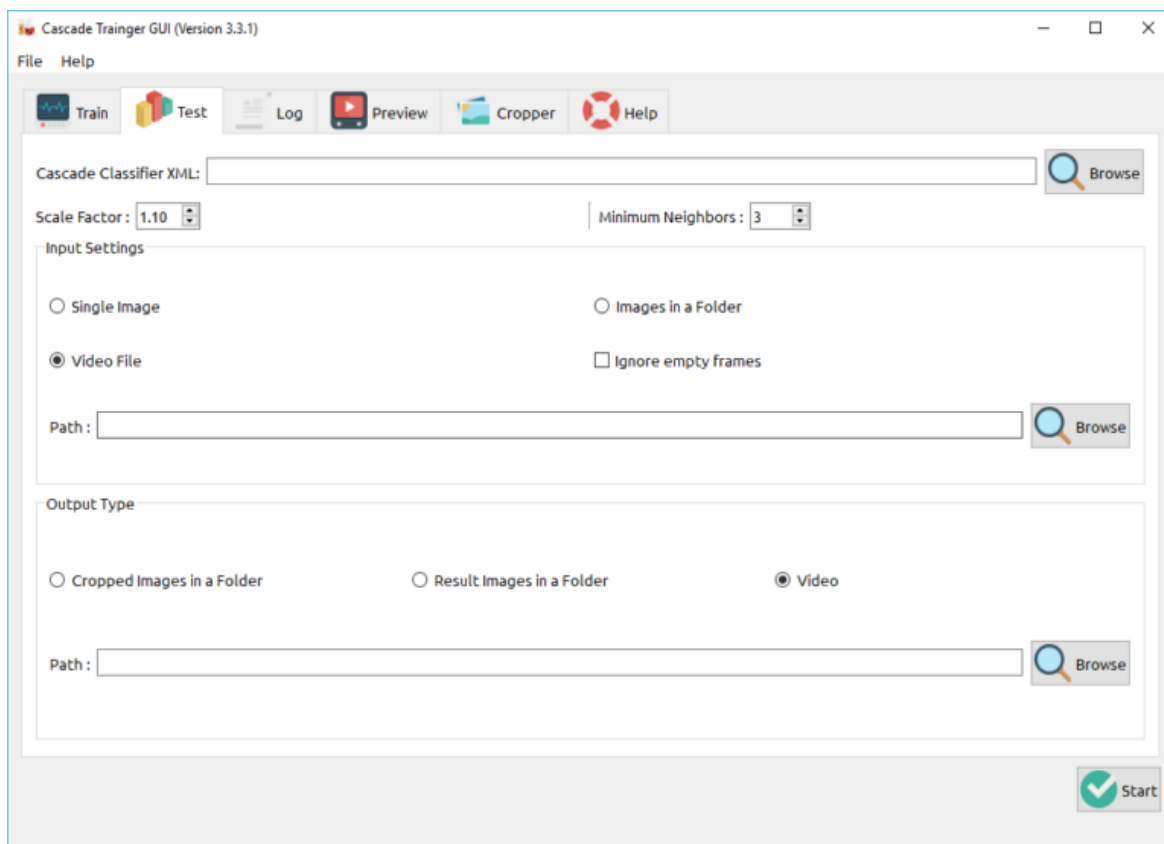
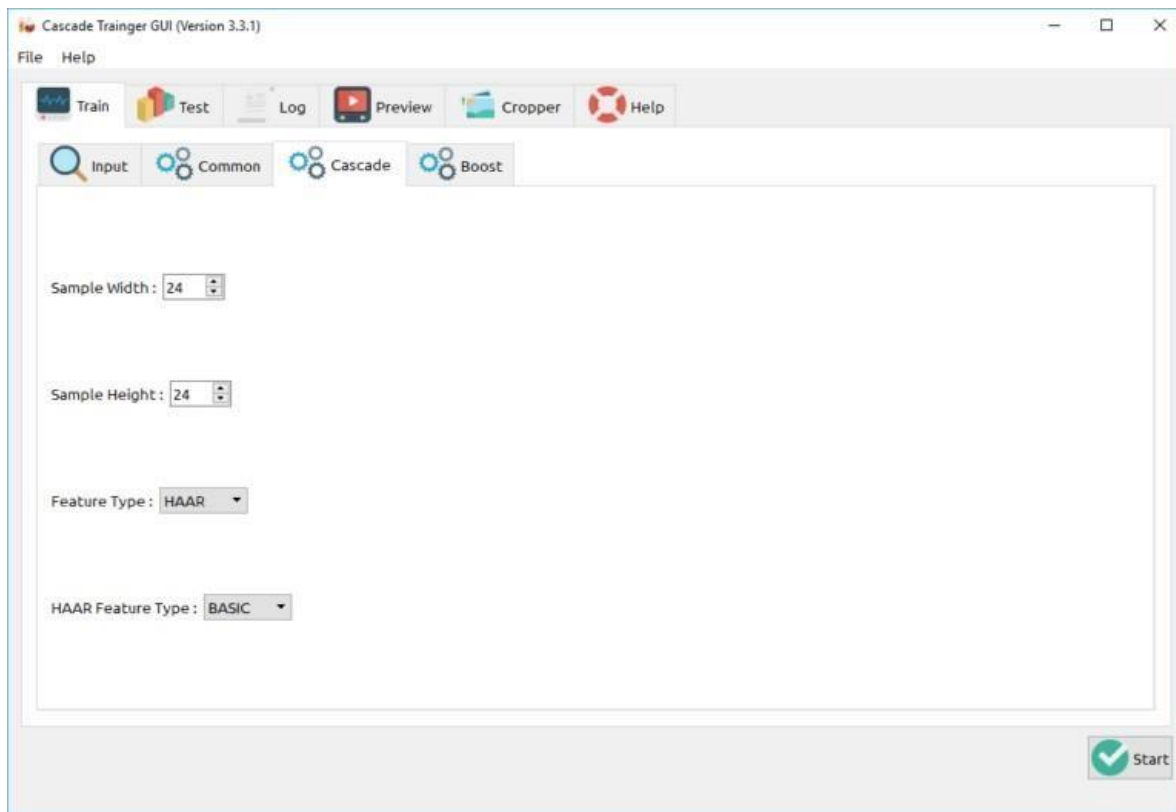


Fig5. 3: Various possible detection scenarios.

5.4.1 Training of Haar Cascade Model for Stop Sign:



5.4.2 Cascade Training Software and Image Cropping:



5.5 Stop Sign Detection on Raspberry Pi3:



Fig 5.5.1 Raspberry Pi 3 Model B Board

5.5.1 Capturing Images

A Raspberry Pi can capture a sequence of images rapidly by utilizing its video- capture port with JPEG encoder. However, several issues need to be considered:

- The video-port capture is only capturing when the video is recording, meaning that images may not be in a desired resolution/size all the time (distorted, rotated, blurred etc.).
- The JPEG encoded captured images do not have exit information (no coordinates, time, noexchangeable).
- The video-port captured images are usually "more " fragmented" than the still port capture images, so before we go through pre-processed images, we may need to apply more denoising algorithms.
- All capture methods found in OpenCV (capture, capture continuous, capture sequence) have to be considered according to their use and abilities. In this project, the capture sequence method was chosen, as it is the fastest method by far.

Using the capture sequence method our Raspberry Pi camera can capture images in rate of 20fps at a 640×480 resolution. One of the key issues with the Raspberry Pi when capturing images rapidly is bandwidth. The I/O bandwidth of

Raspberry Pi is extremely limited, and the format we are pulling pictures makes the process even less efficient. In addition, if the SD card size is not large enough, the card will not be able to hold all pictures that are being captured by camera port, leading to cache exhaustion.

5.5.2 Multithreading in Capturing and Processing Images

Because of limited I/O Bandwidth of the Raspberry Pi, structuring the multithreading is an extremely important initial step of the pre-processing algorithm for images. To succeed this first we need to capture an image from the video-port then process. Raspberry Pi maintains a queue of images and process them as the captured images come in. Most importantly, the Raspberry Pi image processing algorithm must run faster than the frame rate of capturing images, in order to not to stall the encoder

5.5.3. Detecting Speed Signs

When we look at the pictures of speed signs, the most defining feature of a speed sign is rectangular shape with mostly round edges. Before finding the rectangles in a captured image, we retrieve the contours, thus the shape detection algorithm employed loops through a subset of contours and checks if the contour shape is rectangle. The shape detection is based on the OpenCV's implementation preceded by edge detection. To prevent the noise from being mistaken as edges, and produce wrong results, the noise must be reduced to certain level. Thus, the images are smoothed by applying Gaussian Filter. In a two-dimensional space an isotropic Gaussian form $F(x, y)$ equals to(1)

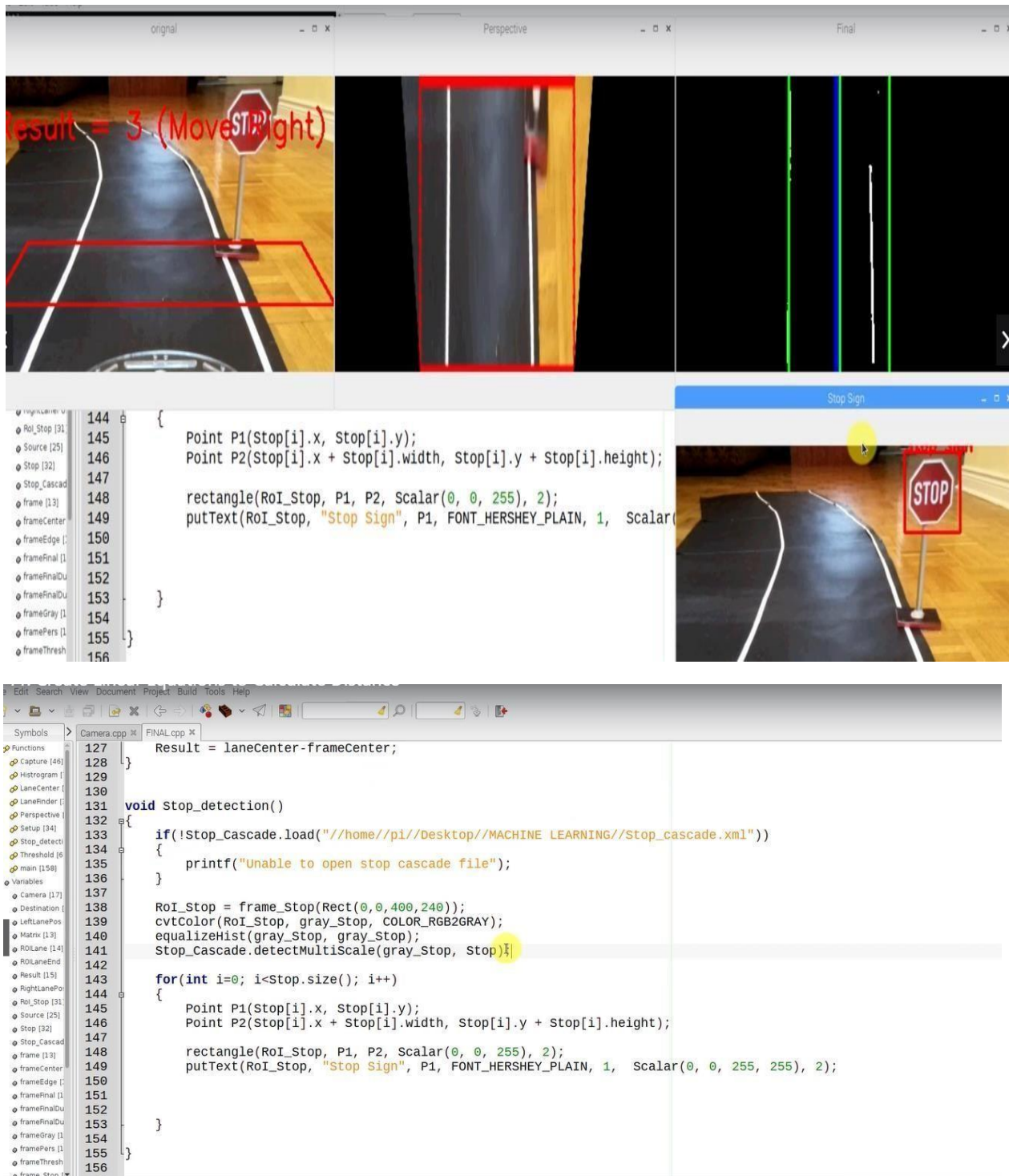
5.5.4 Open CV Contour Features and Edge Detection

The purpose of the edge detection is to significantly reduce the amount of data in the image by converting the image into binary format. Even though Canny edge detection is quite an old method, it has become one of the standard edge detection algorithms. In doing image processing and especially shape analysis it is often required to check the objects shape, and depending on it perform further processing of a particular object. In our application it is important to find the rectangles in each of frames as these may potentially correspond to road speed signs. This shape detection must be done at least once in every 40 frames to ensure close to real processing. Once we check all the contours retrieved, we should look for closed loops then that closed loop should meet the following conditions to become a rectangle. Contour approximation is an important step for finding desired rectangles, since because of distortions other issues in the image perfect rectangle may not be visible. The contour approximation might be in this case better choice for finding convex hulls. After this step we can approximate the rectangular shape in the captured image as shown in Figure 2. Once the individual regions for the signs are identified, they are rotated to fit a common alignment and then OCR is performed. To align the signs that first we find 4 max and min points in a rectangular shape



Figure 5.4. Bounding rectangles for a set of speed signs

5.5.5 Implementation of Linear Equations to Calculate Distance



chapter 6

Traffic Signs Recognition and Distance Estimation

6.1. Overview

Traffic signs are integral component of road transport infrastructure. They deliver essential driving information to road users, which in turn require them to adapt their driving behavior to road regulations. In this study, the deep learning model is implemented with You Only Look Once (YOLO) based on active learning approach to minimize the size of the labeled dataset and provide higher accuracy. This is an efficient approach using a single monocular camera to perform real-time traffic signs recognition and distance estimation between traffic sign and camera pose. YOLO is one of the faster object detection algorithms, and it is a particularly desirable choice when real-time detection is needed, without loss of too much accuracy. The active learning algorithm will bridge the gap between labeled and unlabeled data, thus, only queries the samples that would lead to increase the accuracy. The aim of this work is to alarm and notify the drivers without having them to switch their focus. The results of the performed experiments show that about 97% recognition accuracies could be achieved with Realtime capability in different real-world scenarios.

6.1.1 Traffic signs recognition.

Millions of people are seriously injured in vehicle accidents every year. Usually, road accidents are caused by negligence, ignorance of the rules and traffic signs. All signals contribute to keep order in road traffic and are also intended to reduce the number and severity of road accidents. The traffic signs are placed in predefined areas to ensure the driver's safety.

Nowadays, the recognition of traffic signs has been the subject of interest for automotive community and is even a valuable characteristic of autonomous vehicles. The objective of automatic traffic sign recognition systems is to specify the locations and sizes of traffic signs in natural scene stream images (detection task) and then classify the detected traffic signs into their specific classes (classification task).

Through these functions (detection and classification tasks), the systems can control and alert the driver to avoid hazards. Traffic signs contain important useful information that the driver may ignore due to driving fatigue or searching for address needs.

These improvements, although having a positive impact, face several external non-technical challenges, such as light variations, size and weather conditions, and signs in destroyed conditions, which can ultimately affect the performance of traffic sign recognition systems.

Figure1 shows some examples of non-ideal invalid and challenging traffic signs.



Figure 1 – Shows non-identical traffic signs.



Figure 2 -- (a) Partial occlusion, blurred traffic sign, (c) destroyed traffic sign, (d) multiple traffic signs displayed at the same time.

The main concern with traffic sign recognition system is not how to recognize or identify a traffic sign with a high reminder in a still image. In fact, it is how to achieve high accuracy in high-resolution live video streaming with real-time capability [2][3]. In order to demonstrate the problem of false detection, a traffic sign system at least with 30 frames per second (108,000 frames per 1h video) was considered. Under the assumption that the system has a false positive accuracy of 1%, this means 17 false alarms were detected every minute (1071 in 1 hour) and 1 positive and 68 false alarms within 4 minutes. Traffic signs have distinctive features which can be classified into separate sub-categories. These are divided into five main categories according to shape and color: Warning signs (red triangle), prohibition signs (red round), reservation signs (rectangular blue), mandatory signs (circular blue) and temporary signs (yellow triangle). In this study, only the speed signs and the stop sign are considered as use case studies to evaluate our approach. In the proposed approach, the system will supply the vehicle driver with real-time traffic sign information, the estimated distance between the monocular camera mounted

on the vehicle, and the detected traffic sign. Then the system will notify the vehicle driver if the speed of the vehicle does not match the recognized traffic sign.

6.1.2 Recognizing Traffic Lights with Deep Learning



Figure 3.3 Red-yellow – green Traffic Lights

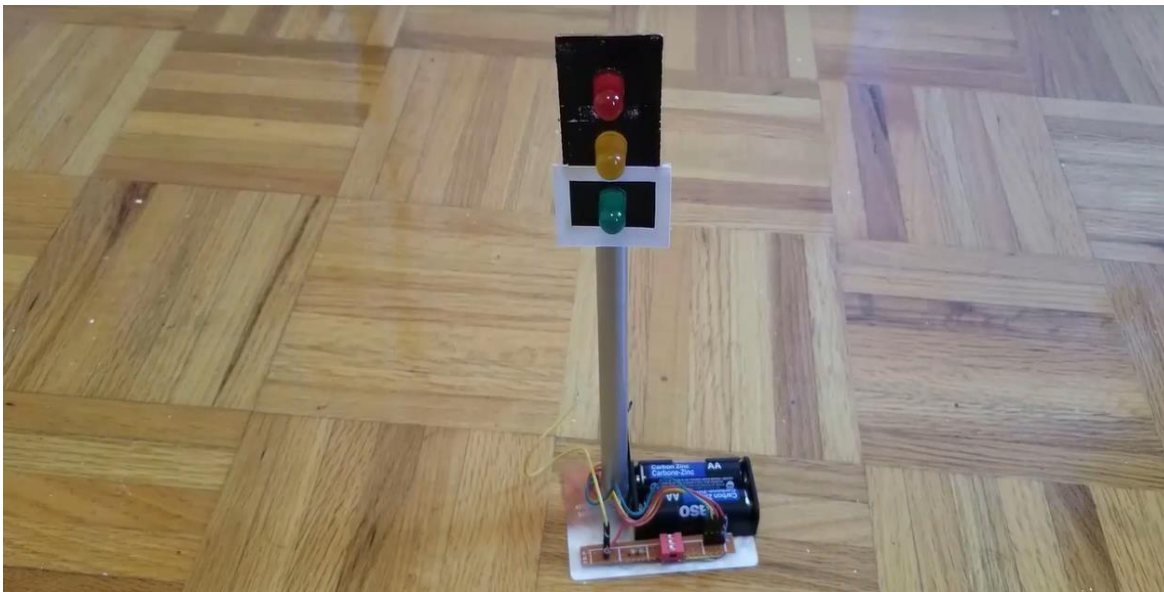
1.1.3 Training Data

All the training samples are taken from the data collected during the summer. Input data to the classifier are obtained from the candidate selection procedure described in A, and the classifier output goes to the tracking algorithm for further processing. Thus, evaluation of the classifier is independent to the candidate selection or the post-processing (tracking). The classifier is trained to distinguish true and false traffic lights, and to recognize the types of the traffic lights.

OpenCV is used for SVM training, which chooses the optimal parameters by performing 10-fold cross-validation. The positive samples, which contain the traffic lights, are manually labeled, and extracted from the data set images. The negative samples, such as segments of trees and vehicle taillights, are obtained by applying the candidate selection procedure over the traffic-light-free images.

The green lights and red lights are classified separately. For green lights, there are three types of based on their aspect ratios. The first type is called Green ROI-1, which contains one green light in each image and its aspect ratio is approximately 1:1. The second type is called Green ROI-3. It contains the traffic light holder area which has one green light and two off lights, and its aspect ratio is approximately 1:3. The third type is called Green ROI-4. e

Traffic Light Model



```

1 #include <opencv2/opencv.hpp>
2 #include <raspicam_cv.h>
3 #include <iostream>
4 #include <chrono>
5 #include <ctime>
6 #include <wiringPi.h>
7
8 using namespace cv;
9 using namespace CV;
10 using namespace raspicam;
11
12 Mat frame;
13
14
15 void Setup( int argc, char**argv, easytamrv\taera)
16 {
17     tacera, stt ( IAPPEOPFPafifiâOT4 ( "-w",argc,argv,400 ) );
18     tactra.set(OP_PsOP_FPAfifi_hfiIG8I, ( "-h",argc,argv,240 ) );
19     Camera.set CAP_PROP_BRIGHTNESS (argc,gv 50 );
20     camera.setSAP.PROP.c0\BAS,( ,argc,args,SO),
21     rariera, set (SAPPROP_SATJRA!IO#, ( "-sa",argc,gv 50 ) );
22     camera.set(CAPPROP_OAU "-g",argc,argv ,50 );
23     Camera.set ( CAP_PROP_FPS, ( "-fps",argc,argv,0));
24 }
25
26
27
28 int main( int argc, char** argv)
29 {

```

```

' (Sample:5620): WARNING **: Error retrieving accessibility bus address: org.freedesktop.DBus.Error.ServiceUnknown: The name org.a11y.Bus was not provided by any .service files
)
a"<<endl;

Sample - x failed to Connect"<<endl;

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

i=0; i<60; i++)

ra.grab();
ra.retrieve(frame);
cvtColor(frame, frame, COLOR_BGR2GRAY);

46 imshow("Sample", frame);
47 imwrite(" "+to_string(i)+".jpg", frame);
48 cv::waitKey(1);
49 }
50
51 return 0;
52 }

```

Chapter 7

Conclusion

The development of a self-driving car with the lane line detection system using images processing technique found that the system can be operated by taking images from the camera on the self-driving car and processed by the Raspberry Pi board using image processing from the car camera to detect and track the road. From the algorithm tests for 20 times found that the real-time lane line detection system can work well. The system is 90 percent accurate in an environment where there are no objects or obstructions, the road is clearly seen and the light not too bright, so the system can detect the color code on the lane line as well as can great track of traffic lane. The researcher has guidelines for further improvement by developing the system so it can be used in the self-driving car which helps driving on the road more safety and reduce the accident caused by the negligence of the driver. Due to limited computation power of Raspberry Pi, complex techniques were not chosen despite their availability within the OpenCV libraries. In order to keep Raspberry Pi running smoothly.

Achieved features: -

- 1-** Lane detection
- 2-** Lane-keeping
- 3-** Stop sign detection
- 4-** Traffic lights detection

References:

Chapter 1

1. The Pathway to Driverless Cars. Claire Perry
2. wordpress.com
3. theguardian.com

Chapter 2

- 1- store.fut-electronics.com/
- 2- Graduation Project Autonomous Driving System

Chapter 3

- 1-arduino.cc/en/Main/ArduinoBoardYun
- 2-Twilio.com
- 3-Electrical engineer.com

Chapter 4

- 1- Chassis Systems Control Adaptive Cruise Control: More comfortable driving
- 2- Electronics|projects|focus/The Working Principle of a PID Controller 4-
Playground.arduino.cc/Code/PIDLibrary
- 3-Modern Control Engineering Fifth Edition/Katsuhiko Ogata

[4] Bahlmann, C., Zhu, Y., Ramesh, V., Pellkofer, M., Koehler, T., “A system for traffic sign detection, tracking and recognition using color, shape, and motion information” Proceedings of the IEEE Intelligent Vehicles Symposium, pp. 255–260. 2005

[5] “OpenCV Documentation”. Web. <http://opencv.org/>, [Accessed November 10th, 2015]

[6] “Road and Traffic Sign Recognition and Detection”. Web.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.2523&rep=rep1&type=pdf>
[Accessed December, 2015]

Chapter 5

- 1- Design of an Automotive Lane Keeping System Based on the Structure of Electric Power Steering. Jing-Fu Liu, Tsung-Hsien Hu, Tsung-Hua Hsu.
- 2- www.unece.org/fileadmin/DAM/trans/doc/2004/itc/itcrt/LKAS_presen_040218_final.pdf

Chapter 6

[1] Kovacs, G., Bokor, J., Palkovics, L., Gianone, L., Semsey, A., and Szell, P. (1998). "Lane-Departure Detection and Control System for Commercial Vehicles". IEEE International Conference on Intelligent Vehicles, pp. 46-50.

[2] Chang, T.H., Lin, C.H., Hsu, C.S., and Wu, Y.J. (2003). "A vision-based vehicle behavior monitoring and warning system". Intelligent Transportation Systems, October 2003, Proceedings 2003 IEEE, Vol. 1, pp. 448-453.

[3] Alyssa, D., Bunna, C., Tunyarat, T., Nalin, A., Kunanon, M., and Veeris, A. Chapter 8

1. Raspberry PI L298N Dual H Bridge DC Motor-Newsletter © 2016 Autodesk, Inc.
2. Adafruit's Raspberry Pi Lesson 9. Controlling a DC Motor Created by Simon Monk
3. Moore, Matthew (2009-12-20). "Women worse at parking than men, study shows". London: telegraph.co.uk. Retrieved 2010-01-02.
4. © 1997-2016 DriversEd.com Home / Driving Information / Driving Techniques / Perpendicular Parking
5. Driving Tips.ORG Reference