CHAPTER 1

INTRODUCTION

Automatic detecting and counting vehicles in unsupervised video on highways is avery challenging problem in computer vision with important practical applications such as to monitor activities at traffic intersections for detecting congestions, and then predictthe traffic of which assists in regulating traffic. Manually reviewing the large amount of data they generate is often impractical.

Traffic counts, speed and vehicle classification are fundamental data for a variety of transportation projects ranging from transportation planning to modern intelligent transportation systems. Traffic Monitoring and Information Systems related to vehicles cascade. In addition to vehicle counts, a much larger set of traffic parameters such as vehicle classifications, lane changes, parking areas etc., can be measured in such type of systems. In large metropolitan areas, there is a need for data about vehicle classes that use a particular highway or a street.

Traffic volume studies are conducted to determine the number, movements, and classifications of roadway vehicles at a given location. These data can help identify critical flow time periods, determine the influence of large vehicles or pedestrians on vehicular traffic flow, or document traffic volume trends. The length of the sampling period depends on the type of count being taken and the intended use of the data recorded. For example, an intersection count may be conducted during the peak flow period. If so, manual count with 15-minute intervals could be used to obtain the traffic volume data.

1.1 CASE STUDY

Most applications of manual counts require small samples of data at any given location. Manual counts are sometimes used when the effort and expense of automated equipment are not justified. Manual counts are necessary when automatic equipment is not available.

Manual counts are typically used for periods of less than a day. Normal intervals for a manual count are 5, 10, or 15 minutes. Traffic counts during a Monday morning rush hour and a Friday evening rush hour may show exceptionally high volumes and are not normally used in analysis; therefore, counts are usually conducted on a Tuesday, Wednesday, or Thursday.

Manual counts are recorded using one of three methods: tally sheets, mechanical counting boards, or electronic counting boards.

1.1.1 Tally Sheets

Recording data onto tally sheets is the simplest means of conducting manual counts. The data can be recorded with a tick mark on a pre-prepared field form. A watch or stopwatch is necessary to measure the desired count interval.

1.1.2 Mechanical Counting Boards

Mechanical count boards consist of counters mounted on a board that record each direction of travel. Common counts include pedestrian, bicycle, vehicle classification, and traffic volume counts. Typical counters are push button devices with three to five registers. Each button represents a different stratification of type of vehicle or pedestrian being counted. The limited number of buttons on the counter can restrict the number of classifications that can be counted on a given board. A watch or a stopwatch is also necessary with this method to measure the desired count interval.

1.1.3 Electronic Counting Boards

Electronic counting boards are battery-operated, hand-held devices used in collecting traffic count data. They are similar to mechanical counting boards, but with some important differences. Electronic counting boards are lighter, more compact, and easier to handle. They have an internal clock that automatically separates the data by time interval. Special functions include automatic data reduction and summary. The data can also be downloaded to a computer, which saves time for an example electronic counting board.

1.2 OBJECTIVE

In this project we are using one Raspberry Pi and one USB camera. This project is used for detecting and counting vehicles. This project runs on two different modes we need to give an option of activating camera for real-time and detecting vehicles for pre-recorded mode.

1.3 METHODOLOGY

The project need to follow certain steps in particular order to work as predicted. The steps are as follows:

- Activating camera in color mode and grayscale mode
- Record video from USB camera
- Detect vehicles from recorded video stream
- Counting vehicle after detecting
- Print the output result

CHAPTER 2

THEORITICAL BACKGROUND

2.1 HARDWARE SPECIFICATION

2.1.1 Raspberry Pi 3 Model B

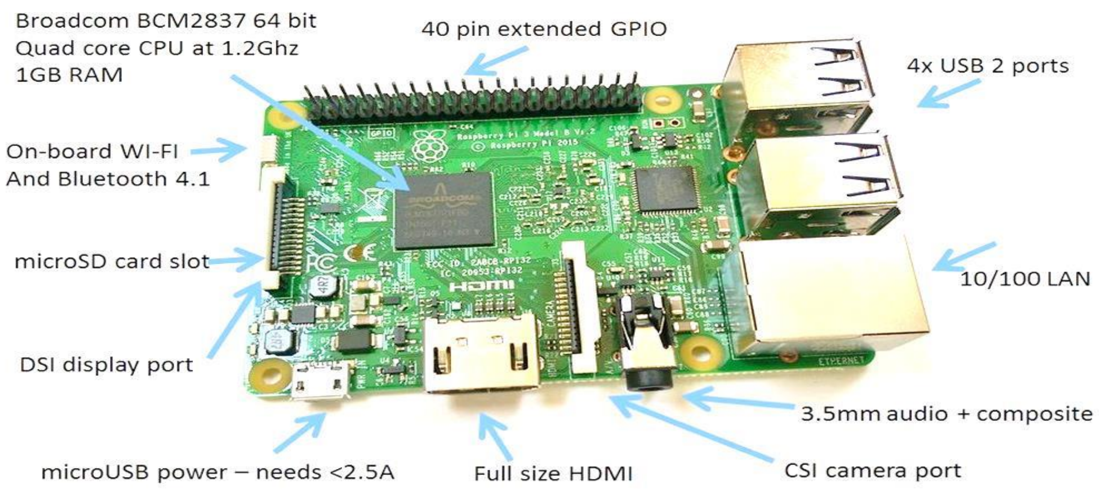


Fig 2.1.1(a): Raspberry pi 3 Model B

The Raspberry Pi 3 is the third generation Raspberry Pi. Raspberry Pi 3 is a credit card sized computer that can perform the computations like a central processing unit. It is smaller in size which can be kept in pocket. Hence it can also be referred as a pocket sized CPU. It has features similar to the CPU of a laptop or a PC that are used in daily life. But the resources present here are limited and hence computations are slower.

Raspberry Pi 3 Model B includes ARM Cortex-A53 1.2GHz quad-core Broadcom BCM2387 CPU with 1GB DDR2 RAM. In this section, we discuss the hardware specifications of Pi are discussed with the project.

Technical Specifications of Raspberry Pi 3 is as follows:

- Broadcom BCM2387 chipset, 1.2GHz Quad-Core ARM Cortex-A53 (64 bit) Processor powered Single Board Computer
- 1GB RAM
- 40 pin extended GPIO
- 4 x USB 2 ports
- 4 pole Stereo output and Composite video port
- Full size HDMI
- CSI camera port for connecting the Raspberry Pi camera
- DSI display port for connecting the Raspberry Pi touch screen display
- Micro SD port for loading your operating system and storing data
- Micro USB power source
- Supports Ethernet ,WI-FI and Bluetooth interface

One powerful feature of the Raspberry Pi is the row of GPIO (general purpose input/output) pins along the edge of the board, next to the yellow video out socket. These pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output). Seventeen of the 26 pins are GPIO pins.

- **POWER SUPPLY**

The device is powered by a 5V micro USB supply. Exactly how much current (mA)
The Raspberry Pi requires is dependent on what is to be connected to it. It has been found that purchasing a 2.5A power supply from a reputable retailer will provide us with ample power to run the Raspberry Pi.

The USB ports on a Raspberry Pi have a design loading of 100mA each - sufficient to
Drive "low-power" devices such as mice and keyboards. Devices such as Wi-Fi adapters, USB hard drives, USB pen drives all consume much more current and should be powered from an external hub with its own power supply. While it is possible to plug a 500mA device into a Pi and have it work with a sufficiently powerful supply, reliable operation is not guaranteed.

Raspberry Pi requires 3.3v. For ultrasonic sensor 5v supply is sufficient using regulator IC7805, whereas motor requires 12v DC supply by external battery.

- **Processor**

The Raspberry Pi is based on the Broadcom BCM2387 system on a chip (SoC), which includes 1.2GHz quad core ARM Cortex-A53processor, VideoCore IV GPU, and RAM.

The earlier V1.1 model of the Raspberry Pi 2 used a Broadcom BCM2836 SoC with a 900 MHz 32-bit quad-core ARM Cortex-A7 processor, with 256 KB shared L2 cache.[26] The Raspberry Pi 2 V1.2 was upgraded to a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor,[27] the same SoC which is used on the Raspberry Pi 3, but underclocked (by default) to the same 900 MHz CPU clock speed as the V1.1. The BCM2836 SoC is no longer in production (as of late 2016). It has a Level 1 cache of 16 KB and a Level 2 cache of 128 KB. The Level 2 cache is used primarily by the GPU. The SoC is stacked underneath the RAM chip, so only its edge is visible. The Raspberry Pi 3, with a quad-core ARM Cortex-A53 processor, is described as 10 times the performance of a Raspberry Pi 1.[28] This was suggested to be highly dependent upon task threading and instruction set use. Benchmarks showed the Raspberry Pi 3 to be approximately 80% faster than the Raspberry Pi 2 in parallelised tasks.

- **RAM**

Raspberry Pi 3 comes with 1GB LPDR2 RAM (shared with GPU). It is sufficient for stand alone 1080p video decoding, or for simple 3D, but not for both together. On the older beta Model B boards, 128 MB was allocated by default to the GPU, leaving 128 MB for the CPU. On the first 256 MB release Model B (and Model A), three different splits were possible. The default split was 192 MB (RAM for CPU), which should be sufficient for standalone 1080p video decoding, or for simple 3D, but probably not for both together. 224 MB was for Linux only, with only a 1080p framebuffer, and was likely to fail for any video or 3D. 128 MB was for heavy 3D, possibly also with video decoding (e.g. XBMC). Comparatively the Nokia 701 uses 128 MB for the Broadcom VideoCore IV.

For the later Model B with 512 MB RAM initially there were new standard memory split files released(arm256_start.elf, arm384_start.elf, arm496_start.elf) for 256 MB, 384 MB and 496 MB CPU RAM (and 256 MB, 128 MB and 16 MB video RAM). But a week or so later the RPF released a new version of start.elf that could read a new entry in config.txt (gpu_mem=xx) and could dynamically assign an amount of

RAM (from 16 to 256 MB in 8 MB steps) to the GPU, so the older method of memory splits became obsolete, and a single start.elf worked the same for 256 and 512 MB Raspberry Pi's.

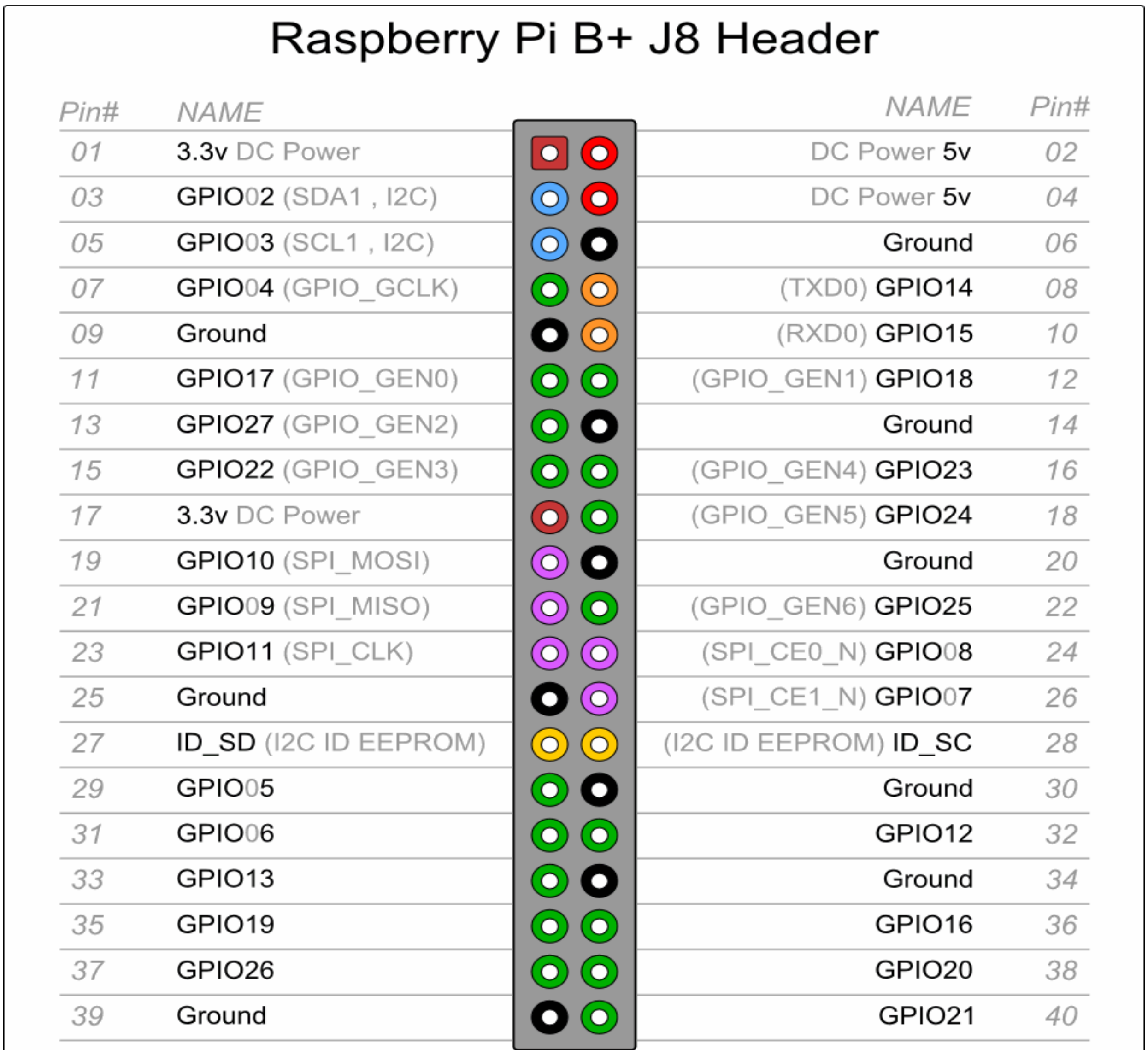
The Raspberry Pi 2 and the Raspberry Pi 3 have 1 GB of RAM.

- **Networking**

Raspberry Pi 3 can be connected to a network using Ethernet,Bluetooth and Wi-Fi Adapter. The Ethernet port is provided by a built-in USB Ethernet adapter.

The Model A, A+ and Pi Zero have no Ethernet circuitry and are commonly connected to a network using an external user-supplied USB Ethernet or Wi-Fi adapter. On the Model B and B+ the Ethernet port is provided by a built-in USB Ethernet adapter using the SMSC LAN9514 chip The Raspberry Pi 3 and Pi Zero W (wireless) are equipped with 2.4 GHz WiFi 802.11n (150 Mbit/s) and Bluetooth 4.1(24 Mbit/s) based on Broadcom BCM43438 FullMAC chip with no official support for Monitor mode but implemented through unofficial firmware patching and the Pi 3 also has a 10/100 Ethernet port. The Raspberry Pi 3B+ features dual-band IEEE 802.11b/g/n/ac WiFi, Bluetooth 4.2, and Gigabit Ethernet (limited to approximately 300 Mbit/s by the USB 2.0 bus between it and the SoC).

- **GPIO**



Raspberry Pi B+ J8 Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	(I2C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

The GPIO have 40-pin 2.54 mm expansion header: 2x20 strip . Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines.

General Purpose Input Output pins can be configured as either general-purpose input, General-purpose output or as one of up to 6 special alternate settings, the functions of which are pin-dependent.
The pads are configurable CMOS push-pull output drivers/input buffers.

- Register-based control settings are available for
- Internal pull-up / pull-down enable/disable
- Output drive-strength
- Input Schmitt-trigger filtering
- Peripherals

Generic USB keyboards and mice are compatible with the Raspberry Pi.

- **USB Port**

The Raspberry Pi 3 is equipped with four USB2.0 ports. These are connected to the
on-board 5-port USB hub. The USB ports enable the attachment of peripherals such as keyboards, mice, webcams that provide the Pi with additional functionality. There are some differences between the USB hardware on the Raspberry Pi and the USB hardware on desktop computers or laptop/tablet devices.

- **Port Power Limits**

USB devices have defined power requirements, in units of 100mA from 100mA to 500mA. The device advertises its own power requirements to the USB host when it is first connected. In theory, the actual power consumed by the device should not exceed its stated requirement.
The USB ports on a Raspberry Pi have a design loading of 100mA each - sufficient to drive "low-power" devices such as mice and keyboards. Devices such as WiFi adapters, USB hard drives, USB pen drives all consume much more current and should be powered from an external hub with its own power supply. While it is possible to plug a 500mA device into a Pi and have it work with a sufficiently powerful supply, reliable operation is not guaranteed.
In addition, hotplugging high-power devices into the Pi's USB ports may cause a brownout which can cause the Pi to reset.

2.1.2 USB CAMERA

Fig 2.1.2 : USB Camera

USB Cameras are imaging cameras that use USB 2.0 or USB 3.0 technology to transfer image data. USB Cameras are designed to easily interface with dedicated computer systems by using the same USB technology that is found on most computers. The accessibility of USB technology in computer systems as well as the 480 Mb/s transfer rate of USB 2.0 makes USB Cameras ideal for many imaging applications. An increasing selection of USB 3.0 Cameras is also available with data transfer rates of up to 5 Gb/s. As Raspberry pi supports USB 2.0 thus USB 3.0 camera is recommended.

iBall Face2Face C8.0 web camera with interpolated 8.0MP Still Image resolution, 4.0MP Video resolution and Wide angle lens provides smooth video. The raspberry pi followesavi (std) to process the video captured from the iball camera .

Features –

- Interpolated 8.0 Mega Pixel Still Image Resolution
- Interpolated 4.0 Mega Pixel Video Resolution
- High quality 5G wide angle lense
- 6 LEDs for night vision, with brightness controller
- Snapshot button for still image capture
- Built-in high sensitive USB microphone
- Built-in 10 Photo frames and 16 special effects for more fun
- 4X Digital Zoom and Auto Face Tracking Function
- Multi-utility camera base for use on Monitors, LCDs and Laptops

2.1.3 MICRO SD card

Fig 2.1.3. Micro SD card

Secure Digital (**SD**) is a non-volatile memory card format developed by the SD Card Association (SDA) for use in portable devices.Secure Digital includes four card families available in three different sizes. The four families are the original Standard-Capacity (SDSC), the High-Capacity (SDHC).

The eXtended-Capacity (SDXC), and the SDIO, which combines input/output functions with data storage. The three form factors are the original size, the mini size, and the micro size. Electrically passive adapters allow a smaller card to fit and function in a device built for a larger card. The SD card's small footprint is an ideal storage medium for smaller, thinner and more portable electronic devices.

microSD is a type of removable flash memory card used for storing information. SD is an abbreviation of Secure Digital, and microSD cards are sometimes referred to as µSD or uSD.[1] The cards are used in mobile phones and other mobile devices.

It is the smallest memory card that can be bought; at 15 mm × 11 mm × 1 mm (about the size of a fingernail), it is about a quarter of the size of a normal-sized SD card. There are adapters that make the small microSD able to fit in devices that have slots for standard SD, miniSD, Memory Stick Duo card, and even USB. But, not all of the different cards can work together. Many microSD cards are sold with a standard SD adapter, so that people can use them in devices that take standard SD but not microSD cards.

TransFlash and microSD cards are the same (they can be used in place of each other), but microSD has support for SDIO mode. This lets microSD slots support non-memory jobs like Bluetooth, GPS, and Near Field Communication by attaching a device in place of a memory card.

MicroSDHC format is a newer version of the MicroSD which is not backwards compatible. Some older devices cannot use the newer format, although third party firmware is available for some devices.

TransFlash cards are sold in 16MB and 32MB sizes. microSD cards are sold in many sizes, from 64 MB to 32 GB, while microSDHC cards are sold in sizes between 4 GB to 64 GB. Larger ones are microSDXC memory cards, sold in sizes between 8 GB and 256 GB.

2.2 SOFTWARE REQUIREMENTS

2.2.1 PYTHON

Fig 2.2.1 Python logo

Python is an interpreted high-level programming language for general-purpose programmingCreated by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales

Python features a dynamic type system and automatic memory management It supports multiple programming paradigmsincluding object-oriented imperativefunctional and proceduraland has a large and comprehensive standard library.

Python is an open source programming language that was made to be easy-to-read and powerful. A Dutch programmer named Guido van Rossum made Python in 1991. He named it after the television show Monty Python's Flying Circus. Many Python examples and tutorials include jokes from the show.

Python is an interpreted language. Interpreted languages do not need to be compiled to run. A program called an interpreter runs Python code on almost any kind of computer. This means that a programmer can change the code and quickly see the results. This also means Python is slower than a compiled language like C, because it is not running machine code directly.

Python is a good programming language for beginners. It is a high-level language, which means a programmer can focus on what to do instead of how to do it. Writing programs in Python takes less time than in some other languages.

Python drew inspiration from other programming languages like C, C++, Java, Perl, and Lisp.

Python has a very easy-to-read syntax. Some of Python's syntax comes from C, because that is the language that Python was written in. But Python uses whitespace to delimit code: spaces or tabs are used to organize code into groups. This is different from C. In C, there is a semicolon at the end of each line and curly braces ({}) are used to group code. Using whitespace to delimit code makes Python a very easy-to-read language.

Features

Simple

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English (but very strict English!). This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the syntax i.e. the language itself.

Easy to Learn

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax as already mentioned.

Free and Open Source

Python is an example of a FLOSS (Free/Libre and Open Source Software). In simple terms, you can freely distribute copies of this software, read the software's source code, make changes to it, use pieces of it in new free programs, and that you know you can do these things. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and improved by a community who just want to see a better Python.

High-level Language

When you write programs in Python, you never need to bother about low-level details such as managing the memory used by your program.

Portable

Due to its open-source nature, Python has been ported (i.e. changed to make it work on) to many many platforms. All your Python programs will work on any of these platforms without requiring any changes at all. However, you must be careful enough to avoid any system-dependent features.

You can use Python on Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC !

Interpreted

This requires a little explanation.

A program written in a compiled language like C or C++ is translated from the source language i.e. C/C++ into a language spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software just stores the binary code in the computer's memory and starts executing from the first instruction in the program.

When you use an interpreted language like Python, there is no separate compilation and execution steps. You just *run* the program from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your specific computer and then runs it. All this makes using Python so much easier. You just *run* your programs - you never have to worry about linking and loading with libraries, etc. They are also more portable this way because you can just copy your Python program into another system of any kind and it just works!

Object Oriented

Python supports procedure-oriented programming as well as object-oriented programming. In *procedure-oriented* languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In *object-oriented* languages, the program is built around objects which combine data and functionality. Python has a very powerful but simple way of doing object-oriented programming, especially, when compared to languages like C++ or Java.

Extensible

If you need a critical piece of code to run very fast, you can achieve this by writing that piece of code in C, and then combine that with your Python program.

Embeddable

You can embed Python within your C/C++ program to give scripting capabilities for your program's users.

Extensive Libraries

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI(graphical user interfaces) using Tk, and also other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the "batteries included" philosophy of Python.

Besides the standard library, there are various other high-quality libraries such as the Python Imaging Library which is an amazingly simple image manipulation library.

There are several versions of python are available in the digital platform and program recommends python 2.7.13 .

Python's design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML

Python is widely used general purpose, high level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of codes than would be possible in languages such as C. The language provides constructs intend to enable clear programmers on both a small and large scales.

Like other dynamic languages, python is often used as a scripting language, but is also used in a wide range of non scripting contexts. Using third party tools, such as Py2exe or Pyinstaller, Python code can be packaged into stand alone executable programs. Python interpreters are available for many operating systems.

CPython, the reference implementation of python, is free and open source software and has a community based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit python software foundation. Python uses dynamic typing and a combination of reference counting and a cycle detecting garbage collector for memory management. An important feature of python is dynamic name resolution (late binding), which binds method and variable names during program execution.

The design of python offers only limited support for functional programming in the lisp tradition. The language has map (), reduce () and filter () functions, comprehensions for lists, directionaries and sets, as well as generator expressions. The standard library has 2 modules(itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

Python can also be embedded in existing applications that need a programmable interface.

SYNTAX AND SEMANTICS

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

.

The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

2.2.2 OpenCV

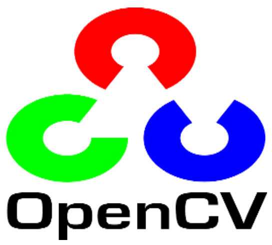


Fig 2.2.2 : OpenCV logo

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez .The library is cross-platform and free for use under the open-source BSD license.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDAand OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

HISTORY

Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.

- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008. The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems).

Official releases now occur every six months and development is now done by an independent Russian team supported by commercial corporations. In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site. On May 2016, Intel signed an agreement to acquire Itseez, the leading developer of OpenCV.

Applications of OpenCV

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

2.2.3 Raspbian OS

Fig 2.2.3 Raspbian OS Logo

Raspbian is a Debian-based computer operating system for Raspberry Pi. There are several versions of Raspbian including Raspbian Stretch and Raspbian Jessie. Since 2015 it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the family of Raspberry Pi single-board computers. Raspbian was created by Mike Thompson and Peter Green as an independent project. The initial build was completed in June 2012.The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs.

Raspbian uses PIXEL, **P**i **I**mproved **X**windows **E**nvironment, **L**ightweight as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Openbox stacking window manager with a new theme and few other changes. The distribution is shipped with a copy of computer algebra program Mathematical and a version of Minecraft called Minecraft Pi as well as a lightweight version of Chromium as of the latest version.

2.2.4 VNC server and viewer

Fig 2.2.4: VNC Viewer Logo

In computing, **Virtual Network Computing** (**VNC**) is a graphical desktop sharing system that uses the Remote Frame Buffer protocol (RFB)to remotely control another computer. It transmits the keyboard and mouse events from one computer to another, relaying the graphical screen updates back in the other direction, over a network.

VNC is platform-independent – there are clients and servers for many GUI-based operating systems and for Java. Multiple clients may connect to a VNC server at the same time. Popular uses for this technology include remote technical support and accessing files on one's work computer from one's home computer, or vice versa.

There are a number of variants of VNC which offer their own particular functionality; e.g., some optimised for Microsoft Windows, or offering file transfer (not part of VNC proper), etc. Many are compatible (without their added features) with VNC proper in the sense that a viewer of one flavour can connect with a server of another; others are based on VNC code but not compatible with standard VNC.

Operation

- The VNC server is the program on the machine that shares some screen (and may not be related to a physical display – the server can be "headless"), and allows the client to share control of it.
- The VNC client (or viewer) is the program that represents the screen data originating from the server, receives updates from it, and presumably controls it by informing the server of collected local input.
- The VNC protocol (RFB protocol) is very simple, based on transmitting one graphic primitive from server to client ("Put a rectangle of pixel data at the specified X,Y position") and event messages from client to server.

2.2.5 NumPY

Fig 2.2.5 : NumPy Logo

NumPy (pronounced ˈnʌmpaɪ (*NUM-py*) or sometimes ˈnʌmpi (*NUM-pee*)) is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier.

An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become *Numeric*, also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at MIT, joined CNRI to work on JPython in 1997 leaving Paul Dubois of LLNL to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

A new package called *Numarray* was written as a more flexible replacement for Numeric. Like Numeric, it is now deprecated. Numarray had faster operations for large arrays, but was slower than Numeric on small ones, so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and numarray v1.5.2 was released on 24 August 2006.

There was a desire to get Numeric into the Python standard library, but Guido van Rossum decided that the code was not maintainable in its state then.

In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006. This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0.

In 2011, PyPy started development on an implementation of the NumPy API for PyPy. It is not yet fully compatible with NumPy.

CHAPTER 3

DESIGN AND IMPLEMENTATION

3.1 BLOCK DIAGRAM

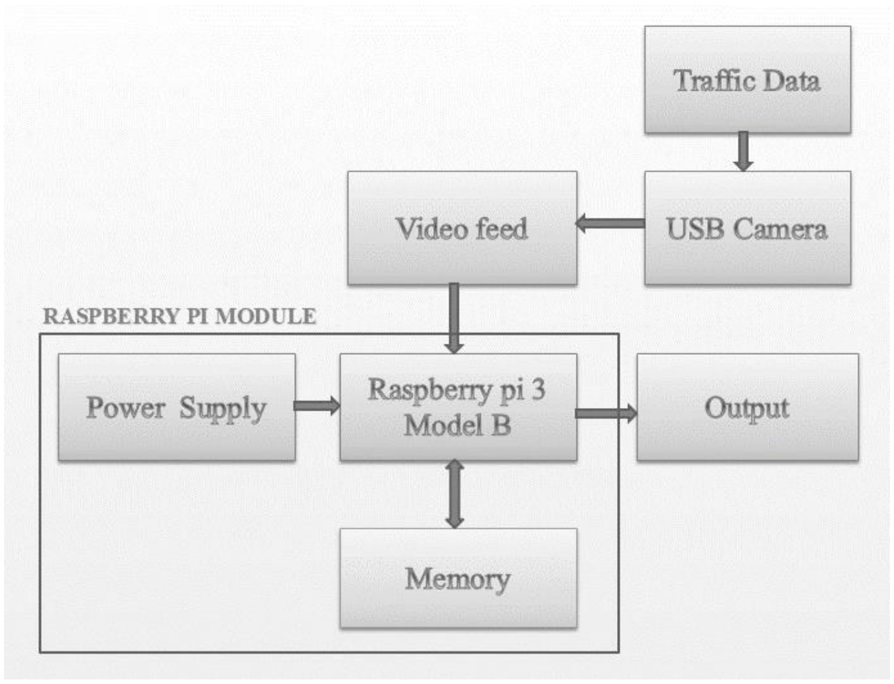


Fig. Block Diagram of the entire system

The block diagram of the project is demonstrated above each block is briefly explained part by part.

3.1.1 Traffic data

Traffic data is the actual traffic on the road of interest. The traffic may be more or less depending upon the area and number of vehicles running in that particular area at that particular instant. The data is to be taken for the project. As already mentioned, the manual traffic counting methods are tedious and also lack accuracy in counting the number of vehicle as few vehicle count can be missed by the person in-charge of counting traffic. Thus this block is the main source of input.

3.1.2 USB Camera

Among the three hardware components being used here, USB camera is another most essential component, as it records the traffic data as video. The USB camera is used because it is compatible to any device. for our project, iBall Face2Face C8.0 is used. The USB camera has 8MP resolution for still images and 4MP resolution for video capturing.

The captured video is saved in standard .mp4 format, which is most known video format in the entire digital realm. The video captures the traffic data whenever the program is called.

3.1.3 Video feed

The Video captured must be in .mp4 format and is fed to the microcomputer, Raspberry pi 3 model B. USB camera records video and not image as the project is aimed to process the video and further enhance the project to detecting and counting vehicles which we get from USB camera through Traffic data present at that particular time and particular area.

3.1.4 Raspberry pi Module

Raspberry pi module have three major components stated as follows:

- Raspberry pi
- Power supply
- Memory

Raspberry pi

In this project Raspberry Pi 3 Model B is used to process the video which was recorded from the USB camera. AS the raspberry pi has 1 GB of RAM it can process the video as quick as possible.

USB camera has USB 2.0 technology which has a data transfer rate of 480Mbps, this rate is very useful to transfer the recorded video that is, the actual traffic data from USB camera to the raspberry pi.

Raspberry pi has the program to detect and count the vehicle from the traffic data recroded using USB camera. The output can be seen on the preview window on the screen.

The counted vehicle is uploaded to Firebase server. The Firebase now has the counted value of the vehicle from the traffic data. The count can be seen on an Application which is linked with the firebase server.

The Raspberry PI has the WiFi module which is connected to the internet that helps to update the values to the firebase server, the updated value can be seen in the application’s window.

Power supply

Raspberry pi uses 5.1V and 2.5 A to run the circuit. In the project we have use micro USB charger to provide the power suppy. raspberry pi needs countinous power supply till the program is being executed, if power is cut down the raspberry pi shuts down and the counting fails.

3.1.5 Memory

Memory is required by all the system working with raspberry pi. Here 16GB Micro SD card is used. About 70% is used by the raspbian os, which is recommended for working with raspberry pi.. the rest of the memory can be used to store the files essential for this project. The video recorded from the USB camera is stored in the SD card itself. SD card is removable and expandable.

3.1.6 Output

Output of the project can either be seen on the desktop or can be seen in the application that is developed for displaying the vehicle count of the traffic data. There are many system that already exist in the present world but this approach may consume less money and time, also the accuracy of the output is also increased..

Software Installation to Raspberry Pi

3.2 Getting Started with Raspberry Pi

3.2.1 Installing Rasbian OS on Raspberry pi 3 Model B

- Download **SD Formatter** to format your Card

  https://www.sdcard.org/downloads/formatter_4/index.html/

- Install the SD Formatter tool as you install normal software on windows machine.
- Select SD card Volume Label
- Click on option and select format adjustment option to **ON**
- Download the **Win32DiskImager** or **etcher**
- **https://win32-disk-imager.en.uptodown.com/windows/**

  or https://etcher.io/

- Unzip it in the same way you did the Raspbian .zip file
- Writing Raspbian to the SD card
- Plug your SD card into your PC
- Run the file named Win32DiskImager.exe from the folder you made earlier
- If the SD card (Device) you are using isn't found automatically then click on the drop down box and select it
- In the Image File box, choose the Raspbian .img file that you downloaded
- Click Write
- After a few minutes you will have an SD card that you can use in your Raspberry Pi
- First, update your system's package list by entering the following command in LXTerminal [standard terminal emulator of LXDE] or from the command line:
- **sudo apt-get update**
- Next, upgrade all your installed packages to their latest versions with the command:
- **sudo apt-get dist-upgrade**

- Generally speaking, doing this regularly will keep your installation up to date, in that it will be equivalent to the latest released image available from raspberrypi.org/downloads

3.2.2 Installing Python Packages

While the RaspberryPi (& Raspian) run Python out-of-the-box, you'll likely want some common packaging tools for more advanced development. The following gets you some common Python tools like “pip” for easy installation/removal of packages.

```
sudo apt-get install python-dev

curl -O https://bootstrap.pypa.io/get-pip.py

sudo python get-pip.py
```

- List of packages used
  - OpenCV
  - Numpy
  - Requests
  - Firebase

Pip is considered comparatively easy method to install the required packages. Packages are installed using the command below in terminal window

```
Sudo pip install “package_name”
```

To unsinstall a package the below command can be used

```
Sudo pip uninstall “package_name”
```

The packages which are required in this project are listed above, can be installed or uninstalled using the command mentioned above.

3.3 FLOW CHART

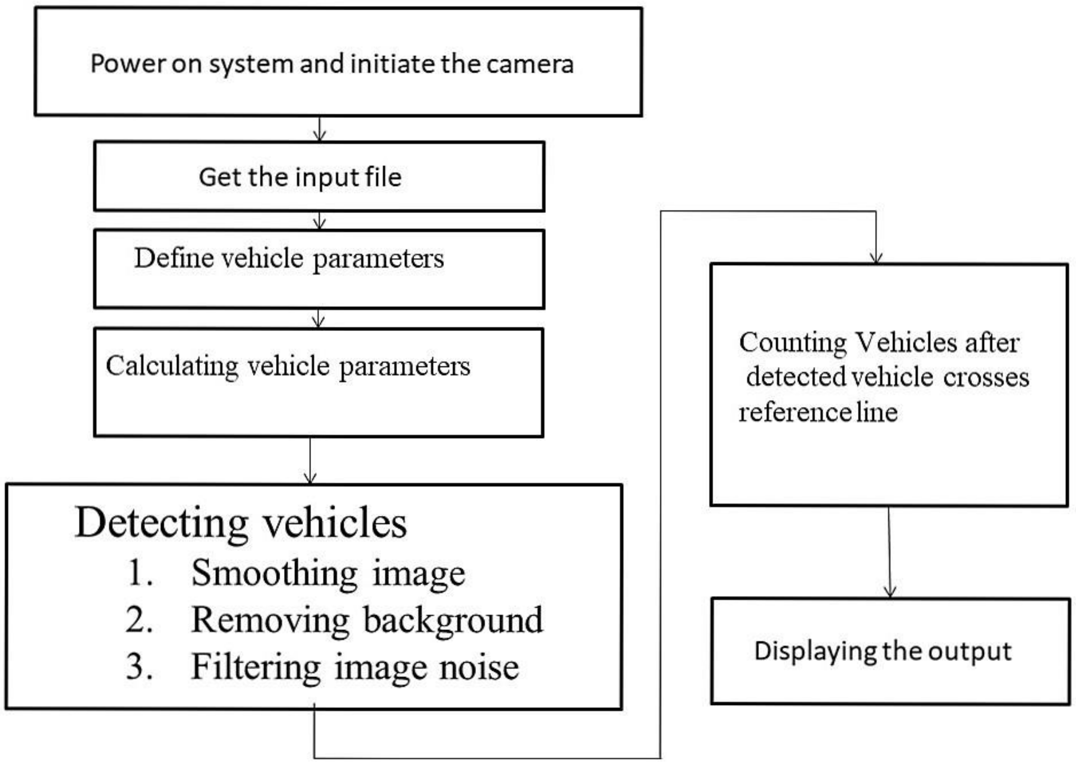

Power on system and initiate the camera
Get the input file
Define vehicle parameters
Calculating vehicle parameters
Detecting vehicles
1. Smoothing image
2. Removing background
3. Filtering image noise
Counting Vehicles after detected vehicle crosses reference line
Displaying the output

Fig. Flow chart

3.4 Code

```
#import necessary packages
import cv2
import numpy as np
import time
import logging
import math
import re
from os import walk
import os

import requests
import json
```

```
firebase_url = 'https://vehiclecount.firebaseio.com/'

#usb cam program start------------------------------

cap = cv2.VideoCapture(0)

# Check if camera opened successfully
if (cap.isOpened() == False):
  print("Unable to read camera feed")

# Default resolutions of the frame are obtained.The default resolutions are system dependent.
# We convert the resolutions from float to integer.
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

# Define the codec and create VideoWriter object.The output is stored in 'traffic_count' file.
out = cv2.VideoWriter('traffic_count.mp4',cv2.VideoWriter_fourcc('M','J','P','G'), 10,
(frame_width,frame_height))

while(True):
  ret, frame = cap.read()

  if ret == True:

    # Write the frame into the file 'traffic_count.mp4'
    out.write(frame)

    # Display the resulting frame
    cv2.imshow('frame',frame)

    # Press Q on keyboard to stop recording
    if cv2.waitKey(1) & 0xFF == ord('q'):
      break
  # Break the loop
  else:
    break

# When everything done, release the video capture and video write objects
cap.release()
out.release()

# Closes all the frames
cv2.destroyAllWindows()

#usb cam program end---------------------------------------------------------
##






# get working directory
loc = os.path.abspath('')

# Video source
inputFile = loc+'/inputs/625_201709280946.mp4'

# for testing
tracked_blobs = []
tracked_conts = []
t_retval = []

# ============================================================================

class Vehicle(object):
    def __init__(self, id, position):
        self.id = id
        self.positions = [position]
        self.frames_since_seen = 0
        self.frames_seen = 0
        self.counted = False
        self.vehicle_dir = 0

    @property
    def last_position(self):
        return self.positions[-1]
    @property
    def last_position2(self):
        return self.positions[-2]

    def add_position(self, new_position):
        self.positions.append(new_position)
        self.frames_since_seen = 0
        self.frames_seen += 1

    def draw(self, output_image):
        for point in self.positions:
            cv2.circle(output_image, point, 2, (0, 0, 255), -1)
            cv2.polylines(output_image, [np.int32(self.positions)]
                , False, (0, 0, 255), 1)

# ============================================================================

class VehicleCounter(object):
    def __init__(self, shape, divider):
        self.log = logging.getLogger("vehicle_counter")

        self.height, self.width = shape
        self.divider = divider

        self.vehicles = []
        self.next_vehicle_id = 0
        self.vehicle_count = 0
        self.vehicle_LHS = 0
        self.vehicle_RHS = 0
        self.max_unseen_frames = 10


    @staticmethod
    def get_vector(a, b):
        """Calculate vector (distance, angle in degrees) from point a to point b.

        Angle ranges from -180 to 180 degrees.
        Vector with angle 0 points straight down on the image.
        Values decrease in clockwise direction.
        """
        dx = float(b[0] - a[0])
        dy = float(b[1] - a[1])

        distance = math.sqrt(dx**2 + dy**2)

        if dy > 0:
            angle = math.degrees(math.atan(-dx/dy))
        elif dy == 0:
            if dx < 0:
                angle = 90.0
            elif dx > 0:
                angle = -90.0
            else:
                angle = 0.0
        else:
            if dx < 0:
                angle = 180 - math.degrees(math.atan(dx/dy))
            elif dx > 0:
                angle = -180 - math.degrees(math.atan(dx/dy))
        else:
            angle = 180.0

        return distance, angle, dx, dy



    @staticmethod
    def is_valid_vector(a, b):
        # vector is only valid if threshold distance is less than 12
        # and if vector deviation is less than 30 or greater than 330 degs
        distance, angle, _, _ = a
        threshold_distance = 12.0
        return (distance <= threshold_distance)



    def update_vehicle(self, vehicle, matches):
        # Find if any of the matches fits this vehicle
        for i, match in enumerate(matches):
            contour, centroid = match

            # store the vehicle data
            vector = self.get_vector(vehicle.last_position, centroid)

            # only measure angle deviation if we have enough points
            if vehicle.frames_seen > 2:
                prevVector = self.get_vector(vehicle.last_position2, vehicle.last_position)
                angleDev = abs(prevVector[1]-vector[1])
            else:
                angleDev = 0

            b = dict(
                    id = vehicle.id,
                    center_x = centroid[0],
                    center_y = centroid[1],
                    vector_x = vector[0],
                    vector_y = vector[1],
                    dx = vector[2],
                    dy = vector[3],
                    counted = vehicle.counted,
                    frame_number = frame_no,
                    angle_dev = angleDev
                    )

            tracked_blobs.append(b)

            # check validity
            if self.is_valid_vector(vector, angleDev):
                    vehicle.add_position(centroid)
                    vehicle.frames_seen += 1
                    # check vehicle direction
                    if vector[3] > 0:
                    # positive value means vehicle is moving DOWN
                    vehicle.vehicle_dir = 1
                    elif vector[3] < 0:
                    # negative value means vehicle is moving UP
                    vehicle.vehicle_dir = -1
                    self.log.debug("Added match (%d, %d) to vehicle #%d. vector=(%0.2f,%0.2f)"
                    , centroid[0], centroid[1], vehicle.id, vector[0], vector[1])
                    return i

        # No matches fit...
        vehicle.frames_since_seen += 1
        self.log.debug("No match for vehicle #%d. frames_since_seen=%d"
            , vehicle.id, vehicle.frames_since_seen)

        return None
    def update_count(self, matches, output_image = None):
        self.log.debug("Updating count using %d matches...", len(matches))

        # First update all the existing vehicles
        for vehicle in self.vehicles:
            i = self.update_vehicle(vehicle, matches)
            if i is not None:
                del matches[i]

        # Add new vehicles based on the remaining matches
        for match in matches:
            contour, centroid = match
            new_vehicle = Vehicle(self.next_vehicle_id, centroid)
            self.next_vehicle_id += 1
            self.vehicles.append(new_vehicle)
            self.log.debug("Created new vehicle #%d from match (%d, %d)."
                , new_vehicle.id, centroid[0], centroid[1])

        # Count any uncounted vehicles that are past the divider
        for vehicle in self.vehicles:
            if not vehicle.counted and (((vehicle.last_position[1] > self.divider) and (vehicle.vehicle_dir == 1))
or
                                        ((vehicle.last_position[1] < self.divider) and (vehicle.vehicle_dir == -1))) and
(vehicle.frames_seen > 6):

                vehicle.counted = True
                # update appropriate counter
                if ((vehicle.last_position[1] > self.divider) and (vehicle.vehicle_dir == 1) and
(vehicle.last_position[0] >= (int(frame_w/2)-10))):
                    self.vehicle_RHS += 1
                    self.vehicle_count += 1
                elif ((vehicle.last_position[1] < self.divider) and (vehicle.vehicle_dir == -1) and
(vehicle.last_position[0] <= (int(frame_w/2)+10))):
                    self.vehicle_LHS += 1
                    self.vehicle_count += 1

                self.log.debug("Counted vehicle #%d (total count=%d)."
                    , vehicle.id, self.vehicle_count)

        # Optionally draw the vehicles on an image
        if output_image is not None:
            for vehicle in self.vehicles:
                vehicle.draw(output_image)

            # LHS
            cv2.putText(output_image, ("LH Lane: %02d" % self.vehicle_LHS), (12, 56)
                , cv2.FONT_HERSHEY_PLAIN, 1.2, (127,255, 255), 2)
            vehi_LHS = self.vehicle_LHS
            vehi_LHS_string=str(vehi_LHS)



            # RHS
            cv2.putText(output_image, ("RH Lane: %02d" % self.vehicle_RHS), (216, 56)
                , cv2.FONT_HERSHEY_PLAIN, 1.2, (127, 255, 255), 2)
            vehi_RHS = self.vehicle_RHS
            vehi_RHS_string=str(vehi_RHS)

            ###########updating value to firebase##########################

            time_hhmmss = time.strftime('%H:%M:%S')
            date_mmddyyyy = time.strftime('%d/%m/%Y')

            #current location name
            location = 'SAMPLE ROAD'
            print (vehi_LHS_string + vehi_RHS_string + ',' + time_hhmmss + ',' + date_mmddyyyy + ',' +
location)

            #insert record
```

```
            data = {'date':date_mmddyyyy,'time':time_hhmmss,'RH LANE':vehi_RHS_string,'LH
LANE':vehi_LHS_string}
            result = requests.post(firebase_url + '/' + location + '/VEHICLECOUNT.json',
data=json.dumps(data))

            print ('Record inserted. Result Code =' + str(result.status_code) + ',' + result.text)
            time.sleep(20)
            ######################################################################

              # Remove vehicles that have not been seen long enough
        removed = [ v.id for v in self.vehicles
            if v.frames_since_seen >= self.max_unseen_frames ]
        self.vehicles[:] = [ v for v in self.vehicles
            if not v.frames_since_seen >= self.max_unseen_frames ]
        for id in removed:
            self.log.debug("Removed vehicle #%d.", id)

        self.log.debug("Count updated, tracking %d vehicles.", len(self.vehicles))

# ============================================================================

camera = re.match(r".*/(\d+)_.*", inputFile)
camera = camera.group(1)

# import video file
cap = cv2.VideoCapture(inputFile)

# get list of background files
f = []
for (_, _, filenames) in walk(loc+"/backgrounds/"):
    f.extend(filenames)
    break

# if background exists for camera: import, else avg will be built on fly
```

```
if camera+"_bg.jpg" in f:
    bg = loc+"/backgrounds/"+camera+"_bg.jpg"
    default_bg = cv2.imread(bg)
    default_bg = cv2.cvtColor(default_bg, cv2.COLOR_BGR2HSV)
    (_,avgSat,default_bg) = cv2.split(default_bg)
    avg = default_bg.copy().astype("float")
else:
    avg = None

# get frame size
frame_w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# create a mask (manual for each camera)
mask = np.zeros((frame_h,frame_w), np.uint8)
mask[:,:] = 255
mask[:100, :] = 0
mask[230:, 160:190] = 0
mask[170:230,170:190] = 0
mask[140:170,176:190] = 0
mask[100:140,176:182] = 0

# The cutoff for threshold. A lower number means smaller changes between
# the average and current scene are more readily detected.
THRESHOLD_SENSITIVITY = 40
t_retval.append(THRESHOLD_SENSITIVITY)
# Blob size limit before we consider it for tracking.
CONTOUR_WIDTH = 21
CONTOUR_HEIGHT = 16#21
# The weighting to apply to "this" frame when averaging. A higher number
# here means that the average scene will pick up changes more readily,
# thus making the difference between average and current scenes smaller.
DEFAULT_AVERAGE_WEIGHT = 0.01
INITIAL_AVERAGE_WEIGHT = DEFAULT_AVERAGE_WEIGHT / 50
```

```
# Blob smoothing function, to join 'gaps' in cars
SMOOTH = max(2,int(round((CONTOUR_WIDTH**0.5)/2,0)))
# Constants for drawing on the frame.
LINE_THICKNESS = 1

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = loc+'/outputs/'+camera+'_output.mp4'
out = cv2.VideoWriter(out, fourcc, 20, (frame_w, frame_h))

outblob = loc+'/outputs/'+camera+'_outblob.mp4'
diffop = loc+'/outputs/'+camera+'_outdiff.mp4'
outblob = cv2.VideoWriter(outblob, fourcc, 20, (frame_w, frame_h))
diffop = cv2.VideoWriter(diffop, fourcc, 20, (frame_w, frame_h))

# A list of "tracked blobs".
blobs = []
car_counter = None  # will be created later
frame_no = 0

total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
total_cars = 0

start_time = time.time()
ret, frame = cap.read()

while ret:
    ret, frame = cap.read()
    frame_no = frame_no + 1

    if ret and frame_no < total_frames:

        print("Processing frame ",frame_no)

        # get returned time
```

```
frame_time = time.time()

# convert BGR to HSV
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# only use the Value channel of the frame
(_,_,grayFrame) = cv2.split(frame)
grayFrame = cv2.bilateralFilter(grayFrame, 11, 21, 21)

if avg is None:
    # Set up the average if this is the first time through.
    avg = grayFrame.copy().astype("float")
    continue

# Build the average scene image by accumulating this frame
# with the existing average.
if frame_no < 10:
    def_wt = INITIAL_AVERAGE_WEIGHT
else:
    def_wt = DEFAULT_AVERAGE_WEIGHT

cv2.accumulateWeighted(grayFrame, avg, def_wt)

# export averaged background for use in next video feed run
#if frame_no > int(total_frames * 0.975):
if frame_no > int(200):
    grayOp = cv2.cvtColor(cv2.convertScaleAbs(avg), cv2.COLOR_GRAY2BGR)
    backOut = loc+"/backgrounds/"+camera+"_bg.jpg"
    cv2.imwrite(backOut, grayOp)

# Compute the grayscale difference between the current grayscale frame and
# the average of the scene.
differenceFrame = cv2.absdiff(grayFrame, cv2.convertScaleAbs(avg))
# blur the difference image
```

```
    differenceFrame = cv2.GaussianBlur(differenceFrame, (5, 5), 0)
#     cv2.imshow("difference", differenceFrame)
    diffout = cv2.cvtColor(differenceFrame, cv2.COLOR_GRAY2BGR)
    diffop.write(diffout)

    # get estimated otsu threshold level
    retval, _ = cv2.threshold(differenceFrame, 0, 255,
                              cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    # add to list of threshold levels
    t_retval.append(retval)

    # apply threshold based on average threshold value
    if frame_no < 10:
        ret2, thresholdImage = cv2.threshold(differenceFrame,
                                             int(np.mean(t_retval)*0.9),
                                             255, cv2.THRESH_BINARY)
    else:
        ret2, thresholdImage = cv2.threshold(differenceFrame,
                                         int(np.mean(t_retval[-10:-1])*0.9),
                                         255, cv2.THRESH_BINARY)

    # We'll need to fill in the gaps to make a complete vehicle as windows
    # and other features can split them!
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (SMOOTH, SMOOTH))
    # Fill any small holes
    thresholdImage = cv2.morphologyEx(thresholdImage, cv2.MORPH_CLOSE, kernel)

    # Remove noise
    thresholdImage = cv2.morphologyEx(thresholdImage, cv2.MORPH_OPEN, kernel)

    # Dilate to merge adjacent blobs
    thresholdImage = cv2.dilate(thresholdImage, kernel, iterations = 2)

    # apply mask
```

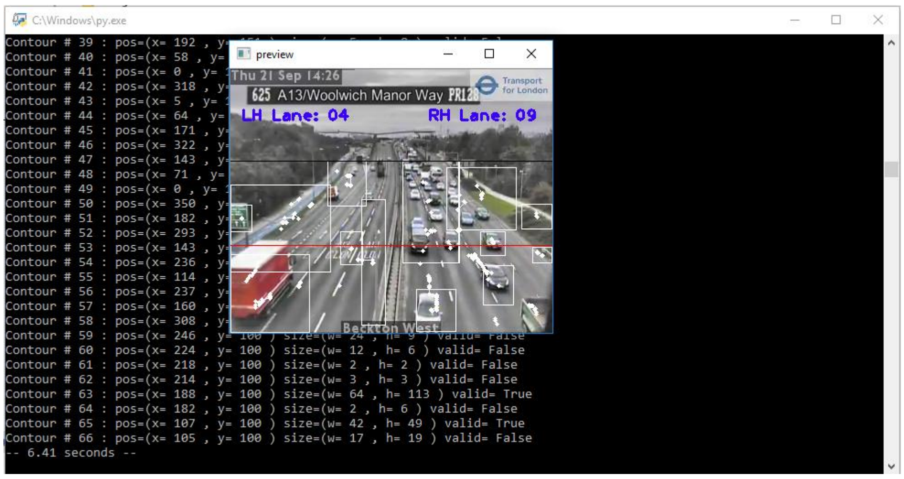
```
        thresholdImage = cv2.bitwise_and(thresholdImage, thresholdImage, mask = mask)
#         cv2.imshow("threshold", thresholdImage)
        threshout = cv2.cvtColor(thresholdImage, cv2.COLOR_GRAY2BGR)
        outblob.write(threshout)

        # Find contours aka blobs in the threshold image.
        _, contours, hierarchy = cv2.findContours(thresholdImage,
                                                  cv2.RETR_EXTERNAL,
                                                  cv2.CHAIN_APPROX_SIMPLE)

        print("Found ",len(contours)," vehicle contours.")
        # process contours if they exist!
        if contours:
            for (i, contour) in enumerate(contours):
                # Find the bounding rectangle and center for each blob
                (x, y, w, h) = cv2.boundingRect(contour)
                contour_valid = (w > CONTOUR_WIDTH) and (h > CONTOUR_HEIGHT)

                print("Contour #",i,": pos=(x=",x,", y=",y,") size=(w=",w,
                      ", h=",h,") valid=",contour_valid)

                if not contour_valid:
                    continue

                center = (int(x + w/2), int(y + h/2))
                blobs.append(((x, y, w, h), center))

        for (i, match) in enumerate(blobs):
            contour, centroid = match
            x, y, w, h = contour

            # store the contour data
            c = dict(
                        frame_no = frame_no,
                        centre_x = x,
                        centre_y = y,
                        width = w,
                        height = h
                        )
            tracked_conts.append(c)

            cv2.rectangle(frame, (x, y), (x + w - 1, y + h - 1), (0, 0, 255), LINE_THICKNESS)
            cv2.circle(frame, centroid, 2, (0, 0, 255), -1)

        if car_counter is None:
            print("Creating vehicle counter...")
            car_counter = VehicleCounter(frame.shape[:2], 2*frame.shape[0] / 3)

        # get latest count
        car_counter.update_count(blobs, frame)
        current_count = car_counter.vehicle_RHS + car_counter.vehicle_LHS

        # print elapsed time to console
        elapsed_time = time.time()-start_time
        print("-- %s seconds --" % round(elapsed_time,2))

        # output video
        frame = cv2.cvtColor(frame, cv2.COLOR_HSV2BGR)

        # draw dividing line
        # flash green when new car counted
        if current_count > total_cars:
            cv2.line(frame, (0, int(2*frame_h/3)),(frame_w, int(2*frame_h/3)),
                (0,255,0), 2*LINE_THICKNESS)
        else:
            cv2.line(frame, (0, int(2*frame_h/3)),(frame_w, int(2*frame_h/3)),
             (0,0,255), LINE_THICKNESS)
```

```
         # update with latest count
        total_cars = current_count

        # draw upper limit
        cv2.line(frame, (0, 100),(frame_w, 100), (0,0,0), LINE_THICKNESS)

        cv2.imshow("preview", frame)
        out.write(frame)

        if cv2.waitKey(27) and 0xFF == ord('q'):
            break
    else:
        break

#cv2.line()
cv2.destroyAllWindows()
cap.release()
out.release()
```

CHAPTER 4

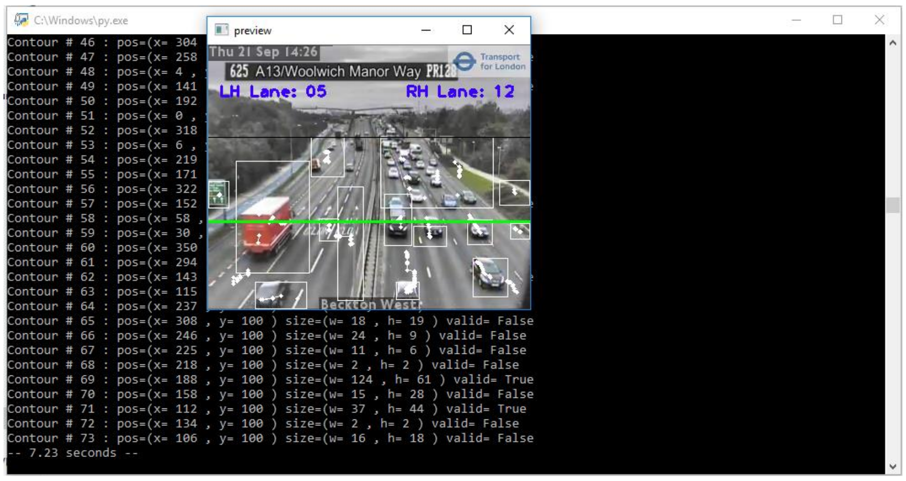
RESULT ANALYSIS

The result of this project is the number of vehicle on the particular road. The output is displayed on the video itself and also the count data is send to firebase server if the raspberry pi is connected to network. The vehicle count data can also be seen on our app designed specially for this.



```
Contour # 39 : pos=(x= 192 , y=
Contour # 40 : pos=(x= 58 , y=
Contour # 41 : pos=(x= 0 , y=
Contour # 42 : pos=(x= 318 , y=
Contour # 43 : pos=(x= 5 , y=
Contour # 44 : pos=(x= 64 , y=
Contour # 45 : pos=(x= 171 , y=
Contour # 46 : pos=(x= 322 , y=
Contour # 47 : pos=(x= 143 , y=
Contour # 48 : pos=(x= 71 , y=
Contour # 49 : pos=(x= 0 , y=
Contour # 50 : pos=(x= 350 , y=
Contour # 51 : pos=(x= 182 , y=
Contour # 52 : pos=(x= 293 , y=
Contour # 53 : pos=(x= 143 , y=
Contour # 54 : pos=(x= 236 , y=
Contour # 55 : pos=(x= 114 , y=
Contour # 56 : pos=(x= 237 , y=
Contour # 57 : pos=(x= 160 , y=
Contour # 58 : pos=(x= 308 , y=
Contour # 59 : pos=(x= 246 , y=
Contour # 60 : pos=(x= 224 , y= 100 ) size=(w= 12 , h= 6 ) valid= False
Contour # 61 : pos=(x= 218 , y= 100 ) size=(w= 2 , h= 2 ) valid= False
Contour # 62 : pos=(x= 214 , y= 100 ) size=(w= 3 , h= 3 ) valid= False
Contour # 63 : pos=(x= 188 , y= 100 ) size=(w= 64 , h= 113 ) valid= True
Contour # 64 : pos=(x= 182 , y= 100 ) size=(w= 2 , h= 6 ) valid= False
Contour # 65 : pos=(x= 107 , y= 100 ) size=(w= 42 , h= 49 ) valid= True
Contour # 66 : pos=(x= 105 , y= 100 ) size=(w= 17 , h= 19 ) valid= False
-- 6.41 seconds --
```

Fig: Result Window1



```
Contour # 46 : pos=(x= 304
Contour # 47 : pos=(x= 258
Contour # 48 : pos=(x= 4 ,
Contour # 49 : pos=(x= 141
Contour # 50 : pos=(x= 192
Contour # 51 : pos=(x= 0 ,
Contour # 52 : pos=(x= 318
Contour # 53 : pos=(x= 6 ,
Contour # 54 : pos=(x= 219
Contour # 55 : pos=(x= 171
Contour # 56 : pos=(x= 322
Contour # 57 : pos=(x= 152
Contour # 58 : pos=(x= 58 ,
Contour # 59 : pos=(x= 30 ,
Contour # 60 : pos=(x= 350
Contour # 61 : pos=(x= 294
Contour # 62 : pos=(x= 143
Contour # 63 : pos=(x= 115
Contour # 64 : pos=(x= 237
Contour # 65 : pos=(x= 308 , y= 100 ) size=(w= 18 , h= 19 ) valid= False
Contour # 66 : pos=(x= 246 , y= 100 ) size=(w= 24 , h= 9 ) valid= False
Contour # 67 : pos=(x= 225 , y= 100 ) size=(w= 11 , h= 6 ) valid= False
Contour # 68 : pos=(x= 218 , y= 100 ) size=(w= 2 , h= 2 ) valid= False
Contour # 69 : pos=(x= 188 , y= 100 ) size=(w= 124 , h= 61 ) valid= True
Contour # 70 : pos=(x= 158 , y= 100 ) size=(w= 15 , h= 28 ) valid= False
Contour # 71 : pos=(x= 112 , y= 100 ) size=(w= 37 , h= 44 ) valid= True
Contour # 72 : pos=(x= 134 , y= 100 ) size=(w= 2 , h= 2 ) valid= False
Contour # 73 : pos=(x= 106 , y= 100 ) size=(w= 16 , h= 18 ) valid= False
-- 7.23 seconds --
```

Fig : Result Window2

CHAPTER 5

4.1 ADVANTAGES

1) Vehicle classifications, lane changes, parking areas etc., can be measured.
2) Efficient system and economical.
3) Labour cost reduction and digital data storage.
4) Statistical analysis possible.

4.2 APPLICATION

1) Vehicle classifications, lane changes, parking areas etc., can be measured.
2) Efficient system and economical.
3) Labour cost reduction and digital data storage.
4) Statistical analysis possible.

4.3 LIMITATIONS

1) Recording during night might affect the detecting of vehicle
2) Constant power supply is needed
3) Weather conditions may damage the system

4.4 CONCLUSION

The main objective of our project traffic counting using open cv and python is achieved. This work investigates on the vehicle using camera input and count the traffic intensity in a particuler area and achieves significantly improved performance over previously proposed approaches. We present an approach for identifying and recognizing the red all kind of vehicle recognise it and to track it.

The objective of the proposed system is to detect and track the object that is being recognised and reduce the human effort in its application use. The video of the working model is being captured. The camera continuously captures the images and can be seen in one window, while the distance measured and the centre calculated for the corresponding image is seen in the parallel window simultaneously. The experimental results indicate that the system is highly accurate.

REFERENCES

a) A. J. Kun and Z. Vamossy, "Traffic monitoring with computer vision," Proc. 7th Int. Symp. Applied Machine Intelligence and Informatics (SAMI 2009).

b) Neeraj K. Kanhere, Stanley T. Birchfield, Wayne A. Sarasua, Tom C. Whitney, Real-time detection and tracking of vehicle base fronts for measuring traffic counts and speeds on highways", Transportation Research Record, No. 1993. (2007).

c) M. Lei, D. Lefloch, P. Gouton, and K. Madani, "A Video-Based Real-Time Vehicle Counting System Using Adaptive Background Method", 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems, pp. 523-528.

d) http://hackster.io

e) http://github.com

f) http://sourceforge.net