CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The importance of surveillance robots is increasing as security is top priority in today's unpredictable society. With the advance in technology, we are able to build more robust robots to handle hazardous situations and save several lives. Introducing image processing into surveillance robot can result in effective method to handle and inform hazardous situations to the control station.

The main purpose of the robot we are making is to provide visual information to a selected area and provide surveillance to that particular area. Hence the main feature of our robot is an onboard video camera. Also the robot mus t be compact and self contained in the sense it must have an onboard battery pack and wir eless interface to the human controller.

This project attempts to address the need for a self-contained security system. Currently, security systems require many costly components and a complicated installation process. Two basic types of systems are currently available. The first is a wired system. One drawback is that installation of a wired system can take a lot of time and money. Another drawback is that it is a permanent part of the home. If the owner moves, the security system must stay. The second type of system is a wireless one. The components for this are also costly. Wireless systems are more mobile, but they require batteries which must be changed every so often.

1.2 OBJECTIVES

- Gain information about the environment
- Work for an extended period without human intervention
- Move either all or part of itself throughout its operating environment without human assistance
- Notify the control station about any changes in the environment

1.3 BLOCK DIAGRAM

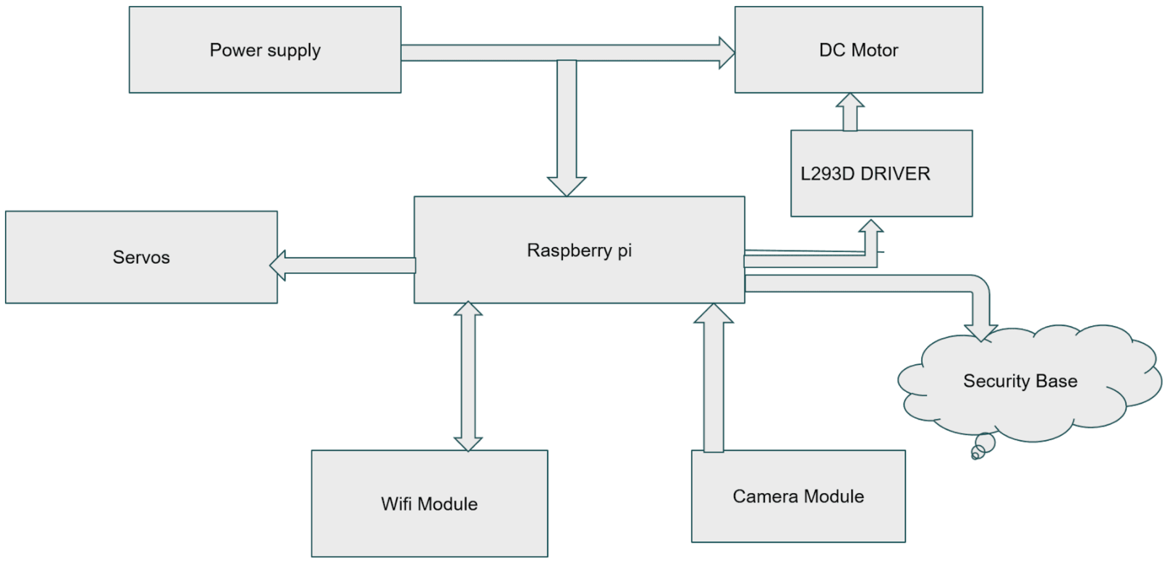


Fig.1.1 block diagram of typical image processing system

Fig 1.1 shows the block diagram of typical image processing system. Raspberry pi is the heart of system which will compute image processing results and uploads the results into the cloud. The robot module will cover the entire given area with randomness and takes images at random points, compares the present image with the previous image. The results are sent to the control station over the wifi module. Camera is attached to the stepper motor which will allow the camera to rotate all over 360^0 both in clockwise direction and anti clockwise direction. The module runs on rechargeable battery supply which will provide 5v and upto 2A of dc power supply.

The robot is expected to serve two main features. One among is to provide surveillance to the given area and the other is number plate recognition. Number plate recognition feature is useful for parking surveillance which will identify false occupation of reserved parking area. The module identifies the vehicle registration number on the number plate and compares it against the database available to verify the parking spot occupation.

CHAPTER 2

HARDWARE DESCRIPTION

2.1 RASPBERRY PI 3 MODEL B

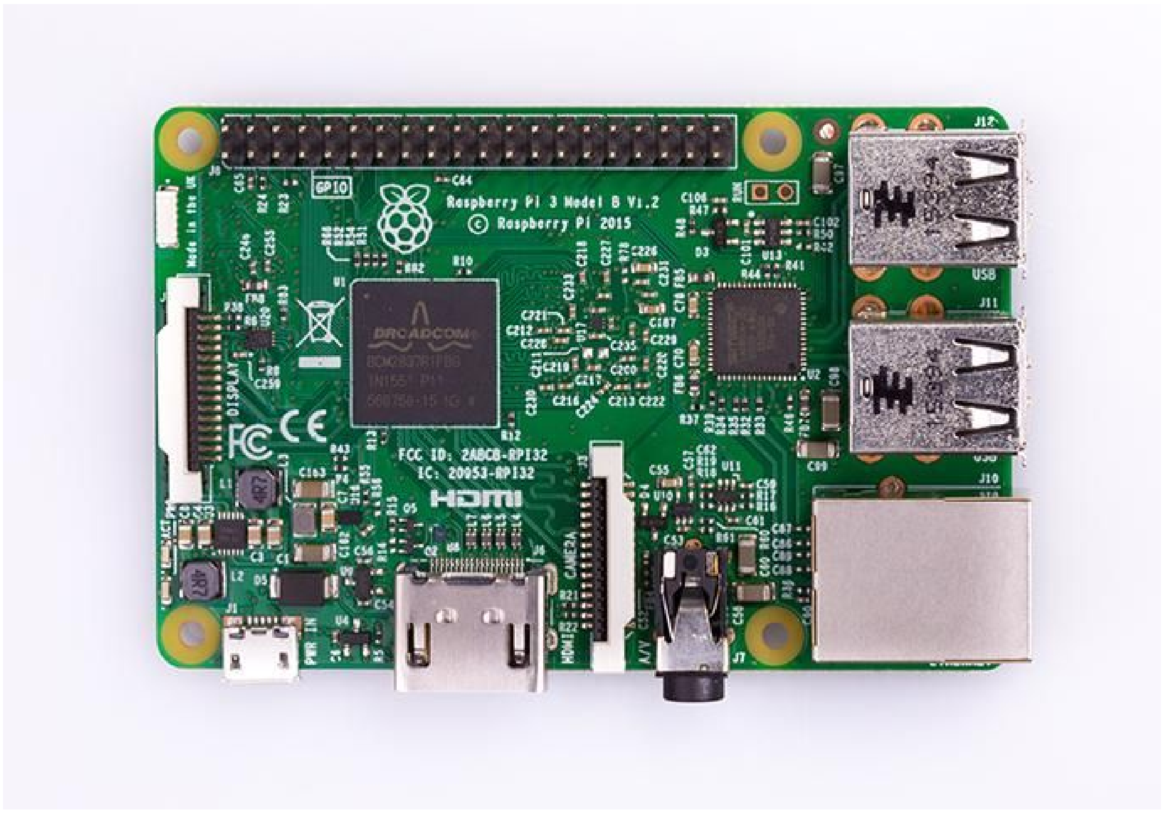


Fig. 2.1 Raspberry pi

2.1.1 OVERVIEW

The Raspberry Pi (fig 2.1) is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside of its target market for uses such as robotics. Peripherals are not included with the Raspberry Pi. Some accessories however have been included in several official and unofficial bundles. The Raspberry Pi 3

Model B is the third generation Raspberry Pi. This powerful credit-card sized single board computer can be used for many applications and supersedes the original Raspberry Pi Model B+ and Raspberry Pi 2 Model B. Whilst maintaining the popular board format the Raspberry Pi 3 Model B brings you a more powerful processer, 10x faster than the first generation Raspberry Pi. Additionally it adds wireless LAN & Bluetooth connectivity making it the ideal solution for powerful connected designs.

2.1.2 FEATURES

Performance to enable excellent end-user experience

- 750MHz (TSMC 65GP) with conservative design 1GHz+ with design optimizations
- Low latency mode for interrupt responsiveness
- TCM for ARM9™ processor migration, real-time demands
- Physically addressed caches for multi-tasking performance
- Broad OS support, multiple Linux distributions, amazing ARM ecosystem
- Full Internet experience

Product maturity enables rapid time to market and low risk

- Well proven technology in wide range of applications
- Available as soft core or hard macro from ARM (TSMC (90G) or from 3rd parties (Socle/GLOBALFOUNDRIES - 65nm, TSMC - 65nm)
- AMBA AXI supported by wide range of fabric
- CoreSight debug offering unrivalled system visibility
- Comprehensive range of development tools from ARM and from third parties
- Range of Reference Methodologies supplied

Low Power Leadership

- Shutdown modes, Clock Gating, and DVFS capability
- 93% of flops are clock gated
- Separate Main TLB and Micro-TLBs --> main TLB is not clocked unless micro-TLB misses
- Avoids unnecessary Tag-RAM and Data-RAM activity for sequential accesses
- Predictive use of Cache or TCM

DVFS - supported by the ARM1176

- Physical IP Power Management Kit Clamps allow core to shut down while TCM still enabled in Dormant mode
- Asynchronous interfaces allow core to run at non-integer multiple of bus frequency
- L-Shift clamps allow core voltage to differ from rest of SoC

Powerful Vector Floating Point Unit

- Fully IEEE 754-compliant mode
- Rare cases bounced to software support code for full IEEE 754 support
- Suited to safety-critical application like Automotive and Avionics

RunFast mode for fast, near-compliant hardware execution of all instructions

- No software support needed, performance increased as no software involved
- Suited to consumer applications like 3D graphics
- Compiler automatically targets VFP

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
- VFP may be powered-down when not in use
- Clock gated and/or power completely removed

VFP has it's own additional register set

- 32 single precision
- 16 double-precision
- Sufficient to double-buffer algorithms
- Linpack, Filters, Graphics transforms, etc.

Secure Computing Environment with TrustZone® technology

- Secures against software-only "hack attacks" and most low budget hardware "shack attacks"
- Delivers two "virtual" cores with deep separating of context & data.
- S/W and JTAG attacks can not enter secure domain
- TrustZone: two virtualized CPUs in one
- CPU MHz/resources are dynamically shared between according to demands
- The two isolated domains are implemented in the same machine with no duplication of HW
- Simpler and more flexible platform designs, lower costs and high power/performance efficiency

2.1.3 ARCHITECTURAL OVERVIEW

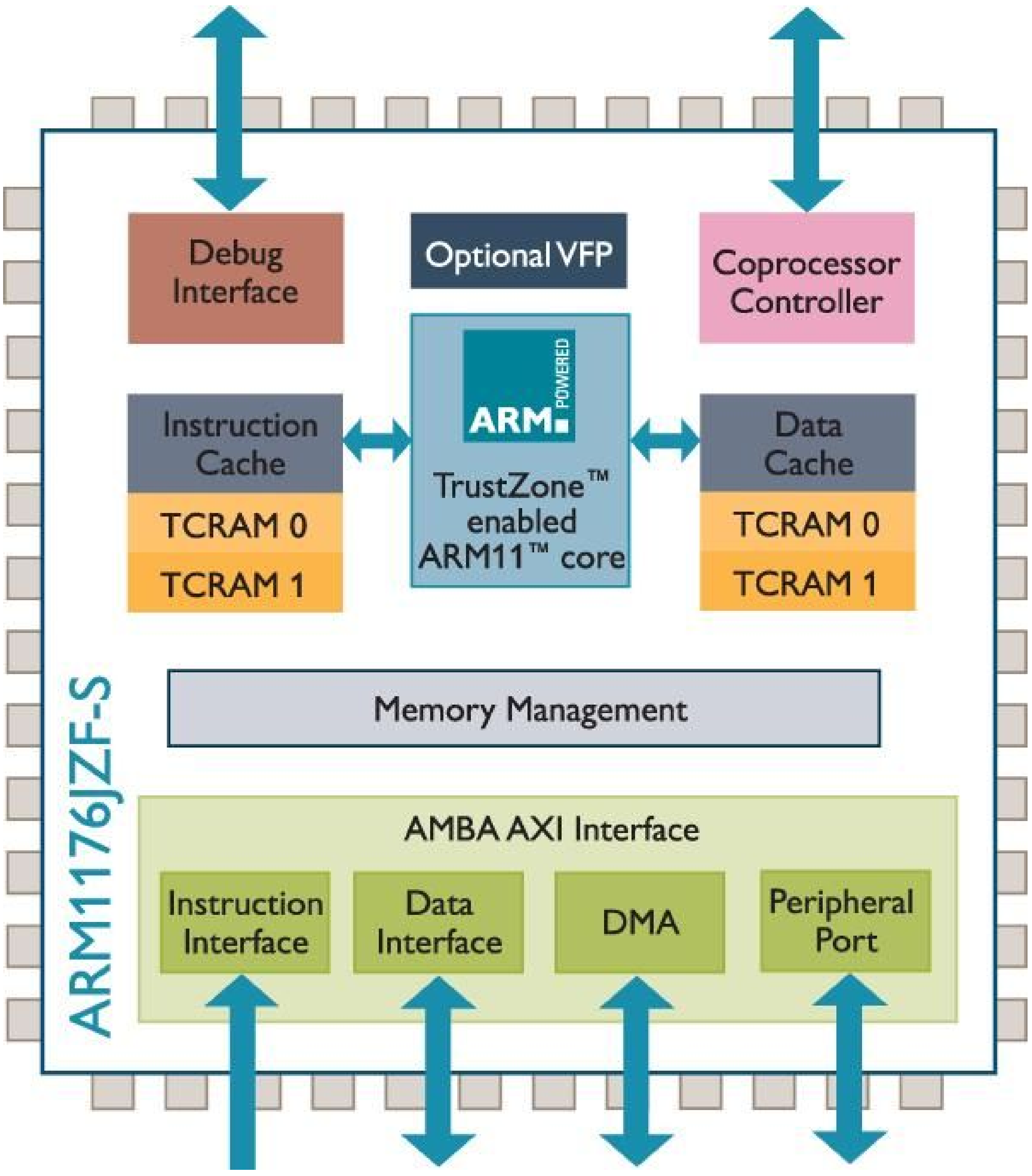


Fig. 2.2: ARM architecture

The ARM1176™ applications processors deployed broadly in devices ranging from smart phones to digital TV's to eReaders, delivering media and browser performance, a secure computing environment, and performance up to 1GHz in low cost designs. The ARM1176JZ-S processor features ARM Trust Zone® technology for secure applications and ARM Jazelle® technology for efficient embedded Java execution. Optional tightly coupled memories simplify ARM9™ processor migration and real-time design, while AMBA® 3 AXI™ interfaces improve memory bus performance. DVFS support enables power optimization below the best-in-class nominal static and dynamic power of the ARM11™ processor architecture.

Instruction compression

A typical 32-bit architecture can manipulate 32-bit integers with single instructions, and address a large address space much more efficiently than a 16-bit architecture. When processing 32-bit data, a 16-bit architecture takes at least two instructions to perform the same task as a single 32-bit instruction.

When a 16-bit architecture has only 16-bit instructions, and a 32-bit architecture has only 32-bit instructions, overall the 16-bit architecture has higher code density, and greater than half the performance of the 32-bit architecture.

Thumb implements a 16-bit instruction set on a 32-bit architecture, giving higher performance than on a 16-bit architecture, with higher code density than a 32-bit architecture.

The ARM1176JZ-S processor can easily switch between running in ARM state and running in Thumb state. This enables you to optimize both code density and performance to best suit your application requirements

The Thumb instruction set

The Thumb instruction set is a subset of the most commonly used 32-bit ARM instructions. Thumb instructions are 16 bits long, and have a corresponding 32-bit ARM instruction that has the same effect on the processor model. Thumb instructions operate with the standard ARM register configuration, enabling excellent interoperability between ARM and Thumb states.

Thumb has all the advantages of a 32-bit core:

- 32-bit address space
- 32-bit registers
- 32-bit shifter and *Arithmetic Logic Unit* (ALU)
- 32-bit memory transfer.

Thumb therefore offers a long branch range, powerful arithmetic operations, and a large address space.

The availability of both 16-bit Thumb and 32-bit ARM instruction sets, gives you the flexibility to emphasize performance or code size on a subroutine level, according to the requirements of their applications. For example, you can code critical loops for applications such as fast interrupts and DSP algorithms using the full ARM instruction set, and linked with Thumb code.

Java bytecodes

ARM architecture v6 with Jazelle technology executes variable length Java bytecodes. Java bytecodes fall into two classes:

Hardware execution

Bytecodes that perform stack-based operations.

Software execution

Bytecodes that are too complex to execute directly in hardware are executed in software. An ARM register is used to access a table of exception handlers to handle these particular bytecodes.

2.1.4 TECHNICAL SPECIFICATIONS

- CPU: Quad-core 64-bit ARM Cortex A53 clocked at 1.2 GHz
- GPU: 400MHz Video Core IV multimedia
- Memory: 1GB LPDDR2-900 SDRAM (i.e. 900MHz)
- USB ports: 4
- Video outputs: HDMI, composite video (PAL and NTSC) via 3.5 mm jack
- Network: 10/100Mbps Ethernet and 802.11n Wireless LAN
- Peripherals: 17 GPIO plus specific functions, and HAT ID bus
- Bluetooth: 4.1
- Power source: 5 V via MicroUSB or GPIO header
- Size: 85.60mm × 56.5mm
- Weight: 45g

2.2 SOFTWARE

lubuntu

The objective of the Lubuntu project is to create a variant of Ubuntu that is lighter, less resource hungry and more energy-efficient by using lightweight applications and LXDE, The Lightweight X11 Desktop Environment, as its default GUI.

Lubuntu is targeted at PC and laptop users running on low-spec hardware that, in most cases, just don't have enough resources for all the bells and whistles of the "full-featured" mainstream distributions. Members of the team take care of LXDE and other packages that are part of Lubuntu. Lubuntu received official recognition as a formal member of the

Ubuntu family, commencing with Lubuntu 11.10. Even though Lubuntu is a lightweight distribution, it does not mean you will not be able to run all applications that other distributions are currently offering. It does mean that it is prepared and developed for low-specification computers. Nonetheless, you can run any application available in the official repositories, as long as your hardware can bear with it

2.2.1 System Requirements

- Lubuntu can be installed on a Pentium II or Celeron system with 128 MB of RAM, but such a system would not perform well enough for daily use.
- With 256MB - 384MB of RAM, the performance will be better and the system will be more usable.
- With 512MB of RAM, you don't need to worry much.
- The default "Desktop" installer requires 384-800 MB of RAM

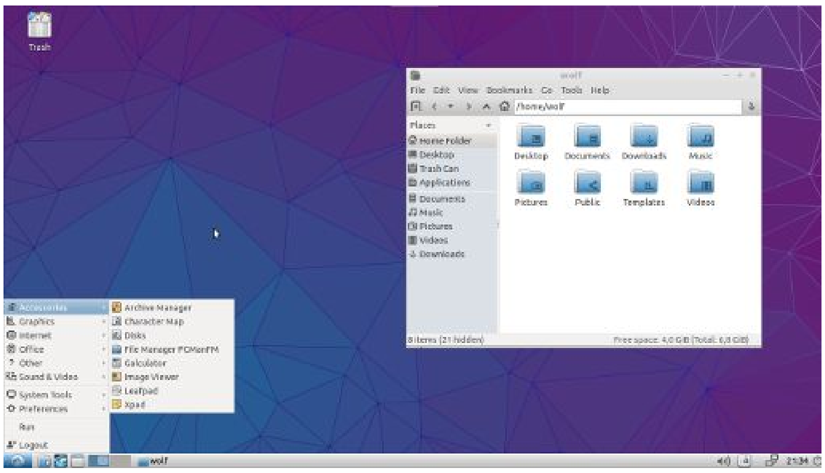


Fig. 2.3 linux os

2.2.2 Lubuntu vs Ubuntu

Both Lubuntu and Ubuntu share two major important things: same **Core System** and same **Repositories**. Lubuntu and Ubuntu belong to the same family and talking about each as totally different two systems is not correct since they have some things in common. Thus, we use the same Forum Area and share many Wiki Pages.

The differences between Lubuntu and Ubuntu are:

- Different DE - Lubuntu uses LXDE while Ubuntu uses Unity as the default DE.
- Different Default Applications

Other than that, they are the same. The DE is what makes Lubuntu a lightweight OS, and of course the selected applications too because we make sure to use the lightest applications which are not resource hungry. However, you are still free to use any application available in Ubuntu's repositories, as long as your computer can run it.

Lubuntu is recognized as a member the Ubuntu family by the developers of Ubuntu.

2.2.3 Python

Python is a widely used high-level programming language for general-purpose programming. An interpreted language, Python has a design philosophy which emphasizes code readability and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.

Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems.Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming. Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a mix 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. CPython, the reference implementation of Python, is open source softwareand has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Advantages of Python over MATLAB:

•Free ; Open source language

•Beautiful programming language: The python programming language is easier to read and to program than the Matlab programming language.

•Powerful : It's so well designed, it's easier than other languages to transform your ideas into code. But also, Python comes with extensive standard libraries, and has a powerful datatypes such as lists, sets and dictionaries. These really help to organize your data.

•Namespaces. Matlab now supports namespaces for the functions that you write, but the core of Matlab is without namespaces; every function is defined in the global namespace. Python works with Modules, which you need to import if you want to use them. Python starts up in under a second. Using namespaces gives structure to a program and keeps it clean and clear. In Python everything is an object, so each object has a namespace itself.

•Introspection:The object oriented nature of Python. Because a program has a clear structure, introspection is easy.

•String manipulation: Incredibly easy in Python

•Portability: Because Python is for free, your code can run everywhere.

•Indexing: Python indexing goes as it does in C.

•Class and function definitions: Functions and classes can be defined anywhere. In one file you can design as many functions and classes as you like

•Great GUI toolkits: With Python you can create a front-end for your application that looks good and works well. You can chose any of the major GUI toolkits like WX or QT

2.2.4 Open CV

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'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++, C, and 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 CUDA and 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 template interface that works seamlessly with STL containers.

- **core** - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- **imgproc** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **video** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **features2d** - salient feature detectors, descriptors, and descriptor matchers.
- **objdetect** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- **highgui** - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.
- **gpu** - GPU-accelerated algorithms from different OpenCV modules

Installing Open CV on Raspberry Pi

Step 0:

Open a terminal and update and upgrade the installed packages on the Raspberry pi.

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

```
$ sudo rpi-update
```

Step 1:

Install the required developer tools and packages:

```
$ sudo apt-get install build-essential cmake pkg-config
```

```
$ sudo apt-get install build-essential cmake pkg-config
```

Step 2:

Install the necessary image I/O packages. These packages allow you to load various image file formats such as JPEG, PNG, TIFF, etc.

```
$ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev
```

Step 3:

Install the GTK development library

```
$ sudo apt-get install libgtk2.0-dev
```

Step 4:

Install the necessary video I/O packages. These packages are used to load video files using OpenCV:

```
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
```

Step 5:

Install libraries that are used to optimize various operations within OpenCV

```
$ sudo apt-get install libatlas-base-dev gfortran
```

Step 6:

Install pip :

```
$ wget https://bootstrap.pypa.io/get-pip.py

$ sudo python get-pip.py

$ wget https://bootstrap.pypa.io/get-pip.py

$ sudo python get-pip.py
```

Step 7:

Install virtuallenv and virtual wrapper

```
$ sudo pip install virtualenv virtualenvwrapper

$ sudo rm -rf ~/.cache/pip
```

Later update the profile

```
# virtualenv and virtualenvwrapper

export WORKON_HOME=$HOME/.virtualenvs

source /usr/local/bin/virtualenvwrapper.sh
```

reload profile and create a vision virtual environment

```
$ source ~/.profile
```

$ mkvirtualenv cv

Step 8:

Install the Python 2.7 development tools:

$ sudo apt-get install python2.7-dev

$ pip install numpy

Step 9:

Download OpenCV and unpack it:

$ wget -Oopencv-2.4.10.zip http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.10/opencv-2.4.10.zip/download

$ unzip opencv-2.4.10.zip

$ cd opencv-2.4.10

$ mkdir build

$ cd build

$cmake-DCMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_NEW_PYTHON_SUPPORT=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON ..

Step 10:

Compile and install OpenCV:

$ make

$ sudo make install

$ sudo ldconfig

```
$ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/

$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so

$ ln -s /usr/local/lib/python2.7/site-packages/cv.py cv.py
```

2.3 CAMERA MODULE

A webcam is a video camera that feeds or streams its image in real time to or through a computer to a computer network. When "captured" by the computer, the video stream may be saved, viewed or sent on to other networks via systems such as the internet, and emailed as an attachment. When sent to a remote location, the video stream may be saved, viewed or on sent there. Unlike an IP camera (which connects using Ethernet or Wi-Fi), a webcam is generally connected by a USB cable, or similar cable, or built into computer hardware, such as laptops.

The term "webcam" (a clipped compound) may also be used in its original sense of a video camera connected to the Web continuously for an indefinite time, rather than for a particular session, generally supplying a view for anyone who visits its web page over the Internet. Some of them, for example, those used as online traffic cameras, are expensive, rugged professional video cameras.

Webcams are known for their low manufacturing cost and their high flexibility, making them the lowest-cost form of videotelephony. Despite the low cost, the resolution offered at present (2015) is rather impressive, with low-end webcams offering resolutions of 320×240, medium webcams offering 640×480 resolution, and high-end webcams offering 1280×720 (aka 720p) or even 1920×1080 (aka 1080p) resolution.

They have also become a source of security and privacy issues, as some built-in webcams can be remotely activated by spyware.The most popular use of webcams is the establishment of video links, permitting computers to act as videophones or videoconference stations. Other popular uses include security surveillance, computer vision, video broadcasting, and for recording social videos.

2.4 ULTRASONIC SENSOR (HC SR-04)

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats or dolphins do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. From 2cm to 400 cm or 1” to 13 feet.

Its operation is not affected by sunlight or black material like Sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with ultrasonic transmitter and receiver module.

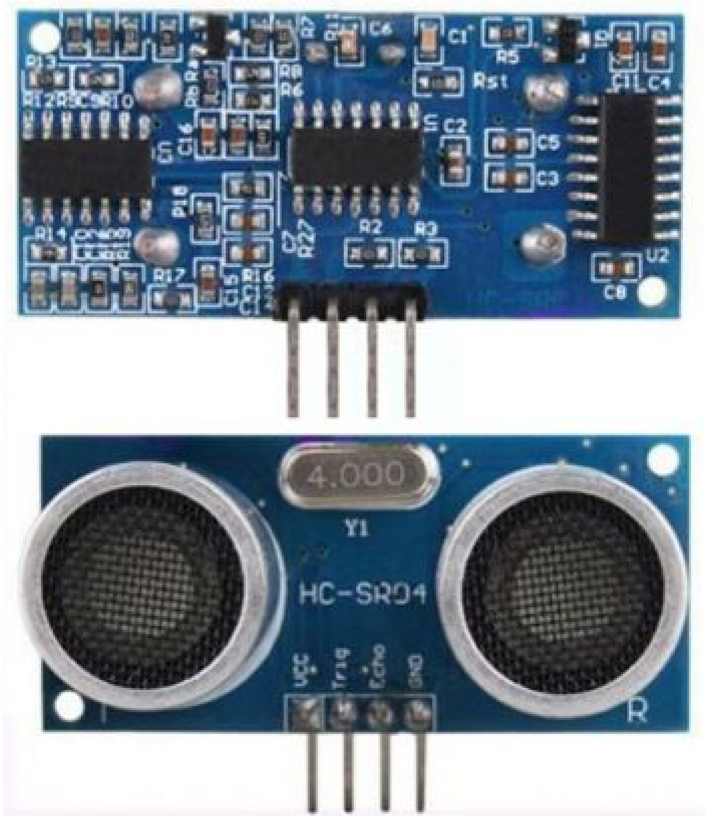


Fig. 2.4 HC SR-04 ultrasonic sensor

2.4.1 FEATURES

- Power Supply : +5V DC

- Quiescent Current : <2mA
- Working current : 15mA
- Effectual Angle : <15 degree
- Ranging distance : 2cm-400cm/1”-13ft
- Resolution : 0.3 cm
- Measuring Angle : 30 degree
- Trigger Input width : 10micro second
- Dimension : 45mm x 20mm x 15mm

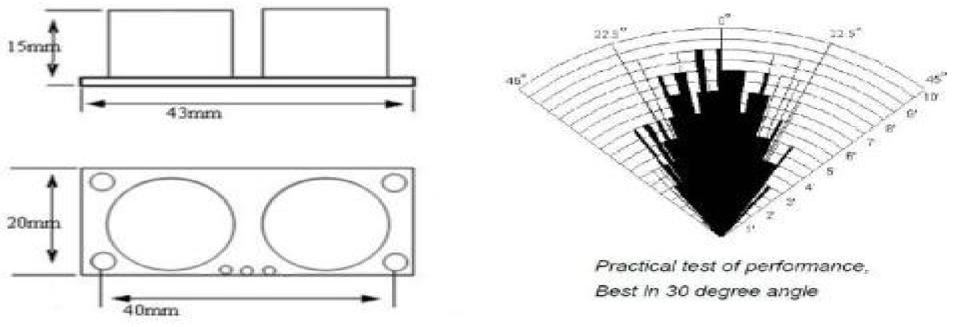


Fig. 2.5 Product layout of HC SR-04

2.4.2 PRINCIPLE OF OPERATION

The timing diagram of HC-SR04 is shown in the Fig 3.6. To start measurement, Trig of SR04 must receive a pulse of high (5V) for at least 10us, this will initiate the sensor will transmit out 8 cycle of ultrasonic burst at 40kHz and wait for the reflected ultrasonic burst. When the sensor detected ultrasonic from receiver, it will set the Echo pin to high (5V) and delay for a period (width) which proportion to distance. To obtain the distance, measure the width (Ton) of Echo pin.

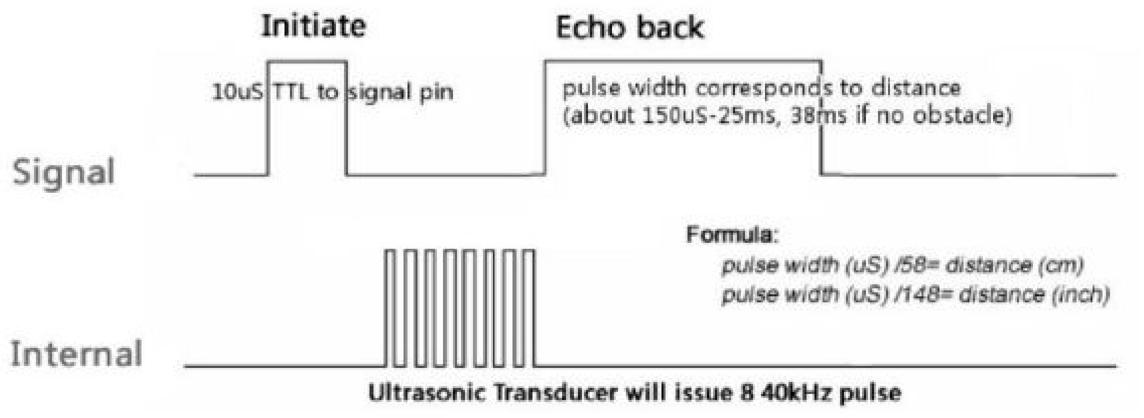

Initiate
Echo back
10uS TTL to signal pin
pulse width corresponds to distance
(about 150uS-25ms, 38ms if no obstacle)
Signal
Formula:
pulse width (uS) /58= distance (cm)
pulse width (uS) /148= distance (inch)
Internal
Ultrasonic Transducer will issue 8 40kHz pulse

Fig. 2.6 Timing diagram of HC SR-04

Time = Width of Echo pulse, in uS (micro second)

- Distance in centimeters = Time / 58
- Distance in inches = Time / 14
- Or the speed of sound, which is 340m/s

2.4.3 PRODUCT SPECIFICATION

Parameter	Min	Typ.	Max	Unit
Operating Voltage	4.50	5.0	5.5	V
Quiescent Current	1.5	2	2.5	mA
Working Current	10	15	20	mA
Ultrasonic Frequency	-	40	-	kHz

Table 2.1 Ultrasonic sensor specifications

2.5 MOTOR DRIVER (L293D)

L293D is a typical Motor driver or Motor Driver IC which allows DC motor to drive on either direction. L293D is a 16-pin IC which can control a set of two DC motors simultaneously in any direction. It means that you can control two DC motor with a single L293D IC.

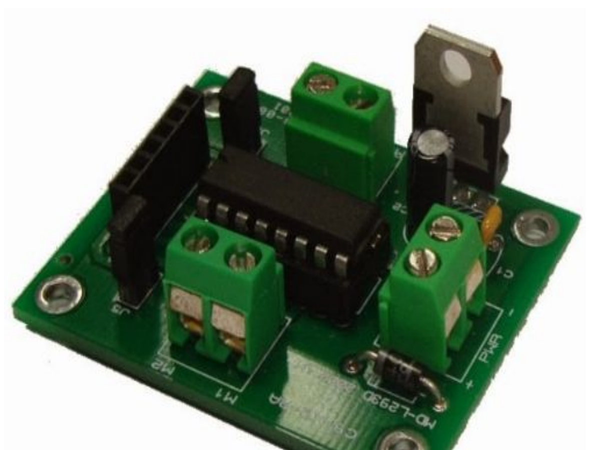


Fig. 2.7 L293D Motor driver

L293D is a dual H-bridge motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher-current signal. This higher current signal is used to drive the motors.

2.5.1 PIN CONFIGURATION

Fig. 2.8 Pin diagram of L293D IC

2.5.2 Pin description

Pin No	**Function**	**Name**
1	Enable pin for Motor 1; active high	Enable 1,2
2	Input 1 for Motor 1	Input 1
3	Output 1 for Motor 1	Output 1
4	Ground (0V)	Ground
5	Ground (0V)	Ground
6	Output 2 for Motor 1	Output 2
7	Input 2 for Motor 1	Input 2
8	Supply voltage for Motors; 9-12V (up to 36V)	Vcc_2
9	Enable pin for Motor 2; active high	Enable 3,4
10	Input 1 for Motor 1	Input 3
11	Output 1 for Motor 1	Output 3
12	Ground (0V)	Ground
13	Ground (0V)	Ground
14	Output 2 for Motor 1	Output 4
15	Input2 for Motor 1	Input 4
16	Supply voltage; 5V (up to 36V)	Vcc_1

Table 2.2 Pin Description of Motor Driver

2.5.3 PRINCIPLE OF OPERATION

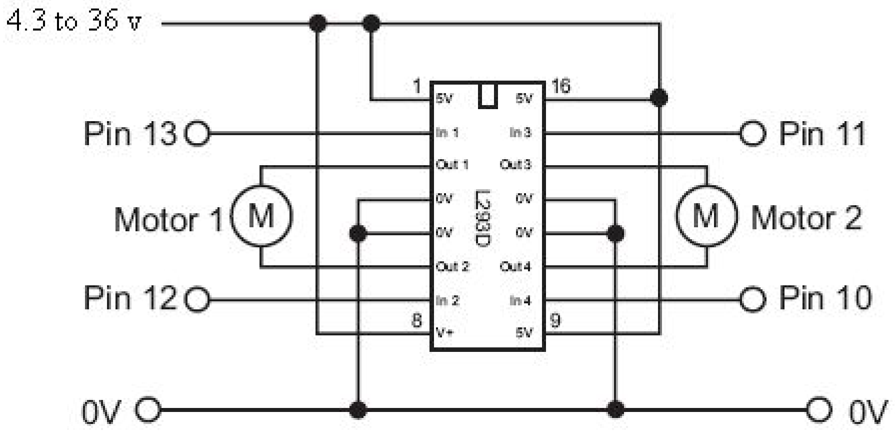


Fig 2.9 Circuit Diagram For l293d motor driver IC controller

It works on the concept of H-bridge. H-bridge is a circuit which allows the voltage to be flown in either direction. As you know voltage need to change its direction for being able to rotate the motor in clockwise or anticlockwise direction, Hence H-bridge IC are ideal for driving a DC motor.

In a single L293D chip there are two h-Bridge circuit inside the IC which can rotate two dc motor independently. Due its size it is very much used in robotic application for controlling DC motors. Given below is the pin diagram of a L293D motor controller.There are two Enable pins on l293d. Pin 1 and pin 9, for being able to drive the motor, the pin 1 and 9 need to be high. For driving the motor with left H-bridge you need to enable pin 1 to high. And for right H-Bridge you need to make the pin 9 to high. If anyone of the either pin1 or pin9 goes low then the motor in the corresponding section will suspend working. It’s like a switch.

There are 4 input pins for l293d, pin 2,7 on the left and pin 15 ,10 on the right as shown on the pin diagram. Left input pins will regulate the rotation of motor connected across left side and right input for motor on the right hand side. The motors are rotated on the basis of the inputs provided across the input pins as LOGIC 0 or LOGIC 1

2.6 IR SENSOR

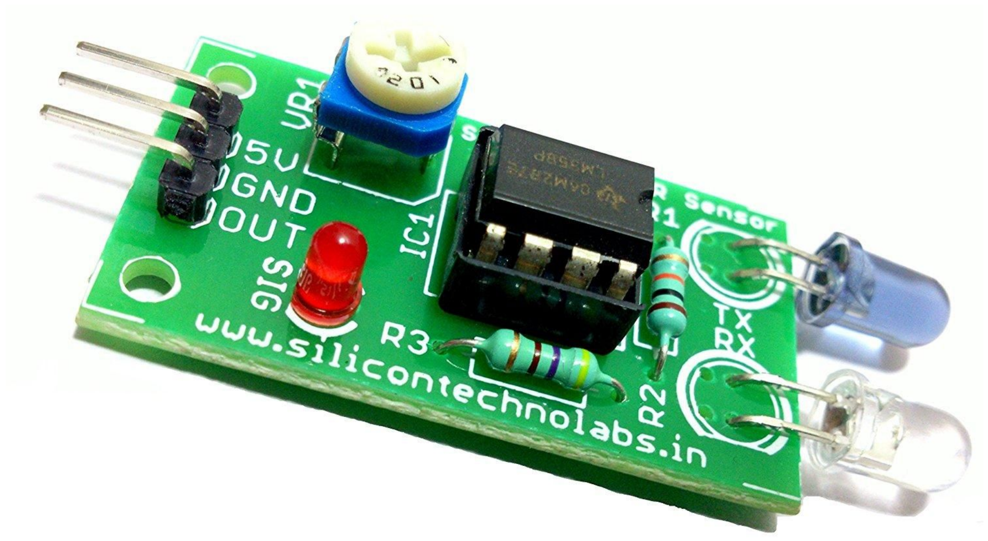


Fig 2.10 IR Sensor

This sensor can be used for most indoor applications where no important ambient light is present. For simplicity, this sensor doesn't provide ambient light immunity, but a more complicated, ambient light ignoring sensor should be discussed in a coming article. However, this sensor can be used to measure the speed of object moving at a very high speed, like in industry or in tachometers. In such applications, ambient light ignoring sensor, which rely on sending 40 Khz pulsed signals cannot be used because there are time gaps between the pulses where the sensor is 'blind'… The solution proposed doesn't contain any special components, like photo-diodes, photo-transistors, or IR receiver ICs, only a couple if IR leds, an Op amp, a transistor and a couple of resistors. In need, as the title says, a standard IR led is used for the purpose of detection. Due to that fact, the circuit is extremely simple, and any novice electronics hobbyist can easily understand and build it.

2.6.1 PRINCIPLE OF OPERATION

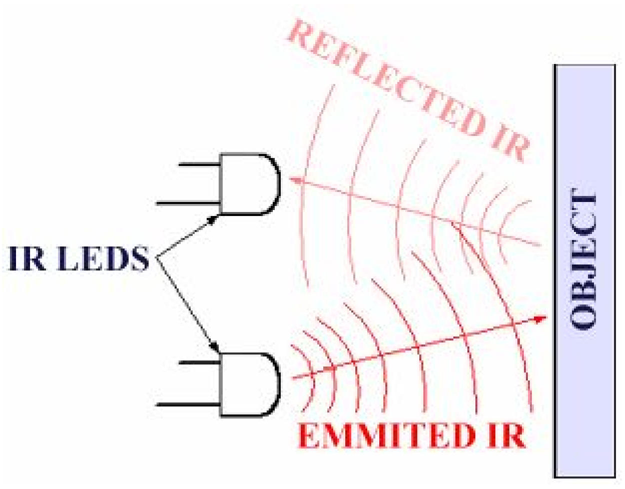


Fig. 2.11 IR operation

It is the same principle in all Infra-Red proximity sensors. The basic idea is to send infra red light through IR-LEDs, which is then reflected by any object in front of the sensor (figure 1).

Then all you have to do is to pick-up the reflected IR light**.** For detecting the reflected IR light, we are going to use a very original technique: we are going to use another IR-LED, to detect the IR light that was emitted from another led of the exact same type! This is an electrical property of Light Emitting Diodes (LEDs) which is the fact that a led produce a voltage difference across its leads when it is subjected to light. As if it was a photo-cell, but with much lower output current.

In other words, the voltage generated by the leds can't be in any way used to generate electrical power from light, it can barely be detected. that's why as you will notice in the schematic, we are going to use a Op-Amp (operational Amplifier) to accurately detect very small voltage changes.

CHAPTER 3

METHODOLOGY

3.1 Algorithm for motion detection

- Background subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.
- As the name suggests, BS calculates the foreground mask performing a subtraction between the current frame and a background model, containing the static part of the scene or, more in general, everything that can be considered as background given the characteristics of the observed scene.

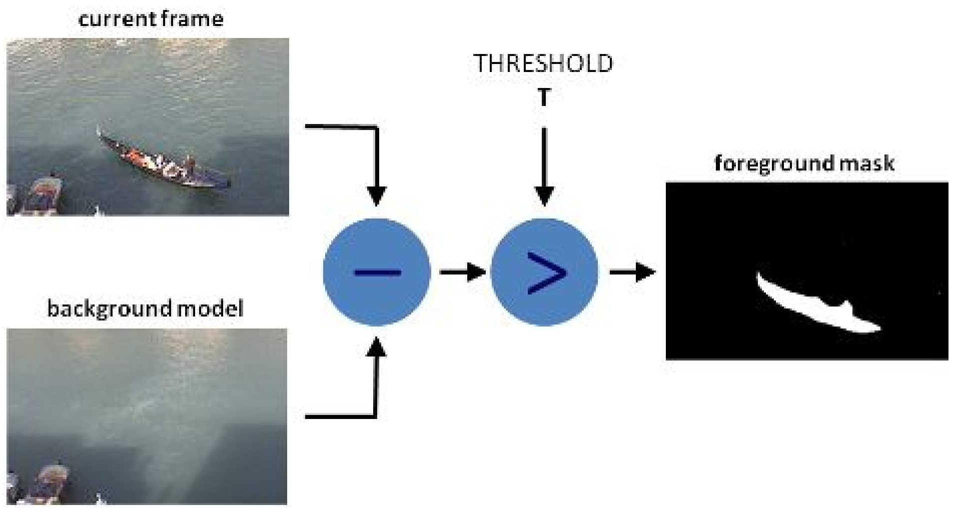


Fig. 3.1 background subtraction

Background modelling consists of two main steps:

1. Background Initialization;
2. Background Update.

In the first step, an initial model of the background is computed, while in the second step that model is updated in order to adapt to possible changes in the scene.

3.1.1 Using frame differencing

A motion detection algorithm begins with the segmentation part where foreground or moving objects are segmented from the background. The simplest way to implement this is to take an image as background and take the frames obtained at the time t, denoted by I(t) to compare with the background image denoted by B. Here using simple arithmetic calculations, we can segment out the objects simply by using image subtraction technique of computer vision meaning for each pixels in I(t), take the pixel value denoted by P[I(t)] and subtract it with the corresponding pixels at the same position on the background image denoted as P[B].

In mathematical equation, it is written as

$$P[F(t)] = P[I(t)] - P(B)$$

The background is assumed to be the frame at time *t*. This difference image would only show some intensity for the pixel locations which have changed in the two frames. Though we have seemingly removed the background, this approach will only work for cases where all foreground pixels are moving and all background pixels are static. A threshold "Threshold" is put on this difference image to improve the subtraction.

$$|P[F(t)] - P[F(t+1)]| > Threshold$$

This means that the difference image's pixels' intensities are 'thresholded' or filtered on the basis of value of Threshold. The accuracy of this approach is dependent on speed of movement in the scene. Faster movements may require higher thresholds.

3.1.2 Mean filter

For calculating the image containing only the background, a series of preceding images are averaged. For calculating the background image at the instant *t*,

$$B(x,y,t) = \tfrac{1}{N} \sum_{i=1}^{N} V(x,y,t-i) .$$

where *N* is the number of preceding images taken for averaging. This averaging refers to averaging corresponding pixels in the given images. *N* would depend on the video speed (number of images per second in the video) and the amount of movement in the video. After calculating the background *B(x,y,t)* we can then subtract it from the image *V(x,y,t)* at time *t*=t and threshold it. Thus the foreground is

$$|V(x,y,t) - B(x,y,t)| > Th$$

where *Th* is threshold. Similarly we can also use median instead of mean in the above calculation of *B*(*x*,*y*,*t*).

Usage of global and time-independent Thresholds (same Th value for all pixels in the image) may limit the accuracy of the above two approaches.

For this method, Wren et al propose fitting a Gaussian probabilistic density function (pdf) on the most recent frames. In order to avoid fitting the pdf from scratch at each new frame time t , a running (or on-line cumulative) average is computed.

The pdf of every pixel is characterized by mean μ_t and variance $\sigma_t^{\,2}$. The following is a possible initial condition (assuming that initially every pixel is background):

$\mu_o = I_o$

$\sigma_o^{\,2}$ =< some default value>

where $I_t$is the value of the pixel's intensity at time . In order to initialize variance, we can, for example, use the variance in x and y from a small window around each pixel.

Note that background may change over time (e.g. due to illumination changes or non-static background objects). To accommodate for that change, at every frame t, every pixel's mean and variance must be updated, as follows:

$\mu_t = \rho\, I_t + (1 - \rho)\, \mu_{t-1}$

$\sigma_t^2 = d^2\rho + (1 - \rho)\, \sigma_{t-1}^{\,2}$

$d = |\,(\,I_t - \mu_t\,)\,|$

Where determines the size of the temporal window that is used to fit the pdf (usually) and is the Euclidean distance between the mean and the value of the pixel.

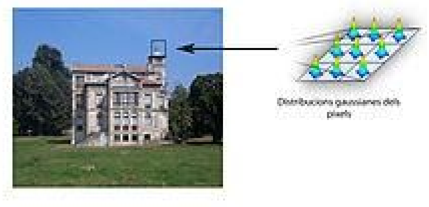


Fig 3.2 Gaussian distribution for each pixel.

We can now classify a pixel as background if its current intensity lies within some confidence interval of its distribution's mean:

$|(I_t - \mu_t)| \, \sigma_t > k \longrightarrow$ Foreground

$|(I_t - \mu_t)| \, \sigma_t \leq k \longrightarrow$ Background

where the parameter k is a free threshold (usually k=2.5). A larger value for allows for more dynamic background, while a smaller increases the probability of a transition from background to foreground due to more subtle changes.

In a variant of the method, a pixel's distribution is only updated if it is classified as background. This is to prevent newly introduced foreground objects from fading into the background. The update formula for the mean is changed accordingly:

$\mu_t = M \mu_{t-1} + (1 - M) (I_t \rho + (1 - \rho) \mu_{t-1})$

where M=1 when I_t is considered foreground and M=0 otherwise. So when M=1, that is, when the pixel is detected as foreground, the mean will stay the same. As a result, a pixel, once it has become foreground, can only become background again when the intensity value gets close to what it was before turning foreground. This method, however, has several issues: It only works if all pixels are initially background pixels (or foreground pixels are annotated as such). Also, it cannot cope with gradual background changes: If a pixel is categorized as foreground for a too long period of time, the background intensity in that location might have changed (because illumination has changed etc.). As a result, once the foreground object is gone, the new background intensity might not be recognized as such anymore.

3.2 Automatic License Plate Recognition

Fig 3.3 licence plate recognition

3.2.1 Block Diagram

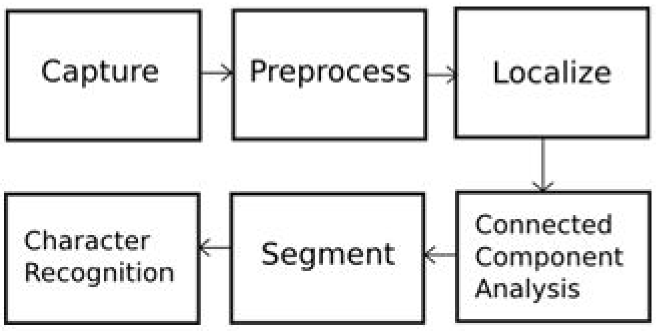


Fig 3.4 block diagram of license plate recognition

A.Capture

The image of the vehicle is captured using a high resolution photographic camera. A better choice is an Infrared(IR) camera. The camera maybe rolled and pitched with respect to the license plates. Character recognition is generally very sensitive to the skew. The readable characters can become distorted due to the obliqueness of the camera. Using a better camera with more definition and resolution will increase the success ratio of the system.

B. Preprocess

Preprocessing is the set algorithms applied on the image to enhance the quality. It is an important and common phase in any computer vision system. For the present system preprocessing involves two processes: Resize –The image size from the camera might be large and can drive the system slow. It is to be resized to a feasible aspect ratio. Convert Colour Space–Images captured using IR or photo- graphic cameras will be either in raw format or encoded into some multimedia standards. Normally, these images will be in RGB mode, with three channels (viz. red, green and blue). Number of channels defines the amount colour information available on the image. The image has to be converted to gray scale.

C. Localize

Rear or front part of the vehicle is captured into an image. The image certainly contains other parts of the vehicle and the environment, which are of no requirement to the system. The area in the image that interests us is the license plate and needs to be localized from the noise. Localization is basically a process of binarizing the image. As shown in Fig.3.5, the image is converted to black and white. There are two motivations for this operation–

1. Highlighting characters

2. Suppressing background.

Localization is done by an image processing technique called Thresholding. The pixels of the image are truncated to two values depending upon the value of threshold. Threshold requires pre-image analysis for identifying the suitable threshold value. Adaptive thresholding technique determines a local optimal threshold value for each image pixel so as to avoid the problem originating from non-uniform illumination.

D. Connected Component Analysis

In order to eliminate undesired image areas, a connected component algorithm is first applied to the binarized plate candidate. Connected component analysis is performed to identify the characters in the image. Basic idea is to traverse through the image and find the connected pixels. Each of the connected components(blobs) are labelled and extracted.

E. Segmentation

Segmentation is the process of cropping out the labelled blobs. These blobs are expected to be the required portion of the license number. A special algorithm called Image Scissoring is introduced here. In this algorithm, the license plate is vertically scanned and scissored at the row on which there is no white pixel and the scissored area is copied into a new matrix, there are unwanted blobs even after segmentation. These are classified using special algorithms.

F. Character Recognition

Finally, the selected blobs are send to a Optical Character Recognition (OCR) Engine, which returns the ASCII of the license number.

3.2.2 KNN Algorithm

In pattern recognition, the ***k*-nearest neighbours algorithm** (***k*-NN**) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression:

- In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its *k* nearest neighbours (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbour.

- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all machine learning algorithms.

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, *k* is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the **overlap metric** (or Hamming distance). In the context of gene expression microarray data, for example, *k*-NN has also been employed with correlation coefficients such as Pearson and Spearman.[3] Often, the classification accuracy of *k*-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the *k* nearest neighbors due to their large number.[4] One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its *k* nearest neighbors. The class (or value, in regression problems) of each of the *k* nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in a self-organizing map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. *K*-NN can then be applied to the SOM.

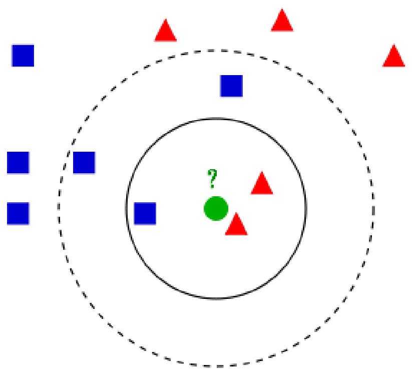


Fig. 3.5 KNN algorithm

Example of *k*-NN classification is as shown in fig.3.5 the test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If *k = 3* (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If *k = 5* (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

3.2.3 Parameter Selection:

The best choice of *k* depends upon the data; generally, larger values of *k* reduce the effect of noise on the classification, but make boundaries between classes less distinct. A good *k* can be selected by various heuristic techniques (see hyper parameter optimization). The special case where the class is predicted to be the class of the closest training sample (i.e. when *k* = 1) is called the nearest neighbour algorithm.

The accuracy of the *k*-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance. Much research effort has been put into selecting or scaling features to improve classification. A particularly popular approach is the use of evolutionary algorithms to optimize feature scaling. Another popular approach is to scale features by the mutual information of the training data with the training classes.

In binary (two class) classification problems, it is helpful to choose *k* to be an odd number as this avoids tied votes. One popular way of choosing the empirically optimal *k* in this setting is via bootstrap method.

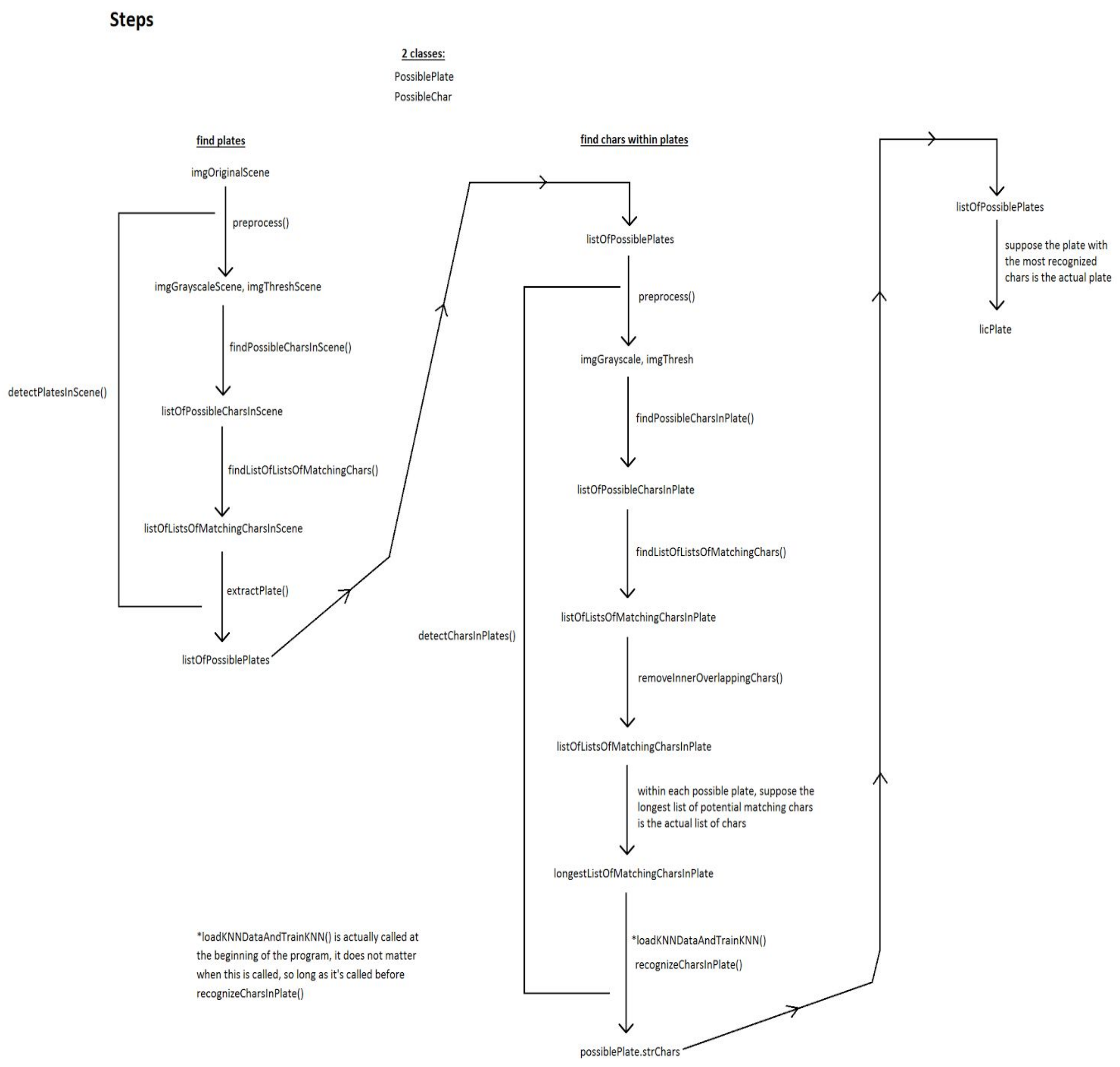
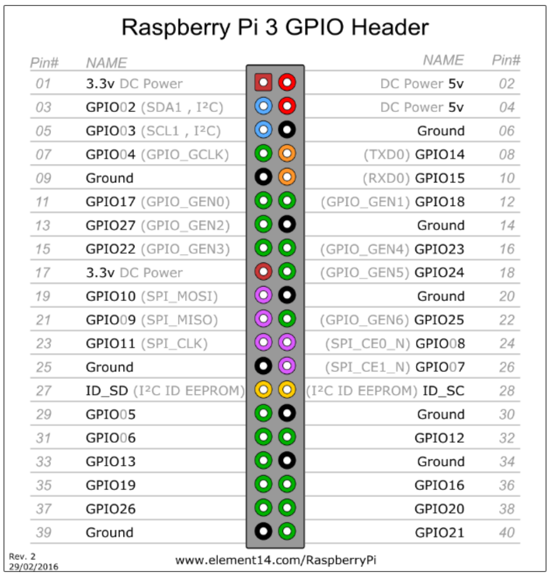

Steps
2 classes:
PossiblePlate
PossibleChar
find plates
imgOriginalScene
preprocess()
imgGrayscaleScene, imgThreshScene
findPossibleCharsInScene()
detectPlatesInScene()
listOfPossibleCharsInScene
findListOfListsOfMatchingChars()
listOfListsOfMatchingCharsInScene
extractPlate()
listOfPossiblePlates
find chars within plates
listOfPossiblePlates
preprocess()
imgGrayscale, imgThresh
findPossibleCharsInPlate()
listOfPossibleCharsInPlate
findListOfListsOfMatchingChars()
listOfListsOfMatchingCharsInPlate
detectCharsInPlates()
removeInnerOverlappingChars()
listOfListsOfMatchingCharsInPlate
within each possible plate, suppose the longest list of potential matching chars is the actual list of chars
longestListOfMatchingCharsInPlate
*loadKNNDataAndTrainKNN() is actually called at the beginning of the program, it does not matter when this is called, so long as it's called before recognizeCharsInPlate()
*loadKNNDataAndTrainKNN()
recognizeCharsInPlate()
possiblePlate.strChars
listOfPossiblePlates
suppose the plate with the most recognized chars is the actual plate
licPlate

Fig 3.6 steps of parameter selection

3.2.4 Obstacle Detection



Raspberry Pi 3 GPIO 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

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

Fig 3.7. Raspberry pi 3 pin configuration

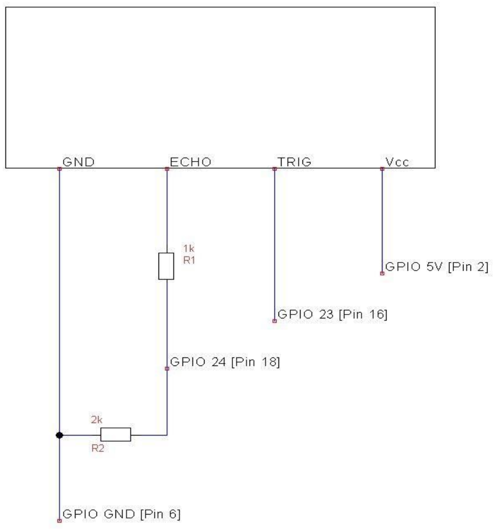


Fig. 3.8 Ultrasonic sensor HC-SR04 to Raspberry pi

Attached is a schematic which shows how to apply the voltage divider.

Briefly, this is used to drop the voltage on the signal back from the ECHO pin on the sensor. 5V leaves the pin and goes through a resistor which creates a voltage drop. At this point, there is a split - one branch goes to GND through a resistor which takes away twice as much voltage - the other branch has only gone through one of these resistors and so only reduces the voltage to 3.3V.

The Pi needs this signal to be reduced to 3.3V otherwise the US sensor will work but will return spurious readings. Thus the need for the voltage divider.

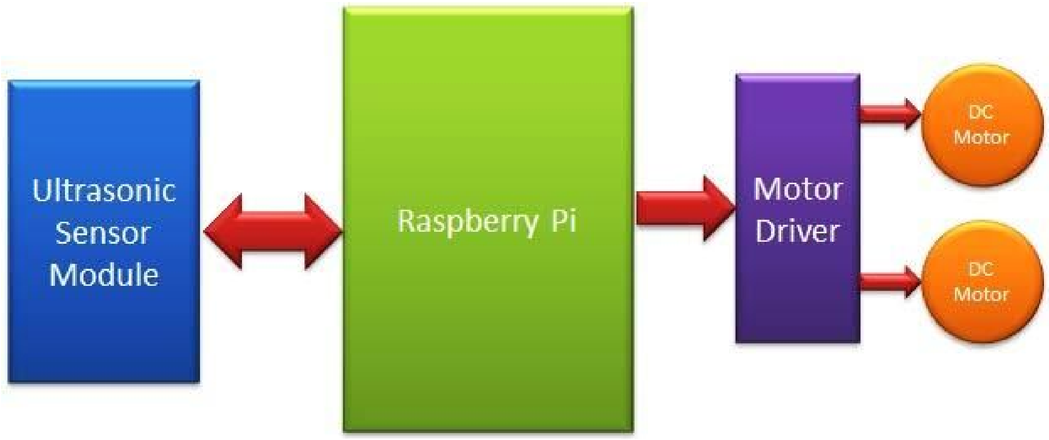


Fig 3.9 Obstacle avoiding using Ultrasonic sensor

When the Robot is powered on and starts running, Raspberry Pi measures the distances of objects, in front of it, by using Ultrasonic Sensor Module and stores in a variable. Then RPi compares this value with predefined values and take decisions accordingly to move the Robot Left, Right, Forward, or backward.

Here in this project, **15cm distance is selected for taking any decision** by Raspberry Pi. Now whenever Raspberry Pi gets less than the 15cm distance from any object then Raspberry Pi stops the robot and moves it back and then turns it left or right. Now before moving it forward again, Raspberry Pi again checks whether any obstacle is present within the range of 15 cm distance, if yes then again repeats the previous process, else move the robot forward until it will detect any obstacle or object again.

3.2.5 Object Tracking

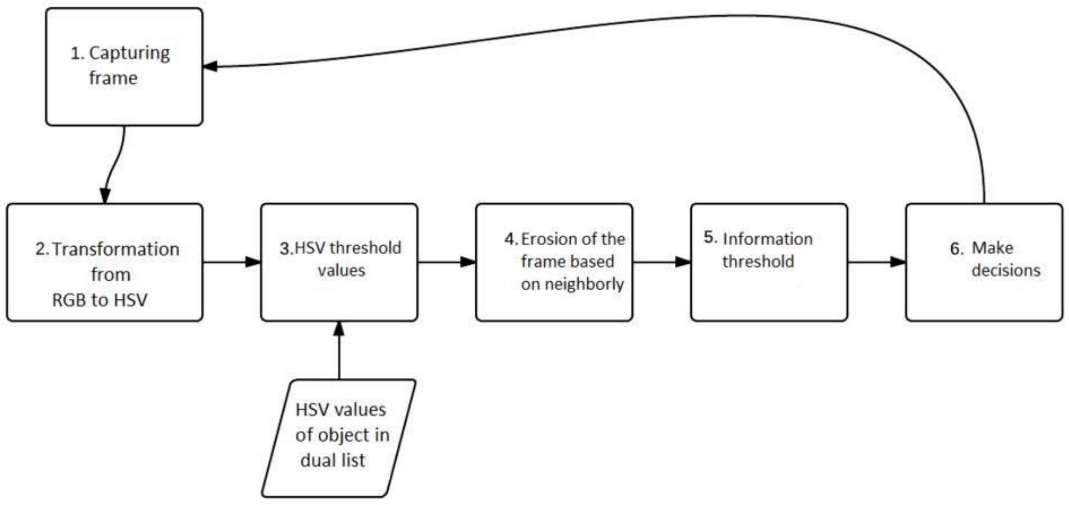

1. Capturing frame
2. Transformation from RGB to HSV
3.HSV threshold values
4. Erosion of the frame based on neighborly
5. Information threshold
6. Make decisions
HSV values of object in dual list

Fig 3.10 Computer Vision System(CVS) steps for object tracking

1. First it was necessary to capture (or receive) the image or, specifically the frame containing the image (frame). The size is **160x120** pixels. The frame at large (eg, 640 pixels wide and 480 pixels high), caused slowdowns in the recognition process when the image was transmitted remotely.The system default is RGB color, this color system is represented in the webcam frame obtained through the basic colors: red (Red), Green (Green) and blue (Blue). These colors are represented on a pixel by pixel dimensional vector, for example, the color red is represented 0com values (0, 255, 0), respectively represented for each channel. That is, each pixel has its RGB value represented by three bytes (red green and blue).
2. After the captured image, the conversion from **RGB** color system to the color **HSV (hue, saturation, and value)** was undertaken, since this model describes similar to the recognition by the human eye colors. Since the **RGB (red, green and blue)** system has the colors based on combinations of the primary colors (red, green and blue) and the HSV system defines colors as their color, sparkle and shine (hue, saturation, and value), facilitating the extraction of information. In diagram the step 2 shows the conversion from RGB to HSV, using the "**cvtColor**" native OpenCV, which converts the input image from an input color system to another function.
3. With the image in **HSV** model, it was necessary to find the correct values of HSV minimum and maximum color of the object that will be followed. To save these values, were made two vectors with minimal HSV and HSV maximum color object as values: **minimum Hue (42) Minimum saturation (62) Minimum brightness (63) Maximum Hue (92) Maximum Saturation (255) Maximum Brightness (235)**. So the next step to generate a binary image, the relevant information may be limited only

in the context of these values. These values are needed to limit the color pattern of the object. A function of comparing the pixel values with the standard values of the inserted vector was used. The result was a binary image providing only one value for each pixel.

4. Having made the segmentation, resulting in the binary image, it is noted that noise are still present in the frame. These noises are elements that hinder the segmentation (including obtaining the actual size) of the object. To fix (or attempt to fix) this problem, it was necessary to apply a morphological transformation through operators in the frame, so that the pixels were removed that did not meet the desired standard. For this, the morphological operator EROSION, who performed a "clean" in the frame, reducing noise contained in it was used.
5. Then it was used to **"Moments"** function, which calculates the moments of positive contour (white) using an integration of all pixels present in the contour. This feature is only possible in a frame already binarizado and without noise, so that the size of the contour of the object is not changed by stray pixels in the frame, which hinder and cause redundancy in information.
   moments = cv2.moments (imgErode, True)

   In the proposed example, it was necessary to find the area of the contour and its location coordinates in the frame to be made the calculations of repositioning the chassis. The calculation of the area of the object performs the binary sum of positive, generating the variable **M00** and recorded in the variable "area":
   area = moments ['m00']
   The specificity of the contour refers to an object, not a polygon. This value is found an approximate area of positive pixels (white) that make up the object. If this value is null area, is disregarded the existence of an object color treated (if the "green" color) in the frame. Using this feature will help accomplish the movement of the robot approaching and distancing of the target object, trying to treat the problem of depth. That is, the fact that the object is approaching or distancing overly chassis.

   And from the targeted area was possible, define the coordinates of the object in this frame. For the coordinates of the object was used parameters obtained Moments function that found its coordinated. But this was coordinated based on centroid of the object, is found only if the area of the object is greater than zero. Using this feature was important to make the movement of horizontal and vertical adjustment of the robot in order to increase the degree of freedom and minimize restriction of movement of the object to be identified. Using the area of the object parameter and combined with M00 x and y parameters Moments of function, it was possible to find the coordinates (x, y).

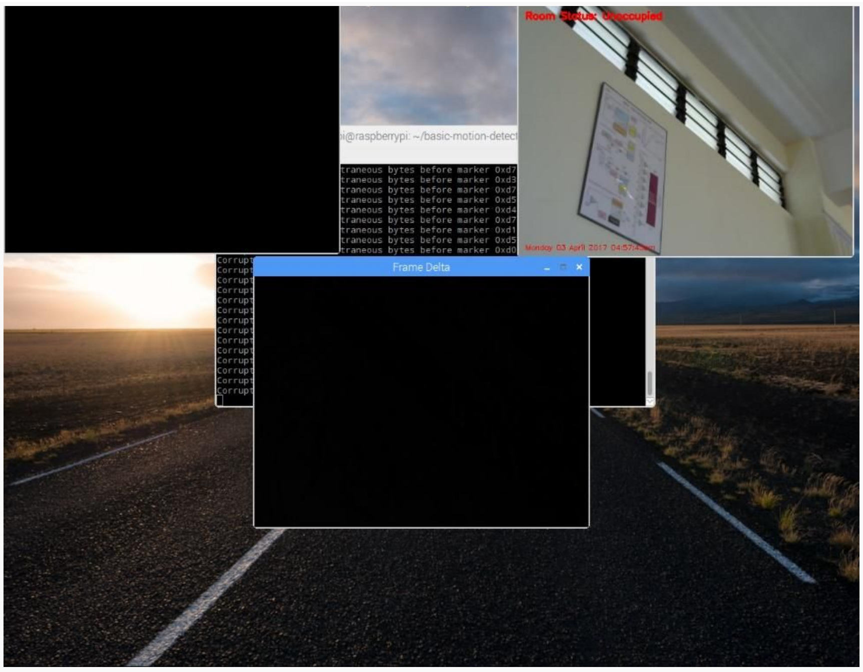
   Thus the values received in the coordinate (x, y) refers to the placement of the found segmentation of the object relative to the frame and to facilitate the interpretation of the information which is being drawn from the coordinate information, a function that **draws a circle** at the centroid was applied the object.

CHAPTER 4

RESULTS

This unmanned surveillance robot uses techniques of image processing to detect any changes in some particular area monitored by it. Operation of this robot includes three steps, first is to move in a given particular area with randomness, observe changes in the environment and notifies the control station if necessary changes have been detected.

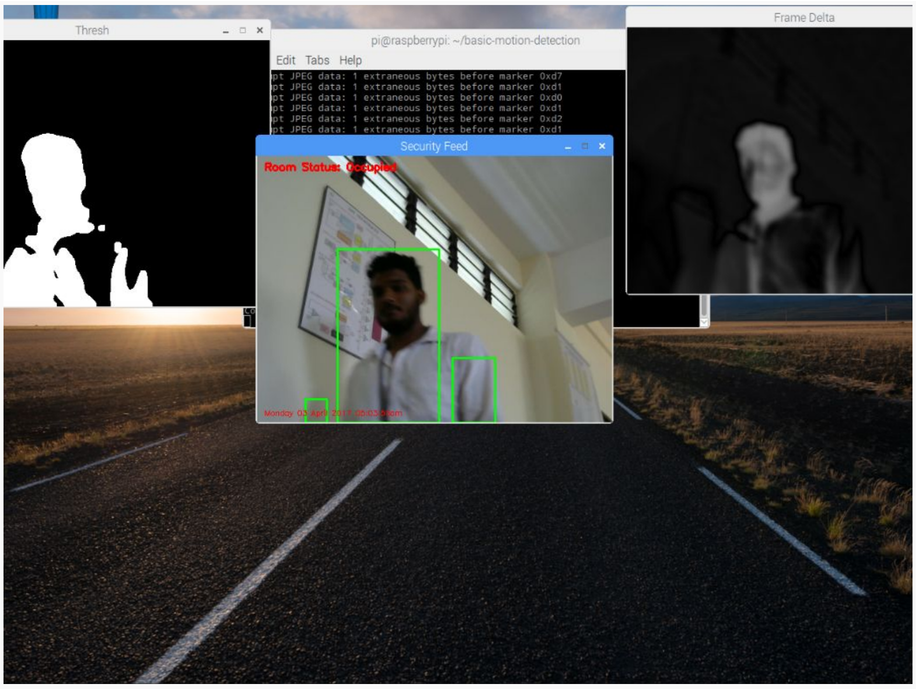
- Random movement is given by chained robot which will roam the entire given area with randomness.
- Changes in the environment is detected using openCV, which detects whether the given particular area is occupied or not. Figure 4.1 shows an **unoccupied** image, where the particular area is empty. Figure 4.2 shows **occupied** image, where a motion is being detected.



```
i@raspberrypi: ~/basic-motion-detect
```

```
traneous bytes before marker 0xd7
traneous bytes before marker 0xd3
traneous bytes before marker 0xd7
traneous bytes before marker 0xd5
traneous bytes before marker 0xd4
traneous bytes before marker 0xd7
traneous bytes before marker 0xd1
traneous bytes before marker 0xd5
traneous bytes before marker 0xd0
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
Corrupt
```

Fig 4.1 Unoccupied


Thresh
pi@raspberrypi: ~/basic-motion-detection
Edit Tabs Help
Security Feed
Room Status: Occupied
Monday 03 April 2017 05:03:56pm
Frame Delta

```
pt JPEG data: 1 extraneous bytes before marker 0xd7
pt JPEG data: 1 extraneous bytes before marker 0xd1
pt JPEG data: 1 extraneous bytes before marker 0xd0
pt JPEG data: 1 extraneous bytes before marker 0xd1
pt JPEG data: 1 extraneous bytes before marker 0xd2
pt JPEG data: 1 extraneous bytes before marker 0xd1
```

Fig 4.2 Occupied

Another application is to provide parking surveillance. Parking surveillance system is another feature which will identify the number plate of the vehicle approaching to the parking lobby and verifies for wrong parking in a parking spot.

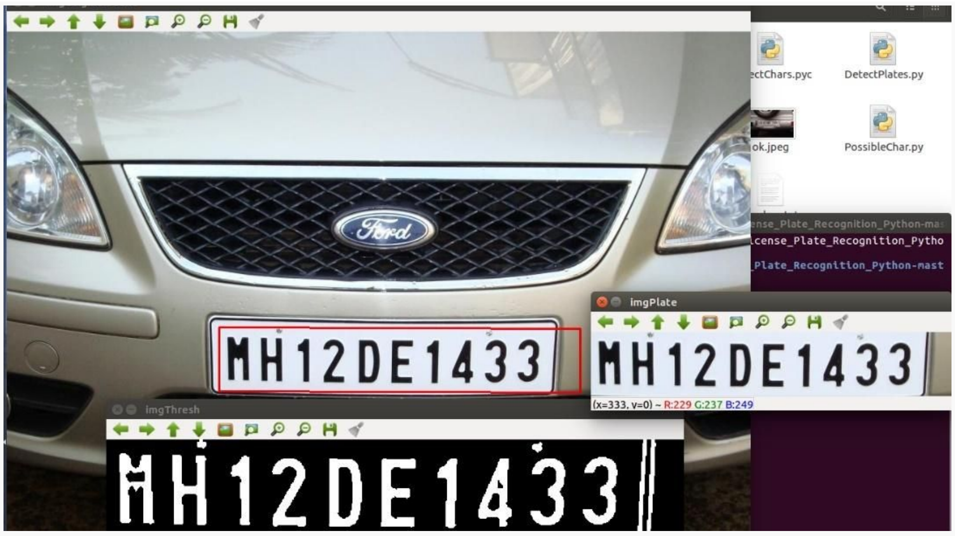


Fig 4.3 Licence plate detection

CHAPTER 5

FUTURE WORK

- Robot can be designed to operate on uneven surfaces and stairs.
- Artificial intelligence can be given to the robot.
- Robot can be connected to the cloud and share data between similar robots on surveillance.

CHAPTER 6

APPLICATIONS

- Unmanned Surveillance
- Border Security
- Parking Surveillance
- Courier-Bot

CHAPTER 7

CONCLUSION

This system can be effectively made use in military application, offices and also as a home security system required by the user. It also provides a wide opportunity for various extensions according to the user requirements. Since it carries a given network with it, it can be accessed from wherever the user can manage within a given radius and if the user runs out of range his partner can take up the controls using the portable device software.

It provides real time video streaming for reconnaissance and surveillance operations. By taking the action against received data, any desired surveillance operation can be done.

Although it's not a finished product in some ways, it can be used as a prototype to develop better, more effective machines(eg: equipped with GSM, Intelligence, cloud etc.) which provide higher security sensitive levels, greater range and more secure body.

BIBLIOGRAPHY

[1]. Elgammal, Ahmed, David Harwood, and Larry Davis. "Non-parametric model for background subtraction." *Computer Vision—ECCV 2000* (2000): 751-767.

[2]. Huang, Shih-Chia. "An advanced motion detection algorithm with video quality analysis for video surveillance systems." *IEEE transactions on circuits and systems for video technology* 21.1 (2011): 1-14.

[3]. Jacob Goldberger, Sam Roweis, and Ruslan Salakhutdinov Geoff Hinton. "Neighbourhood components analysis." *NIPS'04* (2004).

[4]. Soucy, Pascal, and Guy W. Mineau. "A simple KNN algorithm for text categorization." *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, 2001.

[5]. Du, Shan, et al. "Automatic license plate recognition (ALPR): A state-of-the-art review." *IEEE Transactions on Circuits and Systems for Video Technology* 23.2 (2013): 311-325.

[6]. Park, Myoungkuk, et al. "Performance Guarantee of an Approximate Dynamic Programming Policy for Robotic Surveillance." *IEEE Transactions on Automation Science and Engineering* 13.2 (2016): 564-578.

[7]. Yadav, Sanjana, and Archana Singh. "An image matching and object recognition system using webcam robot." *Parallel, Distributed and Grid Computing (PDGC), 2016 Fourth International Conference on*. IEEE, 2016.

APPENDIX (SOURCE CODE)

MOTION DETECTION:

```
# python motion_detector.py
# python motion_detector.py --video videos/example_01.mp4

# import the necessary packages
importargparse
importdatetime
importimutils
importtime
```

```
importcv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=500, help="minimum area size")
args = vars(ap.parse_args())

# if the video argument is None, then we are reading from webcam
ifargs.get("video", None) isNone:
	camera = cv2.VideoCapture(0)
	time.sleep(0.25)

# otherwise, we are reading from a video file
else:
	camera = cv2.VideoCapture(args["video"])

# initialize the first frame in the video stream
firstFrame = None

# loop over the frames of the video
whileTrue:
	# grab the current frame and initialize the occupied/unoccupied
	# text
	(grabbed, frame) = camera.read()
	text = "Unoccupied"

	# if the frame could not be grabbed, then we have reached the end
	# of the video
	ifnot grabbed:
		break

	# resize the frame, convert it to grayscale, and blur it
	frame = imutils.resize(frame, width=500)
	gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
	gray = cv2.GaussianBlur(gray, (21, 21), 0)

	# if the first frame is None, initialize it
	iffirstFrameisNone:
		firstFrame = gray
		continue

	# compute the absolute difference between the current frame and
	# first frame
	frameDelta = cv2.absdiff(firstFrame, gray)
	thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

	# dilate the thresholded image to fill in holes, then find contours
	# on thresholded image
	thresh = cv2.dilate(thresh, None, iterations=2)
	(cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
		cv2.CHAIN_APPROX_SIMPLE)
	# loop over the contours
	for c incnts:
		# if the contour is too small, ignore it
		ifcv2.contourArea(c) <args["min_area"]:
			continue

		# compute the bounding box for the contour, draw it on the frame,
		# and update the text
		(x, y, w, h) = cv2.boundingRect(c)
		cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
		text = "Occupied"

	# draw the text and timestamp on the frame
	cv2.putText(frame, "Room Status: {}".format(text), (10, 20),
		cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
	cv2.putText(frame, datetime.datetime.now().strftime("%A %d %B %Y %I:%M:%S%p"),
		(10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255),
1)

	# show the frame and record if the user presses a key
	cv2.imshow("Security Feed", frame)
	cv2.imshow("Thresh", thresh)
	cv2.imshow("Frame Delta", frameDelta)
	key = cv2.waitKey(1) & 0xFF

	# if the `q` key is pressed, break from the lop
	if key == ord("q"):
		break

# cleanup the camera and close any open windows
camera.release()
cv2.destroyAllWindows()
```

LICENCE PLATE RECOGNITION:

```
# Main.py

importcv2
```

```
importnumpyasnp
importos

importDetectChars
importDetectPlates
importPossiblePlate

# module level variables
SCALAR_BLACK = (0.0, 0.0, 0.0)
SCALAR_WHITE = (255.0, 255.0, 255.0)
SCALAR_YELLOW = (0.0, 255.0, 255.0)
SCALAR_GREEN = (0.0, 255.0, 0.0)
SCALAR_RED = (0.0, 0.0, 255.0)

showSteps = False

def main():

blnKNNTrainingSuccessful = DetectChars.loadKNNDataAndTrainKNN()          # attempt KNN
training

ifblnKNNTrainingSuccessful == False:                          # if KNN training was not successful
print"\nerror: KNN traning was not successful\n"# show error message
return# and exit program
# end if

imgOriginalScene  = cv2.imread("1.png")             # open image

ifimgOriginalSceneisNone:                          # if image was not read successfully
print"\nerror: image not read from file \n\n"# print error message to std out
os.system("pause")                                # pause so user can see error message
return# and exit program
# end if

listOfPossiblePlates = DetectPlates.detectPlatesInScene(imgOriginalScene)          # detect plates

listOfPossiblePlates = DetectChars.detectCharsInPlates(listOfPossiblePlates)        # detect chars in
plates

cv2.imshow("imgOriginalScene", imgOriginalScene)          # show scene image

iflen(listOfPossiblePlates) == 0:                         # if no plates were found
print"\nno license plates were detected\n"# inform user no plates were found
else:                                                 # else
# if we get in here list of possible plates has at leat one plate

# sort the list of possible plates in DESCENDING order (most number of chars to least number of
chars)
listOfPossiblePlates.sort(key = lambdapossiblePlate: len(possiblePlate.strChars), reverse = True)

# suppose the plate with the most recognized chars (the first plate in sorted by string length
descending order) is the actual plate
```

```
licPlate = listOfPossiblePlates[0]

cv2.imshow("imgPlate", licPlate.imgPlate)          # show crop of plate and threshold of plate
cv2.imshow("imgThresh", licPlate.imgThresh)

iflen(licPlate.strChars) == 0:                    # if no chars were found in the plate
print"\nno characters were detected\n\n"# show message
return# and exit program
# end if

drawRedRectangleAroundPlate(imgOriginalScene, licPlate)             # draw red rectangle around
plate

print"\nlicense plate read from image = " + licPlate.strChars + "\n"# write license plate text to std
out
print"----------------------------------------"

writeLicensePlateCharsOnImage(imgOriginalScene, licPlate)           # write license plate text on
the image

cv2.imshow("imgOriginalScene", imgOriginalScene)            # re-show scene image

cv2.imwrite("imgOriginalScene.png", imgOriginalScene)         # write image out to file

# end if else

cv2.waitKey(0)                                    # hold windows open until user presses a key

return
# end main

defdrawRedRectangleAroundPlate(imgOriginalScene, licPlate):

    p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)            # get 4 vertices of
rotated rect

cv2.line(imgOriginalScene, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), SCALAR_RED, 2)
# draw 4 red lines
cv2.line(imgOriginalScene, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), SCALAR_RED, 2)
cv2.line(imgOriginalScene, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), SCALAR_RED, 2)
cv2.line(imgOriginalScene, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), SCALAR_RED, 2)
# end function
defwriteLicensePlateCharsOnImage(imgOriginalScene, licPlate):
ptCenterOfTextAreaX = 0                           # this will be the center of the area the text will be
written to
ptCenterOfTextAreaY = 0

ptLowerLeftTextOriginX = 0                        # this will be the bottom left of the area that the text
will be written to
ptLowerLeftTextOriginY = 0

sceneHeight, sceneWidth, sceneNumChannels = imgOriginalScene.shape
```

```
plateHeight, plateWidth, plateNumChannels = licPlate.imgPlate.shape

intFontFace = cv2.FONT_HERSHEY_SIMPLEX                  # choose a plain jane font
fltFontScale = float(plateHeight) / 30.0                # base font scale on height of plate area
intFontThickness = int(round(fltFontScale * 1.5))           # base font thickness on font scale

textSize,     baseline     =     cv2.getTextSize(licPlate.strChars,     intFontFace,     fltFontScale,
intFontThickness)       # call getTextSize

# unpack roatatedrect into center point, width and height, and angle
    ( (intPlateCenterX, intPlateCenterY), (intPlateWidth, intPlateHeight), fltCorrectionAngleInDeg
) = licPlate.rrLocationOfPlateInScene

intPlateCenterX = int(intPlateCenterX)              # make sure center is an integer
intPlateCenterY = int(intPlateCenterY)

ptCenterOfTextAreaX = int(intPlateCenterX)          # the horizontal location of the text area is the
same as the plate

ifintPlateCenterY< (sceneHeight * 0.75):                            # if the license plate is in
the upper 3/4 of the image
ptCenterOfTextAreaY = int(round(intPlateCenterY)) + int(round(plateHeight * 1.6))       # write
the chars in below the plate
else:                                                    # else if the license plate is in the lower
1/4 of the image
ptCenterOfTextAreaY = int(round(intPlateCenterY)) - int(round(plateHeight * 1.6))       # write
the chars in above the plate
# end if

textSizeWidth, textSizeHeight = textSize# unpack text size width and height

ptLowerLeftTextOriginX = int(ptCenterOfTextAreaX - (textSizeWidth / 2))          # calculate the
lower left origin of the text area
ptLowerLeftTextOriginY = int(ptCenterOfTextAreaY + (textSizeHeight / 2))         # based on the
text area center, width, and height

# write the text on the image
cv2.putText(imgOriginalScene,              licPlate.strChars,              (ptLowerLeftTextOriginX,
ptLowerLeftTextOriginY), intFontFace, fltFontScale, SCALAR_YELLOW, intFontThickness)
# end function

if __name__ == "__main__":
   main()
```