IMPLEMENTATION of IRIS SCANNING- by COLOUR DETECTION and SCALE INVARIANT FEATURE TRANSFORM(SIFT)

A Project report submitted in partial fulfillment of the requirements for

the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

S.V.V Sai Pradeep (317126512166) P. Mahendra (318126512L33)

V. Rohit (317126512174) P. Achyuth (317126512154)

Under the guidance of

G.GAYATRI

ASSISTANT PROFESSOR

PRAGNANAM BRAHMA

ANITS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES
(UGC AUTONOMOUS)
(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)
Sangivalasa, bheemili mandal, visakhapatnam dist.(A.P)
2020-2021

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES
(UGC AUTONOMOUS)

(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)

Sangivalasa, Bheemili mandal, Visakhapatnam dist.(A.P)

PRAGNANAM BRAHMA

ANITS

CERTIFICATE

This is to certify that the project report entitled "**IMPLEMENTATION of IRIS SCANNING-by COLOUR DETECTION and SCALE INVARIANT FEATURE TRANSFORM(SIFT)**" submitted by **S.V.V Sai Pradeep(317126512166), P.Mahendra (318126512L33), V.Rohit (317126512174), P. Achyuth (317126512154),** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Electronics & Communication Engineering** of Andhra University, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

[illegible]

Project Guide
G. GAYATRI

Assistant Professor
Department of E.C.E
ANITS

Assistant Professor
Department of E.C.E.
Anil Neerukonda
Institute of Technology & Sciences
Sangivalasa, Visakhapatnam-531 162

[illegible]

Head of the Department
Dr. V.Rajyalakshmi

Professor & HOD
Department of E.C.E
ANITS

Head of the Department
Department of E.C.E
Anil Neerukonda Institute of Technology & Sciences
Sangivalasa - 531 162

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **G.Gayatri** Assistant Professor, Department of Electronics and Communication Engineering, ANITS, for her guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr. V. Rajyalakshmi**, Head of the Department, Electronics and Communication Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa,** for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of ECE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **all non-teaching staff** of the Department of ECE, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS

S.V.V Sai Pradeep(317126512166),
P.Mahendra(318126512L33),
V.Rohit(317126512174),
P.Achyuth(317126512154)

ABSTRACT

Colour detection is the process of detecting the name of any colour. Human eyes and brains work together to translate light into colour. Light receptors that are present in our eyes transmit the signal to the brain. Our brain then recognizes the colour. But one of the limitations of human brain is that it cannot recognize all the shades of colours. But, through this process we can detect even the shades of most of the colours (approximately around 1500) by placing a target object for which the colour should be detected in front of the camera. We are formulating a process/algorithm through which we can produce faster results in Iris detection by first considering the colour of the Iris and then scanning the user data with the reduced data through scale invariant feature transform (SIFT). Iris is one of the unique features which can be distinguished between two different individuals. Since the Iris feature are unique, this can be used in Biometric applications. Fingerprint biometrics are efficient, but one need to move to Iris scanning for the biometrics because this method is more hygienic and accurate.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

CMOS	Complementary metal–oxide–semiconductor
CCD	Charged Coupled Device
PIL	Python Image Library
ML	Machine Learning
CV	Computer Vision
HSV	Hue Saturation Values
BGR	Blue Green Red
RGB	Red Green Blue
Collab	Collaboratory
API	Application Programming Interface
HTML	Hypertext markup language

CHAPTER 1

INTRODUCTION

A process for finding the colour of a sample is intended to develop firstly. Colour detection is the process of detecting the name of any colour. Human eyes and brains work together to translate light into colour. Light receptors that are present in our eyes transmit the signal to the brain. Our brain then recognizes the colour. But one of the limitations of human brain is that it cannot recognize all the shades of colours. Colours are made up of 3 primary colours; red, green, and blue. In computers, we define each colour value within a range of 0 to 255. So, we can define a colour in 256*256*256 = 16,581,375 ways. There are approximately 16.5 million different ways to represent a colour. Through this process we have targeted to detect most of the colours (approximately around 1500) and can display the name of the colour as the output and can be stored in data base, by giving a sample (Iris) through a camera as input. Using this technique, we are primarily focusing on detection of colour of the iris which is the main step in Iris scanning. Iris is one of the unique features that varies from person to person. It is unique to a level where the Iris features varies from one eye to the other of the same person. So, this unique identification feature can be used for biometric purposes. The problem with the conventional methods like fingerprint is that the error rate is present during biometric authentication and chance of duplication of fingerprint is there. In these cases, there will be a breach to the data integrity of the person. But we are still using the fingerprint biometric authentication because fingerprint biometrics are faster. Whereas on the other hand, even though the Iris biometrics is the more accurate and reliable than Fingerprint biometrics, searching of biometrics of a person through Iris can put high load on the processing system. So, to avoid this problem we are formulating a method to find the colour of the Iris. Before understanding this method, we need to know the biology of the Iris. The iris is a thin circular ring region, a part of the human eye positioned between the black pupil and white sclera presenting a unique and rich texture information, such as patterns, rifts, colours, rings, spots, stripes, filaments, coronal, furrows, minutiae and recess and other detailed characteristics seen under the infrared light. “Fig. 1” shows image of an Iris with all the patterns and texture. Out of all the features of an Iris mentioned above, we are concentrating on the colour pigment of the Iris. By using this process and first finding the colour and by screening the data available in the database based on identified colour, the data on which a system should verify to identify the biometrics of person an later through SIFT we can detect the actual biometrics of person by just matching with the reduced data due to which the system efficiency can improved. In this way the system capacity can be improved.

1.1 Project Overview:

In this paper is to improve system efficiency during Iris detection by preliminary screening of data on basis of colour. From the result obtained we could see that the process we have developed can identify the colour of the Iris. On the basis of these results, it is clear that we can improve the efficiency of the biometric systems as a particular person's biometric information is searched against only a limited set of data where the result colour matches with the colour of the Iris in the database, thereby improving the system efficiency. The motivation behind the formulating of this process is to develop more efficient biometrics system.

1.2 Literature Survey

Developing a high end security system for either identification or authentication purpose have always been an active research area and attractive goal in almost all fields. Traditional security systems provide security to a process or a product with the help of "something that we have or we know", i.e., a key or a password, whereas a biometric security system uses "something that we are", i.e., a person's physical or behavioral traits. Physiological or behavioural traits of a person may include, but not limited to the following: face, finger print, iris, retina, voice, DNA, gait, etc. Biometric traits have highly reliable and unique features that make it best suited for security systems over a traditional or conventional security system. Jain et al. (1999) [1], identified seven factors that could be used to identify a person's physical or behavioural characteristics as a biometric trait to be used in biometric security systems. They are (1) Universality (ease of availability in an individual), (2) Uniqueness (distinct characteristics), (3) Permanence (stability or durability), (4) Measurability or collectability (ease of acquisition), (5) Performance (quality of being efficient), (6) Acceptability (degree of approval) and (7) Circumvention (ease use of a substitute). Though a biometric trait cannot satisfy all of these, some of them must be satisfied to make a characteristic a biometric trait. For circumvent ability low is desirable instead of high. Based on the above observation, Iris satisfies almost all the factors with good score and hence used as a popular biometric trait in biometric recognition systems, among various other identifiers. Iris is a well protected muscle present inside the eye with unique and rich patterns like furrows, rings, freckles and crypts. It has a distinguishable colour, which is immutable and invariant over time. It has been proved that for an individual, there are differences in iris patterns even between right and left eye. Even iris patterns differ for twins, who are identical. Thus recognition techniques developed using iris patterns could be considered as a best suited identification and authentication technique, especially in areas like, physical or personal authentication systems, time and attendance maintenance systems, law enforcement systems, banking applications, etc.

1.3 Problem Definition:

Among the different biometric modalities like a fingerprint, finger vein, palm vein, facial, retina, etc. iris is one of the most reliable ones. Iris biometric authentication is the complex mathematical pattern recognition technique that identifies the unique and stable video images of the single or both irises of an individual which is possible to distinguish from a distance range.The characteristics of an iris are significantly unique for each and can be recognized from a distance. That is why it is challenging to be forged and compared to other biometric modalities false acceptance rate and the false rejection rate is remarkably lower. Such way it becomes most secured authentication technology and has been used in hospitals, borders, financial institutes, and several sensitive projects. With this many advantages it is of utmost necessity to improve the Iris efficiency for faster results.

CHAPTER 2

IRIS ANATOMY

The iris is a thin, annular structure in the eye, responsible for controlling the diameter and size of the pupil, thus the amount of light reaching the retina. Eye colour is defined by that of the iris. In optical terms, the pupil is the eye's aperture, while the iris is the diaphragm.

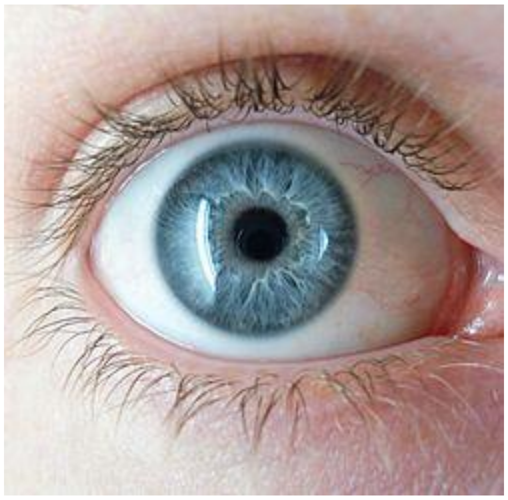


Fig.2.1 Human Iris

2.1 STRUCTURE

The iris consists of two layers: the front pigmented fibrovascular layer known as a stroma and, beneath the stroma, pigmented epithelial cells.

The stroma is connected to a sphincter muscle (sphincter pupillae), which contracts the pupil in a circular motion, and a set of dilator muscles (dilator pupillae), which pull the iris radially to enlarge the pupil, pulling it in folds.

The circle circumference sphincter constricting muscle is the opposing muscle of the circle-radius dilator muscle. The iris inner smaller circle-circumference changes size when constricting or dilating. The iris outer larger circle-circumference does not change size. The constricting muscle is located on the iris inner smaller circle-circumference.

The back surface is covered by a heavily pigmented epithelial layer that is two cells thick (the iris pigment epithelium), but the front surface has no epithelium. This anterior surface projects as the dilator muscles. The high pigment content blocks light from passing through the iris to the retina, restricting it to the pupil.[1] The outer edge of the iris, known as the root, is attached to the sclera and the anterior ciliary body. The iris and ciliary body together are known as the anterior uvea. Just in front of the root of the iris is the region referred to as the trabecular meshwork, through which the aqueous humour constantly drains out of the eye, with the result that diseases of the iris often have important effects on intraocular pressure and indirectly on vision. The iris along with the anterior ciliary body provide a secondary pathway for aqueous humour to drain from the eye.

The iris is divided into two major regions:

1. The pupillary zone is the inner region whose edge forms the boundary of the pupil.
2. The ciliary zone is the rest of the iris that extends to its origin at the ciliary body.

The collarette is the thickest region of the iris, separating the pupillary portion from the ciliary portion. The collarette is a vestige of the coating of the embryonic pupil.It is typically defined as the region where the sphincter muscle and dilator muscle overlap. Radial ridges extend from the periphery to the pupillary zone, to supply the iris with blood vessels. The root of the iris is the thinnest and most peripheral.

The muscle cells of the iris are smooth muscle in mammals and amphibians, but are striated muscle in reptiles (including birds). Many fish have neither, and, as a result, their irides are unable to dilate and contract, so that the pupil always remains of a fixed size.

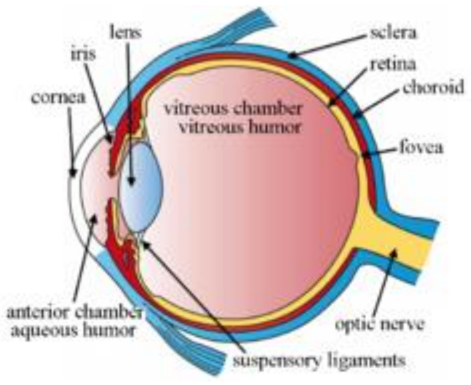


Fig 2.2 Structure of human eye

2.1.1 Front:

The crypts of Fuchs are a series of openings located on either side of the collarette that allow the stroma and deeper iris tissues to be bathed in aqueous humor. Collagen trabeculae that surround the border of the crypts can be seen in blue irises.

The midway between the collarette and the origin of the iris: These folds result from changes in the surface of the iris as it dilates.[citation needed]

Crypts on the base of the iris are additional openings that can be observed close to the outermost part of the ciliary portion of the iris

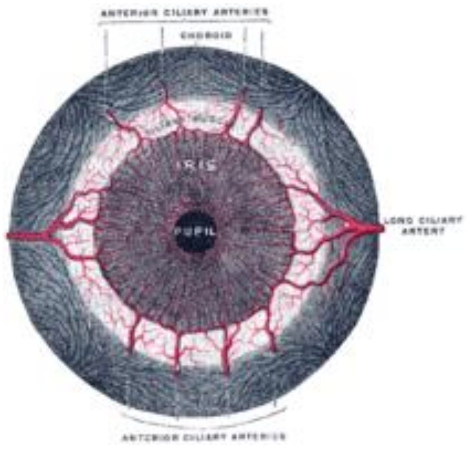


Fig 2.3 Iris, front view

2.1.2 Back:

The radial contraction folds of Schwalbe are a series of very fine radial folds in the pupillary portion of the iris extending from the pupillary margin to the collarette. They are associated with the scalloped appearance of the pupillary ruff.

The structural folds of Schwalbe are radial folds extending from the border of the ciliary and pupillary zones that are much broader and more widely spaced, continuous with the "valleys" between the ciliary processes.

Some of the circular contraction folds are a fine series of ridges that run near the pupillary margin and vary in thickness of the iris pigment epithelium; others are in ciliary portion of iris.

2.1.3 Microanatomy:

From anterior (front) to posterior (back), the layers of the iris are:

1. Anterior limiting layer
2. Stroma of iris
3. Iris sphincter muscle
4. Iris dilator muscle (myoepithelium)
5. Anterior pigment epithelium
6. Posterior pigment epithelium

2.1.4 Development:

The stroma and the anterior border layer of the iris are derived from the neural crest, and behind the stroma of the iris, the sphincter pupillae and dilator pupillae muscles, as well as the iris epithelium, develop from optic cup neuroectoderm.

2.2 Eye Colour:

The iris is usually strongly pigmented, with the colour typically ranging between brown, hazel, green, gray, and blue. Occasionally, the colour of the iris is due to a lack of pigmentation, as in the pinkish white of oculocutaneous albinism, or to obscuration of its pigment by blood vessels, as in the red of an abnormally vascularised iris. Despite the wide range of colours, the only pigment that contributes substantially to normal human iris colour is the dark pigment melanin. The quantity of melanin pigment in the iris is one factor in determining the phenotypic eye colour of a person. Structurally, this huge molecule is only slightly different from its equivalent found in skin and hair. Iris colour is due to variable amounts of eumelanin (brown/black melanins) and pheomelanin (red/yellow melanins) produced by melanocytes. More of the former is found in brown-eyed people and of the latter in blue- and green-eyed people.

2.2.1 Genetic and physical factors determining iris colour:

Iris colour is a highly complex phenomenon consisting of the combined effects of texture, pigmentation, fibrous tissue, and blood vessels within the iris stroma, which together make up an individual's epigenetic constitution in this context. A person's "eye colour" is the colour of one's iris, the cornea being transparent and the white sclera entirely outside the area of interest.

Melanin is yellowish-brown to dark brown in the stromal pigment cells, and black in the iris pigment epithelium, which lies in a thin but very opaque layer across the back of the iris. Most human irises also show a condensation of the brownish stromal melanin in the thin anterior border layer, which by its position has an overt influence on the overall colour.[2] The degree of dispersion of the melanin, which is in subcellular bundles called melanosomes, has some influence on the observed colour, but melanosomes in the iris of humans and other vertebrates are not mobile, and the degree of pigment dispersion cannot be reversed. Abnormal clumping of melanosomes does occur in disease and may lead to irreversible changes in iris colour (see heterochromia, below). Colours other than brown or black are due to selective reflection and absorption from the other stromal components. Sometimes, lipofuscin, a yellow "wear and tear" pigment, also enters into the visible eye colour, especially in aged or diseased green eyes.

The optical mechanisms by which the nonpigmented stromal components influence eye colour are complex, and many erroneous statements exist in the literature. Simple selective absorption and reflection by biological molecules (haemoglobin in the blood vessels, collagen in the vessel and stroma) is the most important element. Rayleigh scattering and Tyndall scattering, (which also happen in the sky) and diffraction also occur. Raman scattering, and constructive interference, as in the feathers of birds, do not contribute to the colour of the human eye, but interference phenomena are important in the brilliantly coloured iris pigment cells (iridophores) in many animals. Interference effects can occur at both molecular and light-microscopic scales and are often associated (in melanin-bearing cells) with quasi crystalline formations, which enhance the optical effects. Interference is recognised by characteristic dependence of colour on the angle of view, as seen in eyespots of some butterfly wings, although the chemical components remain the same. White babies are usually born blue-eyed since no pigment is in the stroma, and their eyes appear blue due to scattering and selective absorption from the posterior epithelium. If melanin is deposited substantially, brown or black colour is seen; if not, they will remain blue or gray.

All the contributing factors towards eye colour and its variation are not fully understood. Autosomal recessive/dominant traits in iris colour are inherent in other species, but colouration can follow a different pattern.

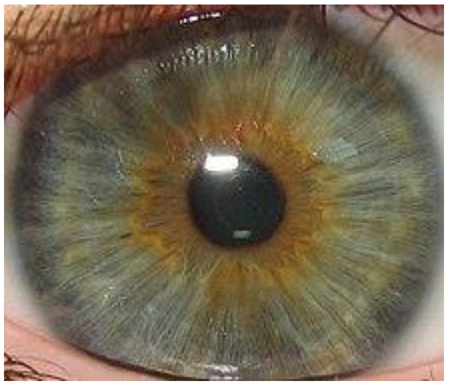


Fig 2.4.1: Example of a gray-green-brown iris

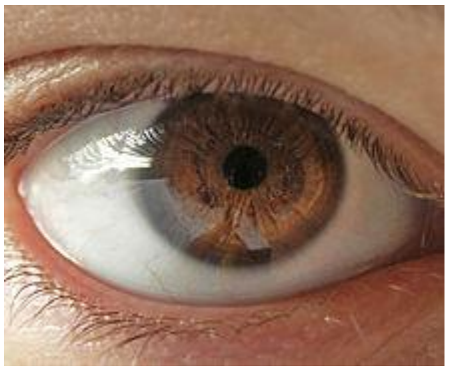


Fig 2.4.2: Example of a brown iris

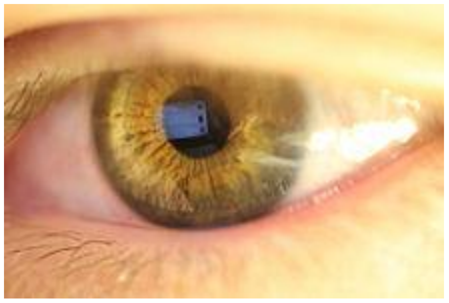


Fig 2.4.3: Example of a green-brown (hazel) iris

2.2.1.1 Amber eyes:

Amber-coloured eyes are extremely rare in humans. They consist of a solid orange/gold colour that may contain lighter shades of the same pigment within the iris. This is an unusual occurrence that happens when the yellow pigment pheomelanin is dominant within the iris. Pheomelanin is also found on individuals with green eyes in much smaller amounts. This is because green eyes have a strong presence of both melanin and pheomelanin. Often in poor lighting, one may mistake amber eyes for brown. This also happens when viewed from far away or in pictures with poor lighting, as well. In natural or well-lit areas, though, telling the difference between the two colours is easy. Another common mistake people make is referring to amber eyes as hazel. Although similar, hazel eyes have a stronger presence of melanin with two very distinct colours within the iris (usually green/brown), and often contain many speckles or blotches of mixed hues.

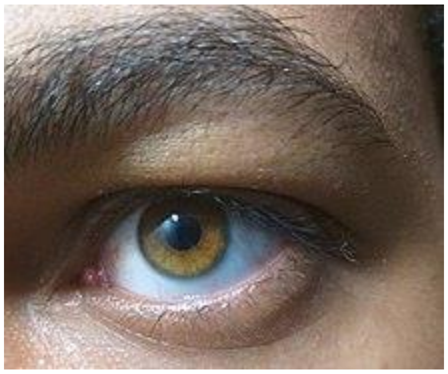


Fig 2.5: Adult male with amber-coloured eyes

2.2.2 Different colours in the two eyes:

Heterochromia (also known as a heterochromia iridis or heterochromia iridum) is an ocular condition in which one iris is a different colour from the other iris (complete heterochromia), or where the part of one iris is a different colour from the remainder (partial heterochromia or sectoral heterochromia). Uncommon in humans, it is often an indicator of ocular disease, such as chronic iritis or diffuse iris melanoma, but may also occur as a normal variant. Sectors or patches of strikingly different colours in the same iris are less common. Anastasius the First was dubbed dikoros (having two irises) for his patent heterochromia since his right iris had a darker colour than the left one.

In contrast, heterochromia and variegated iris patterns are common in veterinary practice. Siberian Husky dogs show heterochromia,[8][better source needed] possibly analogous to the genetically determined Waardenburg syndrome of humans. Some white cat fancies (e.g., white Turkish Angora or white Turkish van cats) may show striking heterochromia, with the most common pattern being one uniformly blue, the other copper, orange, yellow, or green.[8] Striking variation within the same iris is also common in some animals, and is the norm in some species. Several herding breeds, particularly those with a blue merle coat colour (such as Australian Shepherds and Border Collies) may show well-defined blue areas within a brown iris, as well as separate blue and darker eyes.[citation needed] Some horses (usually within the white, spotted, palomino, or cremello groups of breeds) may show amber, brown, white and blue all within the same eye, without any sign of eye disease

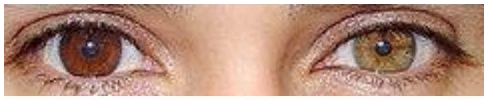


Fig 2.6: Example of heterochromia - one eye of the subject is brown, the other hazel.

CHAPTER 3

IMAGE ACQUISITION

An image can be defined as a 2-D function f(x,y) where (x, y) is co-ordinate in two dimensional space and f is the intensity of that co-ordinate[3]. Each co-ordinate position is called as pixel. Pixel is the smallest unit of the image it is also called as picture element or pel. So digital images are composed of pixels, each pixel represents the colour (gray level for black and white images) at a single point in the image. Pixel is like tiny dot of particular colour. A digital image is a rectangular array of pixels also called as Bitmap. From the point of view of photography the digital images are of two types [4][5]

- Black and white Images.
- Colour Images.

1. Black and White Images:

Black and white images are made of different shades of gray. These different shades lies between 0 to 255, where 0 refers to black, 255 refers to white and intermediate values refer to different shades of black and white. Grayscale refers to the range of neutral tonal values (shades) from black to white.

2. Colour Image :

Colour images are made up of coloured pixels. Colour can capture a much broader range of values than grayscale. “The spectrum – the band of colours produced when sunlight passes through a prism – includes billions of colours, of which the human eye can perceive seven to ten million”. The electronic capture and display of colour is complicated. RGB (Red, Green, and Blue) is the most commonly adopted colour system.

Example: A one-bit image can assign only one of two values to a single pixel: 1 or 0 (black or white). An 8-bit (28) grayscale image can assign one of 256 colours to a single pixel. A 24-bit (2(3x8)) RGB image (8-bits each for red, green and blue colour channels) can assign one of 16.8 million colours to a single pixel.

3.1 IMAGE ACQUISITION:

The general aim of Image Acquisition is to transform an optical image (Real World Data) into an array of numerical data which could be later manipulated on a computer, before any video or image processing can commence an image must be captured by camera and converted into a manageable entity. The Image Acquisition process consists of three steps: -
1. Optical system which focuses the energy
2. Energy reflected from the object of interest
3. A sensor which measure the amount of energy. Image Acquisition is achieved by suitable camera. We use different cameras for different application. If we need an x-ray image, we use a camera (film) that is sensitive to x-ray. If we want infra-red image, we use camera which are

sensitive to infrared radiation. For normal images (family pictures etc) we use cameras which are sensitive to visual spectrum. Image Acquisition is the first step in any image processing system.

Table 3.1: Shades/Colours Depends on the Bits Required to Represent the Digital Image

Bits/Digital Image type	**Shades/colours**
8 bits black and white image	256 shades
24 bits coloured	16.8 million colour
10 bits black and white image	1024 shades
30 bits coloured	1 billion colour
12 bits black and white image	4096 shades

3.1.1. IMAGE ACQUISITION CONCEPT:

In order to capture an image a camera requires some sort of measurable energy. The energy of interest in this context is light or more generally electromagnetic waves. An EM waves can be described as mass less entity, a photon, whose electric and magnetic field varies sinusoidaly, hence the name waves. A photon can be described in three different ways: -

1. A photon can be described By its energy E (measured in eV)
2. A photon can be described by its frequency f (H_2)
3. A photon can be described by its wavelength λ(m)

$$E = (hc)/ \lambda$$

$$E = hf$$

3.1.2 QUANTUM DETECTORS:

Quantum Detector is the most important mechanism of image sensing and acquisition it relies upon the energy of absorbed photon being used to promote electrons from their stable state to a higher state above an energy threshold. Whenever this occurs, the properties of that material get altered in some measurable way.

Planck/Einstein came up with a relationship between λ of the incident photon and the E that it carries:-

$$E = (hc)/\lambda$$

On collision the photon transfer all or none of this quantum of energy to the electron.

3.1.3 IMAGE ACQUISITION MODEL:

The images are generated by combination of an illumination source and the reflection or absorption of the energy by the elements of scene being imaged. Illumination may be originated by radar, infrared energy source, computer generated energy pattern, ultrasound energy source, X-ray energy source etc.

To sense the image, we use sensor according to the nature of illumination. The process of image sense is called image acquisition.

By the sensor, basically illumination energy is transformed into digital image. The idea is that incoming illumination energy is transformed into voltage by the combination of input electrical energy and sensor material that is responsive to the particular energy that is being detected. The output waveform is response of sensor and this response is digitalized to obtain digital image.

Image is represented by 2-D function f(x, y). Practically an image must be non-zero and finite quantity that is [1]:

$$0< f(x, y) <“\square \qquad (2)$$

- It is also discussed that for an image f (x, y), we have two factors:
- The amount of source illumination incident on the scene being imaged. Let us represent it by :

$$i(x, y)$$

The amount of illumination reflected or absorbed by the object in the scene. Let us represent it by:

$$r(x, y)$$

Then f(x, y) can be represented by : $f(x, y) = i(x, y).r(x, y) \qquad (3)$

Where $0< i(x, y)<“$

It means illumination will be a non-zero and finite quantity and its quantity depends on illumination source.

and $0< r(x, y)<1$

Here 0 indicates no reflection or total absorption and 1 means no absorption or total reflection.

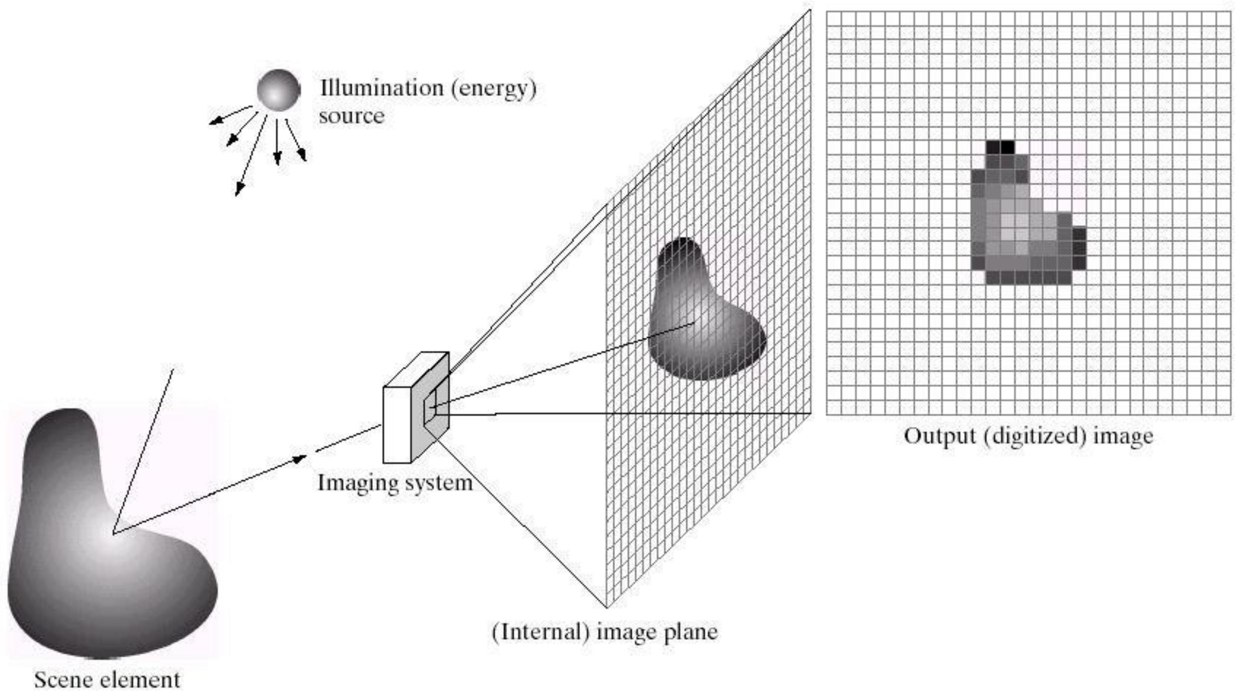


Fig 3.1: Image Acquisition Model

3.1.4 TECHNIQUES TO PERFORM IMAGE ACQUISITION:

Image Acquisition process totally depends on the hardware system which may have a sensor that is again a hardware device. A sensor converts light into electrical charges. The sensor inside a camera measures the reflected energy by the scene being imaged. The image sensor employed by most digital cameras is a charge coupled device (CCD) [5]. Some cameras use complementary metal oxide semiconductor (CMOS) technology instead [5].

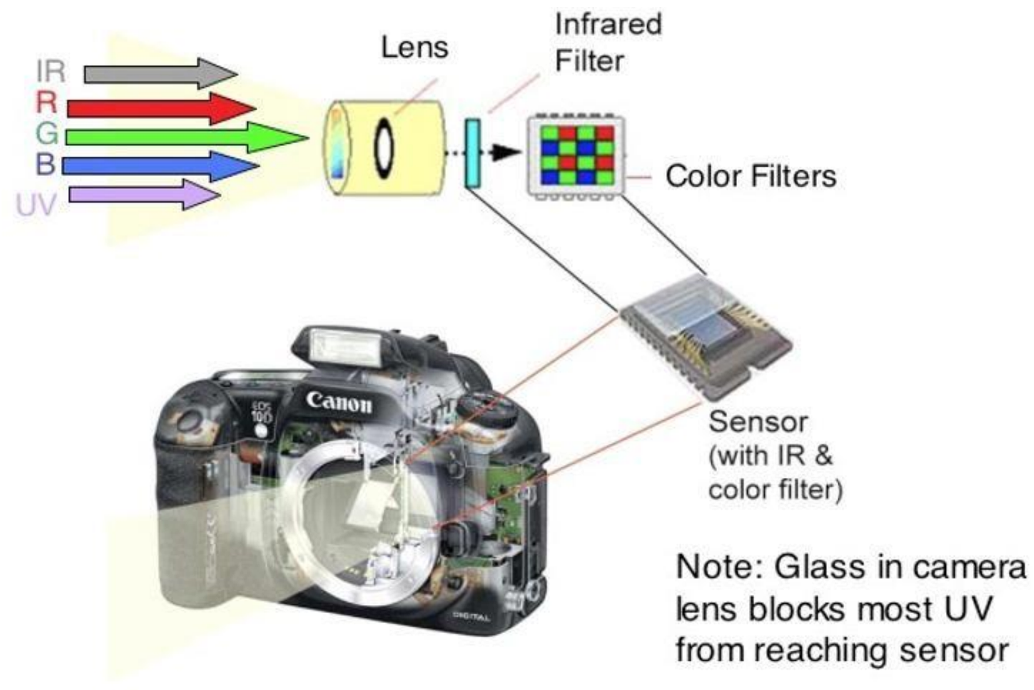

IR
R
G
B
UV
Lens
Infrared Filter
Color Filters
Sensor (with IR & color filter)
Note: Glass in camera lens blocks most UV from reaching sensor

Fig.3.2: Inside a Digital Camera

Both CCD and CMOS image sensors convert light into electrons. A simplified way to think about these sensors is to think of a 2-D array of thousands or millions of tiny solar cells. (in this case the sensors are called photosites). Once the sensor converts the light into electrons, it reads the value (accumulated charge) of each cell in the image. A CCD transports the charge across the chip and reads it at one corner of the array. An analog-to- digital converter (ADC) then turns each pixel's value into a digital value by measuring the amount of charge at each photosite and converting that measurement to binary form. CMOS devices use several transistors at each pixel to amplify and move the charge using more traditional wires. CCD sensors create high-quality, low-noise images. CMOS sensors are generally more susceptible to noise.

CMOS sensors traditionally consume little power. CCDs, on the other hand, use a process that consumes lots of power. CCDs consume as much as 100 times more power than an equivalent CMOS sensor. CCD sensors have been mass produced for a longer period of time, so they are more mature. They tend to have higher quality pixels, and more of them.

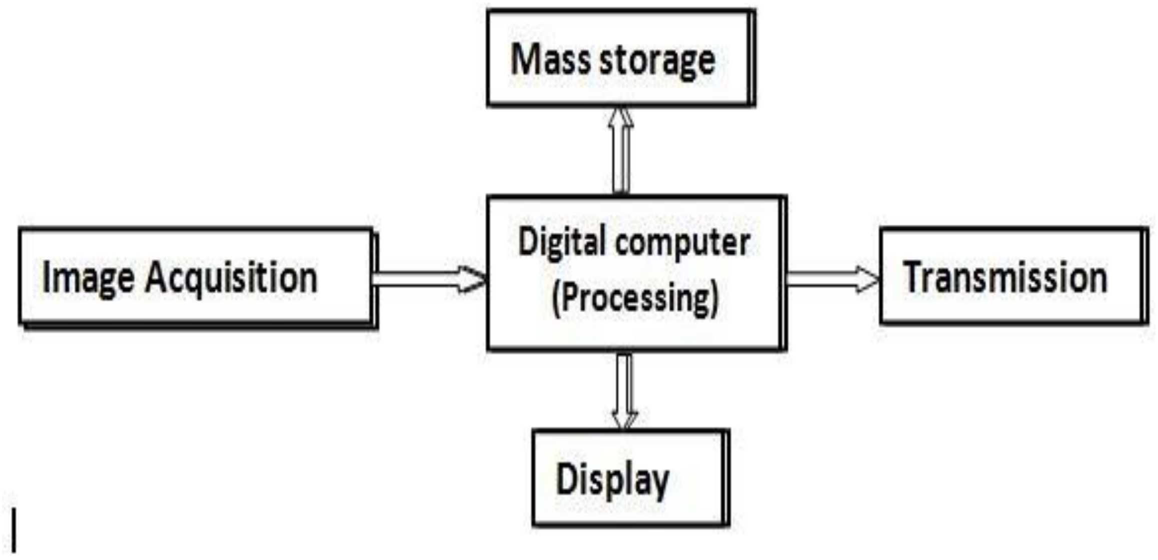


Fig 3.3: After image acquisition it is sent for processing, transmission, display or storage

CHAPTER 4

PIXEL EXTRACTION

Pixel

In digital imaging, a pixel, pel or picture element is a smallest addressable element in a raster image, or the smallest addressable element in an all points addressable display device; so it is the smallest controllable element of a picture represented on the screen.

Each pixel is a sample of an original image; more samples typically provide more accurate representations of the original. The intensity of each pixel is variable. In colour imaging systems, a colour is typically represented by three or four component intensities such as red, green, and blue, or cyan, magenta, yellow, and black.

In some contexts (such as descriptions of camera sensors), pixel refers to a single scalar element of a multi-component representation (called a photosite in the camera sensor context, although sensel is sometimes used), while in yet other contexts it may refer to the set of component intensities for a spatial position.

4.1 Pixel Values:

Each of the pixels that represents an image stored inside a computer has a *pixel value* which describes how bright that pixel is, and/or what colour it should be. In the simplest case of binary images, the pixel value is a 1-bit number indicating either foreground or background. For a grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common *pixel format* is the *byte image*, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black, and 255 is taken to be white. Values in between make up the different shades of gray.

To represent colour images, separate red, green and blue components must be specified for each pixel (assuming an RGB colourspace), and so the pixel `value' is actually a vector of three numbers. Often the three different components are stored as three separate `grayscale' images known as *colour planes* (one for each of red, green and blue), which have to be recombined when displaying or processing.

Multi-spectral images can contain even more than three components for each pixel, and by extension these are stored in the same kind of way, as a vector pixel value, or as separate colour planes.

The actual grayscale or colour component intensities for each pixel may not actually be stored explicitly. Often, all that is stored for each pixel is an index into a colourmap in which the actual intensity or colours can be looked up.

Although simple 8-bit integers or vectors of 8-bit integers are the most common sorts of pixel values used, some image formats support different types of value, for instance 32-bit signed integers or floating point values. Such values are extremely useful in image processing as they allow processing to be carried out on the image where the resulting pixel values are not necessarily 8-bit integers. If this approach is used then it is usually necessary to set up a colourmap which relates particular ranges of pixel values to particular displayed colours.

4.2 OpenCV Getting and Setting Pixels:

4.2.1 What are pixels?

Pixels are the raw building blocks of an image. Every image consists of a set of pixels. There is no finer granularity than the pixel.
Normally, a pixel is considered the “color” or the “intensity” of light that appears in a given place in our image.
If we think of an image as a grid, each square in the grid contains a single pixel. Let’s look at the example image in figure 4.1

Fig 4.1: This image is 600 pixels wide and 450 pixels tall for a total of *600 x 450 = 270,000* pixels.

Most pixels are represented in two ways:

1. Grayscale/single channel
2. Color

In a grayscale image, each pixel has a value between 0 and 255, where 0 corresponds to “black” and 255 being “white.” The values between 0 and 255 are varying shades of gray, where values closer to 0 are darker and values closer 255 are lighter.

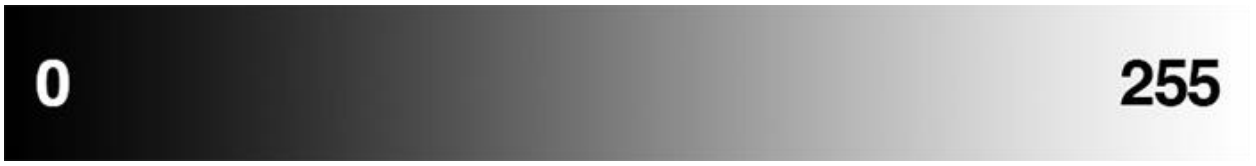


Fig 4.2 : Image gradient demonstrating pixel values going from black (0) to white (255)

The grayscale gradient image in Figure 4.2 demonstrates *darker pixels* on the left-hand side and progressively *lighter pixels* on the right-hand side.

Color pixels, however, are normally represented in the RGB color space — one value for the Red component, one for Green, and one for Blue leading to a total of *3 values per pixel:*

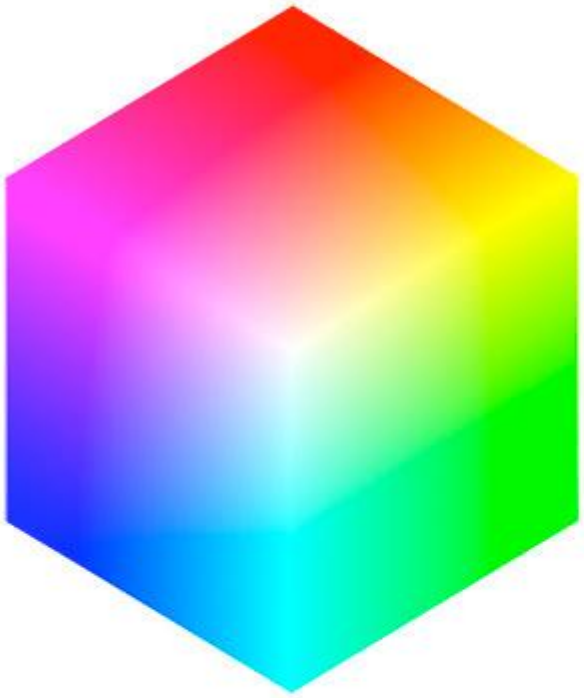


Fig 4.3: The RGB cube.

Other color spaces exist (HSV (Hue, Saturation, Value), L*a*b*, etc.).

Each of the three Red, Green, and Blue colors are represented by an integer in the range from 0 to 255, which indicates how “much” of the color there is. Given that the pixel value only needs to be in the range *[0, 255]*, we normally use an 8-bit unsigned integer to represent each color intensity.
We then combine these values into an RGB tuple in the form (red, green, blue).This tuple represents our color.

To construct a white color, we would completely fill each of the red, green, and blue buckets, like this: (255, 255, 255) — since white is the presence of all colors.

Then, to create a black color, we would completely empty each of the buckets: (0, 0, 0)— since black is the absence of color.

To create a pure red color, we would completely fill the red bucket (and only the red bucket): (255, 0, 0).Look at the figure4.4 to make this concept more clear.

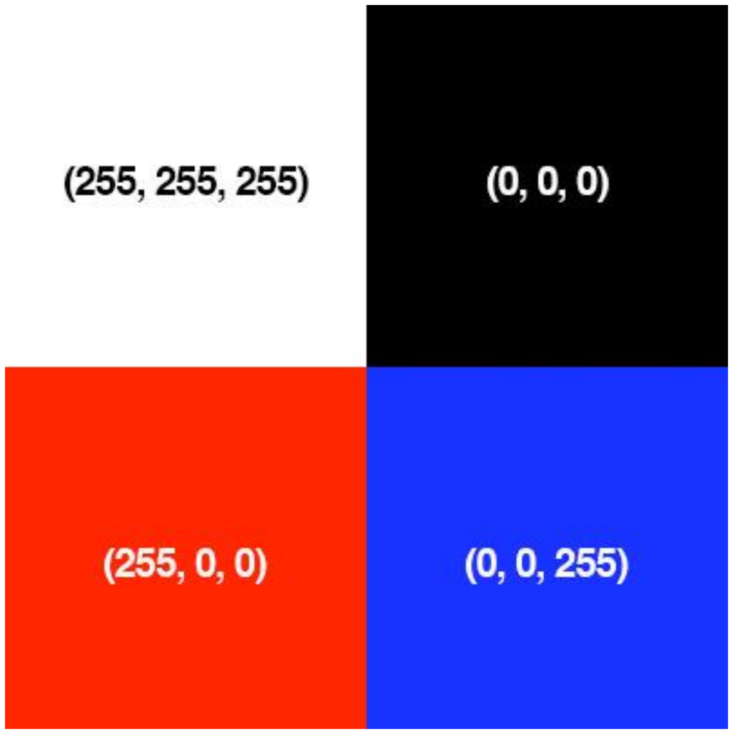


Fig 4.4: Here, we have four examples of colors and the “bucket” amounts for each of the Red, Green, and Blue components, respectively.

In the *top-left* example, we have the color *white* — each of the Red, Green, and Blue buckets have been completely filled to form the white color.
And on the *top-right,* we have the color black — the Red, Green, and Blue buckets are now totally empty.
Similarly, to form the color red in the *bottom-left,* we simply fill the Red bucket completely, leaving the other Green and Blue buckets totally empty.
Finally, blue is formed by filling only the Blue bucket, as demonstrated in the *bottom-right.*
For your reference, here are some common colors represented as RGB tuples:

1. **Black:**

(0, 0, 0)

2. **White:**

(255, 255, 255)

3. **Red:**

(255, 0, 0)

4. **Green:**
(0, 255, 0)
5. **Blue:**
(0, 0, 255)
6. **Aqua:**
(0, 255, 255)
7. **Fuchsia:**
(255, 0, 255)
8. **Maroon:**
(128, 0, 0)
9. **Navy:**
(0, 0, 128)
10. **Olive:**
(128, 128, 0)
11. **Purple:**
(128, 0, 128)
12. **Teal:**
(0, 128, 128)
13. **Yellow:**
(255, 255, 0)

4.2.2 Overview of the image coordinate system in OpenCV:

As mentioned in Figure 4.1, an image is represented as a grid of pixels. Imagine our grid as a piece of graph paper. Using this graph paper, the point *(0, 0)* corresponds to the *top-left* corner of the image (i.e., the *origin*). As we move *down* and to the *right*, both the *x* and *y*-values increase. Let's look at the image in Figure 4.5 to make this point more clear:

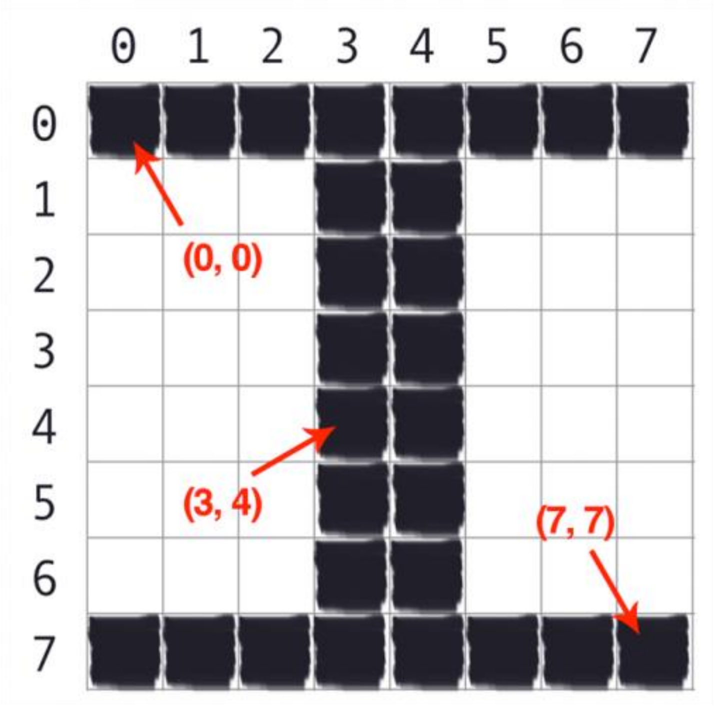


Fig 4.5: In OpenCV, pixels are accessed by their *(x, y)*-coordinates. The origin, *(0, 0)*, is located at the *top-left* of the image. OpenCV images are zero-indexed, where the *x*-values go *left-to-right* (column number) and *y*-values go *top-to-bottom* (row number).

Here, we have the letter "I" on a piece of graph paper. We see that we have an *8 x 8* grid with 64 total pixels.
The point at *(0, 0)* corresponds to the *top-left* pixel in our image, whereas the point *(7, 7)* corresponds to the *bottom-right* corner.
It is important to note that we are counting from *zero* rather than *one*. The Python language is zero-indexed, meaning that we always start counting from zero.
Finally, the pixel 4 columns to the *right* and 5 rows *down* is indexed by the point *(3, 4)*, keeping in mind that we are counting from zero rather than one.

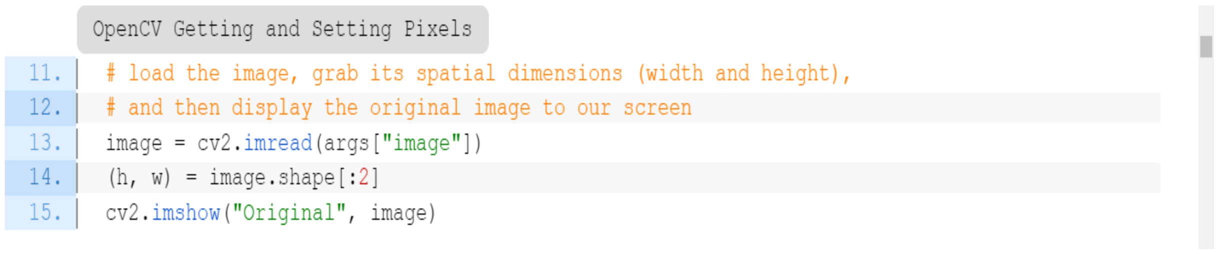
4.2.3 Configuring your development environment:

OpenCV Getting and Setting Pixels:

```
$ pip install opencv-contrib-python
```

4.2.3.1 Project structure:

Before we start looking at code, let's review our project directory structure:
OpenCV Getting and Setting Pixels



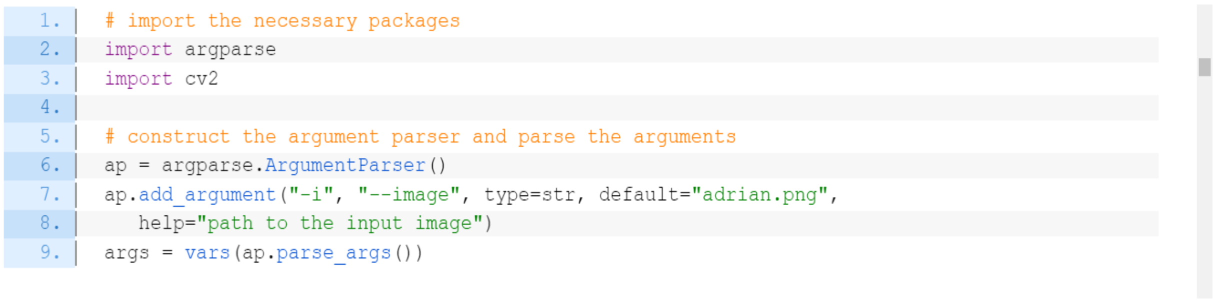
OpenCV Getting and Setting Pixels

```
# load the image, grab its spatial dimensions (width and height),
# and then display the original image to our screen
image = cv2.imread(args["image"])
(h, w) = image.shape[:2]
cv2.imshow("Original", image)
```

opencv_getting_setting.py, which will allow us to access and manipulate the image pixels from the image
adrian.png
.

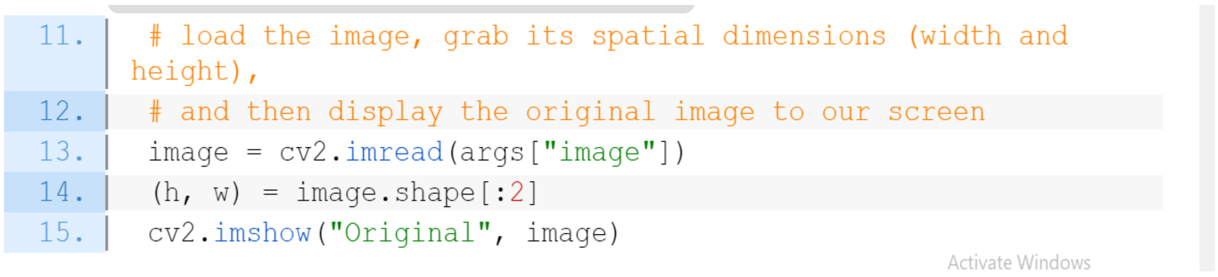
4.2.3.2 Getting and setting pixels with OpenCV :

Let us learn how to get and set pixels with OpenCV. Open the opencv_getting_setting.py file in your project directory structure.



```
1.  # import the necessary packages
2.  import argparse
3.  import cv2
4.
5.  # construct the argument parser and parse the arguments
6.  ap = argparse.ArgumentParser()
7.  ap.add_argument("-i", "--image", type=str, default="adrian.png",
8.     help="path to the input image")
9.  args = vars(ap.parse_args())
```

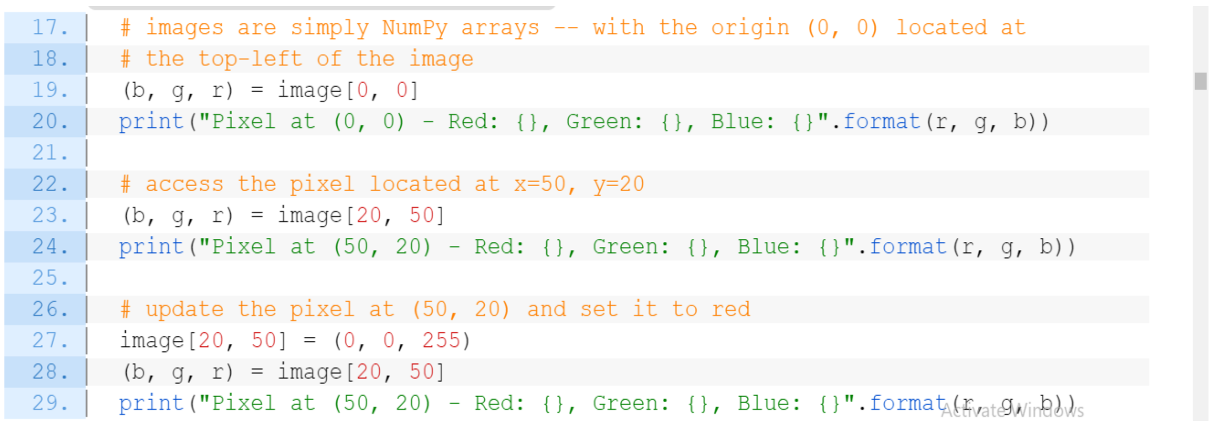
Lines 2 and 3 import our required Python packages. We only need argparse for our command line arguments cv2 for our OpenCV bindings.
The –image command line argument points to the image we want to manipulate residing on disk. By default, the –image command line argument is set to adrian.png
.
Next, let us load this image and start accessing pixel values:



```
11.  # load the image, grab its spatial dimensions (width and
height),
12.  # and then display the original image to our screen
13.  image = cv2.imread(args["image"])
14.  (h, w) = image.shape[:2]
15.  cv2.imshow("Original", image)
```

Lines 13-15 load our input image

Images in OpenCV are represented by NumPy arrays. To access a particular image pixel, all we need to do is pass in the *(x, y)*-coordinates as image[y, x]



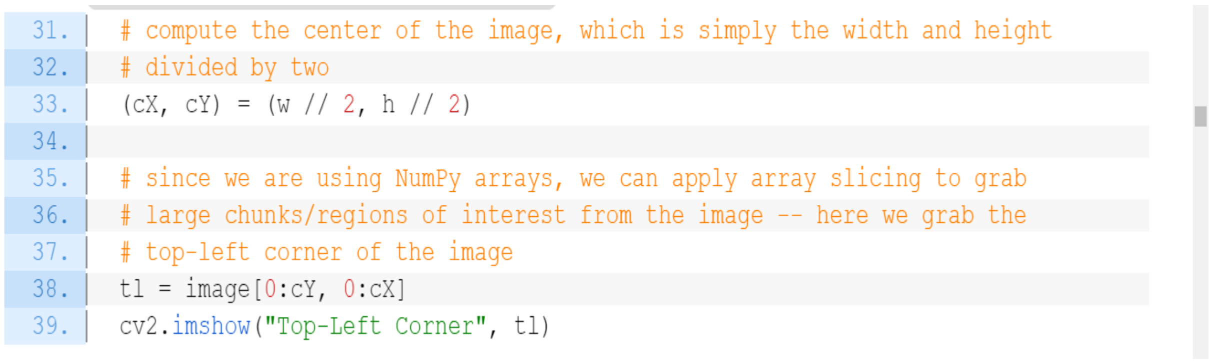
```
17.  # images are simply NumPy arrays -- with the origin (0, 0) located at
18.  # the top-left of the image
19.  (b, g, r) = image[0, 0]
20.  print("Pixel at (0, 0) - Red: {}, Green: {}, Blue: {}".format(r, g, b))
21.
22.  # access the pixel located at x=50, y=20
23.  (b, g, r) = image[20, 50]
24.  print("Pixel at (50, 20) - Red: {}, Green: {}, Blue: {}".format(r, g, b))
25.
26.  # update the pixel at (50, 20) and set it to red
27.  image[20, 50] = (0, 0, 255)
28.  (b, g, r) = image[20, 50]
29.  print("Pixel at (50, 20) - Red: {}, Green: {}, Blue: {}".format(r, g, b))
```

Line 19 accesses the pixel located at *(0, 0)*, which is the *top-left* corner of the image. In return, we receive the blue, green, and red intensities (BGR), in that order.

The big question here is, *Why does OpenCV represent images in BGR channel ordering rather than the standard RGB?*
The answer is that back when OpenCV was originally developed, BGR ordering *was* the standard! It was only later that the RGB order was adopted. The BGR ordering is standard in OpenCV, so get used to seeing it.
Line 23 then accesses the pixel located at *x = 50*, *y = 20* using the array indexing of image[20, 50]
.
But wait . . . isn't that backward? Shouldn't it instead be image [50, 20] since *x = 50* and *y = 20*?
Let's back up a step and consider that an image is simply a matrix with a width (number of columns) and height (number of rows). If we were to access an individual location in that matrix, we would denote it as the x-value (column number) and y-value (row number).

Therefore, to access the pixel located at *x = 50*, *y = 20*, you pass the *y*-value first (the row number) followed by the *x*-value (the column number), resulting in
image[y, x]
.
***Note:** we have found that the concept of accessing individual pixels with the syntax of image[y, x] is the correct syntax based on the fact that the x-value is your column number (i.e., width), and the y-value is your row number (i.e., height).*

Lines 27 and 28 update the pixel located at *x = 50*, *y = 20*, setting it to red, which is (0, 0, 255) in BGR ordering. Line 29 then prints the updated pixel value to our terminal, thereby demonstrating that it has been updated.
Next, let us learn how to use NumPy array slicing to grab large chunks/regions of interest from an image:



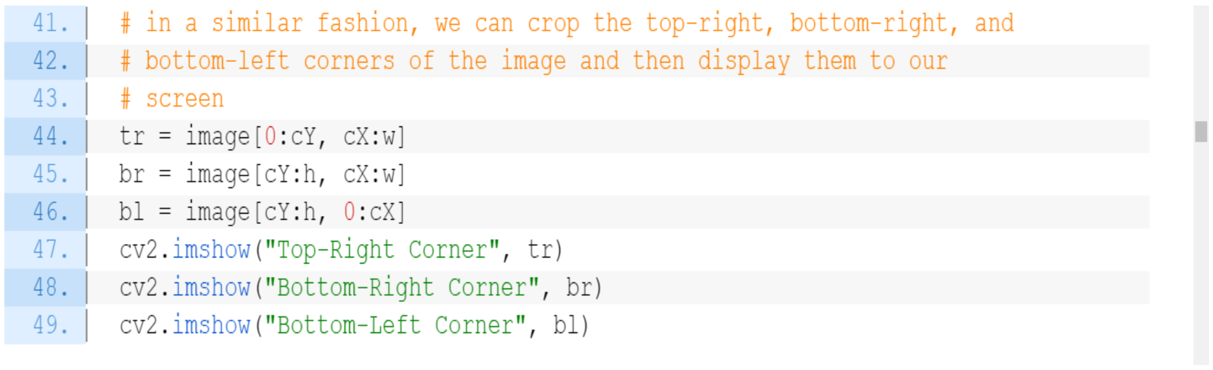
```
31. # compute the center of the image, which is simply the width and height
32. # divided by two
33. (cX, cY) = (w // 2, h // 2)
34.
35. # since we are using NumPy arrays, we can apply array slicing to grab
36. # large chunks/regions of interest from the image -- here we grab the
37. # top-left corner of the image
38. tl = image[0:cY, 0:cX]
39. cv2.imshow("Top-Left Corner", tl)
```

On Line 33, we compute the center *(x, y)*-coordinates of the image. This is accomplished by simply dividing the width and height by two, ensuring integer conversion (since we cannot access "fractional pixel" locations).

Then, on Line 38, we use simple NumPy array slicing to extract the [0, cX) and [0, cY) region of the image. In fact, this region corresponds to the *top-left* corner of the image! To grab chunks of an image, NumPy expects we provide four indexes:

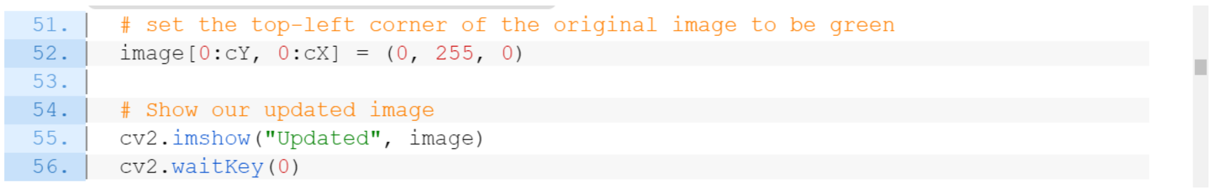
- Start *y*: The first value is the starting *y*-coordinate. This is where our array slice will start along the *y*-axis. In our example above, our slice starts at *y = 0*.
- End *y*: Just as we supplied a starting *y*-value, we must provide an ending *y*-value. Our slice stops along the *y*-axis when *y = cY*.
- Start *x*: The third value we must supply is the starting *x*-coordinate for the slice. To grab the *top-left* region of the image, we start at *x = 0*.
- End *x*: Lastly, we need to provide the *x*-axis value for our slice to stop. We stop when *x = cX*.

Once we have extracted the *top-left* corner of the image, Line 39 shows the cropping result. Notice how our image is just the *top-left* corner of our original image:



```
41. # in a similar fashion, we can crop the top-right, bottom-right, and
42. # bottom-left corners of the image and then display them to our
43. # screen
44. tr = image[0:cY, cX:w]
45. br = image[cY:h, cX:w]
46. bl = image[cY:h, 0:cX]
47. cv2.imshow("Top-Right Corner", tr)
48. cv2.imshow("Bottom-Right Corner", br)
49. cv2.imshow("Bottom-Left Corner", bl)
```

Let's extend this example a little further so we can get some practice using NumPy array slicing to extract regions from images.In a similar fashion to the example above, Line 44 extracts the *top-right* corner of the image, Line 45 extracts the *bottom-right* corner, and Line 46 the *bottom-left*.
Finally, all four corners of the image are displayed on screen on Lines 47-49, like this:

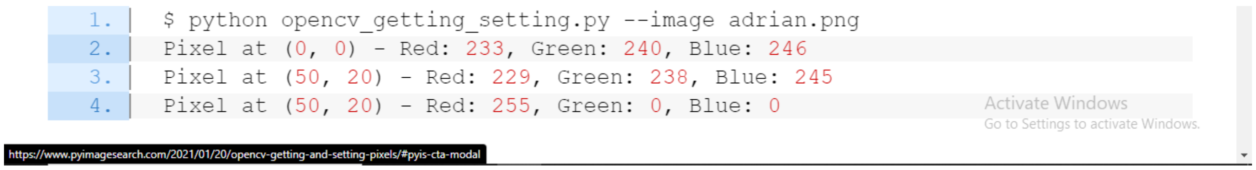


```
51. # set the top-left corner of the original image to be green
52. image[0:cY, 0:cX] = (0, 255, 0)
53.
54. # Show our updated image
55. cv2.imshow("Updated", image)
56. cv2.waitKey(0)
```

On Line 52, you can see that we are again accessing the *top-left* corner of the image; however, this time, we are setting this region to have a value of (0, 255, 0) (green). Lines 55 and 56 then show the results of our work:

4.2.3.3 OpenCV pixel getting and setting results:

The individual pixel can be set using the following command.



```
$ python opencv_getting_setting.py --image adrian.png
Pixel at (0, 0) - Red: 233, Green: 240, Blue: 246
Pixel at (50, 20) - Red: 229, Green: 238, Blue: 245
Pixel at (50, 20) - Red: 255, Green: 0, Blue: 0
```

https://www.pyimagesearch.com/2021/01/20/opencv-getting-and-setting-pixels/#pyis-cta-modal

Once our script starts running, you should see some output printed to your console.

The first line of output tells us that the pixel located at *(0, 0)* has a value of *R = 233*, *G = 240*, and *B = 246*. The buckets for all three channels are nearly white, indicating that the pixel is very bright.
The next two lines of output show that we have successfully changed the pixel located at *(50, 20)* to be red rather than the (nearly) white color.

4.3 System Requirements

4.3.1 Major Software's and Libraries Used

Google Collab

Google Collab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Collab supports many popular machine learning libraries which can be easily loaded in your notebook.

As a programmer, we can perform the following using Google Collab:

- Write and execute code in Python
- Document your code that supports mathematical equations
- Import/Save notebooks from/to Google Drive
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

Python: The programming style of Python is simple, clear and it also contains powerful different kinds of classes. Moreover, Python can easily combine other programming languages, such as C or C++. As a successful programming language, it has its own advantages:

- Simple and easy to learn
- Open source
- Scalability

OpenCV: OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real- time computer vision. The library is cross-platform and free for use under the open-source BSD license. OpenCV supports the deep leaning framework TensorFlow, Torch/PyTorch and caffe.

NumPy: In Python, there is data type called array. To implement the data type of array with python, NumPy is the essential library for analysing and calculating data. They are all open source libraries. NumPy is mainly used 22 for the matrix calculation

Pandas , Matplotlib: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool,built on top of the Python programming language. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Pillow: Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

The **Python Imaging Library** adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

CHAPTER 5

COLOUR DETECTION

Colour, one of the most influential attributes of light, has a distinctive importance in various industries and scientific applications. The colour of a material can be used to evaluate the properties of that material. Colour sensors are employed to recognise/detect the colour of a material in RGB (red, green, blue) scale, while rejecting the unwanted infrared or ultraviolet light. The ultimate challenge with colour sensing has been to detect subtle differences among similar or highly reflective surfaces. Fortunately, the advances in electronics, optics and software technology have led to the development of colour recognition techniques that involve outputting the reading intensity and colour value. These colour recognition systems are deemed highly suitable for quality control applications in various industries, such as food, automotive, glass, manufacturing, for ensuring high productivity and cost reduction.

A typical colour sensor comprises a high-intensity white LED (light-emitting diode) that projects a modulated light onto the target. The white light holds a mixture of three basic colours having different wavelengths as mentioned above. These colours combine with one another to form different other shades of colours. When the white light falls on any surface, based on the properties of the surface material, some of the wavelengths of light are absorbed and some are reflected. A human eye detects the colour of the material when these reflected wavelengths fall on it.

Colour recognition systems based on white LED allow for a greater spectrum evaluation than those based on RGB LED. Apart from LEDs, there are various integral light sources such as fibre optics, lasers, and halogen lamps that can be used in the design of colour sensors.

5.1 What is Colour Detection?

Colour detection is the process of detecting the name of any colour. Simple, isn’t it? Well, for humans this is an extremely easy task but for computers, it is not straightforward. Human eyes and brains work together to translate light into colour. Light receptors that are present in our eyes transmit the signal to the brain. Our brain then recognizes the colour.

5.2 How Colour Sensors Work?

Colour sensors are developed based on diffuse technology that can detect a wide range of colours. The combination of colour sensitive filters and sensors array perform colour sensing, which is further used to analyse the colour present in an image or in a specified object. The colour measurement process involves a light source to illuminate the surface, the target surface, and a receiver that measures the reflected wavelengths. A white light emitter is used to illuminate the surface. The sensor then activates three filters with three wavelength sensitivities to measure the wavelengths of RGB colours respectively. Based on these three colours, the colour of the material is determined.

Modern colour sensing has seen the involvement of fibre optics in the colour detection process. In this technology, the light transmission to the object and back depends on the optical glass fibres, which operate on the principle of total internal reflection. This phenomenon causes fibre to act as a waveguide and enables the complete reflection of light. In a fibre optic colour sensor, the white light spot is projected via the fibre optic onto the target surface; the part of light that is reflected back from the target is directed onto the detector via the same optical fibre. The reflected light is then separated into long, medium and short-wave light components

White light is a mixture of three basic colours known as primary colours. They are red, blue and green. These colours have different wavelengths. Combinations of these colours at different proportions create different types of colours. When the white light falls on any surface, some of the wavelengths of the light are absorbed by the surface while some are reflected based on the properties of the surface material. Colour of the material is detected when these reflected wavelengths fall on the human eye. A material reflecting wavelengths of red light appears as red. The component used to detect colours is the Colour sensor.
colours are made up of 3 primary colours; red, green, and blue. In computers, we define each colour value within a range of 0 to 255. So, in how many ways we can define a colour? The answer is 256*256*256 = 16,581,375. There are approximately 16.5 million different ways to represent a colour. In our dataset, we need to map each colour's values with their corresponding names. But don't worry, we don't need to map all the values. We will be using a dataset that contains RGB values with their corresponding names.

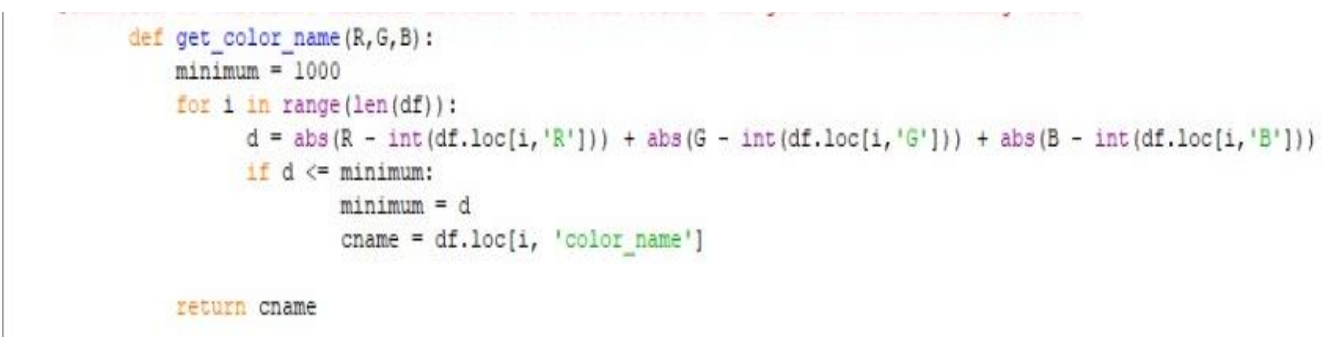
Finding the Euclidean distance from the detected pixel to the colours present in the data sheet. The Euclidean Distance between two points in either the plane or 3-dimensional space measures the length of a segment connecting the two points. It is the most obvious way of representing distance between two points. The Euclidean distance between two points in either the plane or 3-dimensional space measures the length of a segment connecting the two points. It is the most obvious way of representing distance between two points.

The Pythagorean Theorem can be used to calculate the distance between two points, as shown in the figure below. If the points (x1, y1) and (x2,y2) are in 2-dimensional space, then the Euclidean distance between them is given in Eq.1 $\sqrt{((\mathrm{x2}-\mathrm{x1})^2 + (\mathrm{y2}-\mathrm{y1})^2))}$ (1)

Now after calculating the distance, the pixel with the shortest distance from the colours in data sheet is considered as the match colour and then displayed as output. Table 1. shows few of the 1500 different colours it can identify using this method.

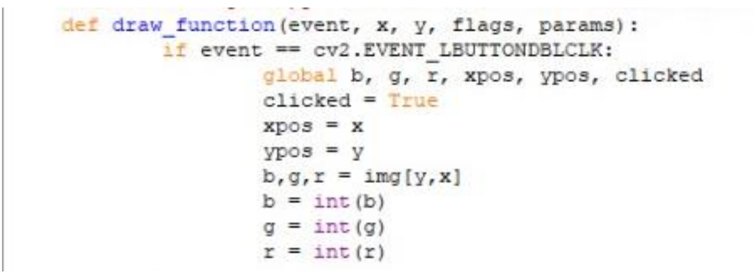
Further the colour of the Iris is found through the code below:

1. Function to calculate minimum distance from all the colours and get the minimum distance:



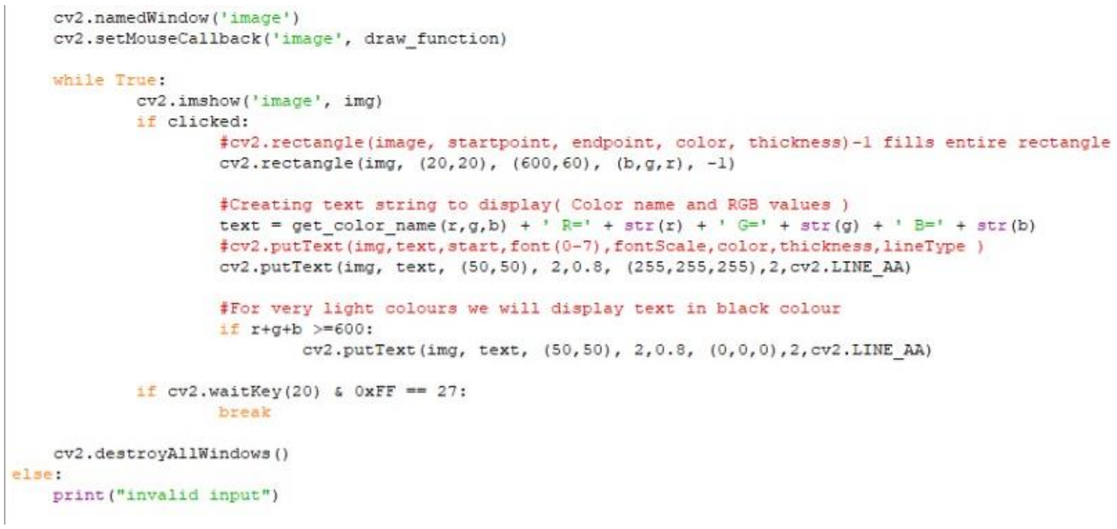
```
def get_color_name(R,G,B):
    minimum = 1000
    for i in range(len(df)):
        d = abs(R - int(df.loc[i,'R'])) + abs(G - int(df.loc[i,'G'])) + abs(B - int(df.loc[i,'B']))
        if d <= minimum:
            minimum = d
            cname = df.loc[i, 'color_name']

    return cname
```

2. Get the x,y Coordinates:



```
def draw_function(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDBLCLK:
        global b, g, r, xpos, ypos, clicked
        clicked = True
        xpos = x
        ypos = y
        b,g,r = img[y,x]
        b = int(b)
        g = int(g)
        r = int(r)
```

3. open the window and find the colour of the Iris of the person as sjown in fig 5.1 and fig 5.2:



```
    cv2.namedWindow('image')
    cv2.setMouseCallback('image', draw_function)

    while True:
        cv2.imshow('image', img)
        if clicked:
            #cv2.rectangle(image, startpoint, endpoint, color, thickness)-1 fills entire rectangle
            cv2.rectangle(img, (20,20), (600,60), (b,g,r), -1)

            #Creating text string to display( Color name and RGB values )
            text = get_color_name(r,g,b) + ' R=' + str(r) + ' G=' + str(g) + ' B=' + str(b)
            #cv2.putText(img,text,start,font(0-7),fontScale,color,thickness,lineType )
            cv2.putText(img, text, (50,50), 2,0.8, (255,255,255),2,cv2.LINE_AA)

            #For very light colours we will display text in black colour
            if r+g+b >=600:
                cv2.putText(img, text, (50,50), 2,0.8, (0,0,0),2,cv2.LINE_AA)

        if cv2.waitKey(20) & 0xFF == 27:
            break

    cv2.destroyAllWindows()
else:
    print("invalid input")
```

Table 5.1. Sample Colours

Colour name	Hex code	R	G	B
Gray	#808080	93	138	168
Antique Ruby	#841b2d	132	27	45
Jet	#343434	52	52	52
Amber	#ffbf00	255	191	0

The displayed colour along with the RGB values will be displayed on the image (In an image the place where we want to identify the colour). In the entire input frame when we are selecting the region where we want to know the exact colour [3]-[7], we are selecting the region by using mouse click for example either by left button double click or by right button double click. In this case, since we need to identify the colour of the iris the process identifies the colour of the Iris can be just automated.

When the function is executed the pixels values at the target region is taken and find the distance among them and displaying the colour in small rectangular box along with RGB values [8]. This data is stored in database against the name of the person and his/her iris data. “Fig. 2, 3” shows the colour of the Iris of two different persons.

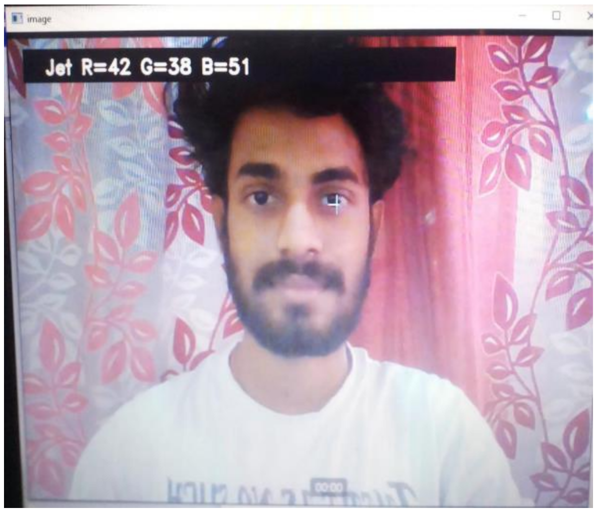


Fig.5.1. Iris colour detected of a person1

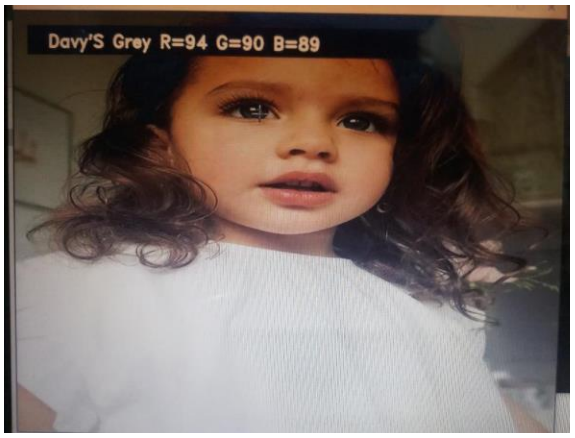


Fig.5.2. Iris colour detected of a person2

CHAPTER 6

Scale Invariant Feature Transform (SIFT)

SIFT

The scale-invariant feature transform (SIFT) is an algorithm used to detect and describe local features in digital images. It locates certain *key points* and then furnishes them with quantitative information (so-called *descriptors*) which can for example be used for object recognition. The descriptors are supposed to be invariant against various transformations which might make images look different although they represent the same object(s).

We now have our final set of key points (well, almost) and will, as the last step, compute the *descriptors* for each of them.

This step is pretty similar to the one above. We will again compute a histogram for the distribution of the directions of the gradients in a neighbourhood of each key point. The difference is that this time the neighbourhood is a circle and the coordinate system is rotated to match the reference orientation. Also, the full truth is that we not only compute *one*, but rather *sixteen* histograms. Each histogram corresponds to a point near the center of the new coordinate system and the contribution of each gradient from within the circle-shaped neighbourhood is distributed over these histograms according to proximity.

(Also, as a minor technical detail, some key points might be discarded at this last step if their circle wouldn't fit into the image.)

You can click on the key points above to see the neighbourhood and the coordinate system used for the descriptor generation below. You will also see a rendering of the actual descriptor, i.e. of the histograms (which are normalized and represented internally as 4×4×8=128 8-bit integers). (Like above, one should actually imagine the histograms to be rendered as pie charts because we're talking about angles here.)

So, what do we have now? We have a potentially large set of descriptors. Practical experience has shown that these descriptors can often be used to identify objects in images even if they are depicted with different illumination, from a different perspective, or in a different size compared to a reference image. Why does this work? Here are some reasons:

- Key points are extracted at different scales and blur levels and all subsequent computations are performed within the scale space framework. This will make the descriptors invariant to image scaling and small changes in perspective.
- Computation relative to a reference orientation is supposed to make the descriptors robust against rotation.
- Likewise, the descriptor information is stored relative to the key point position and thus invariant against translations.

- Many potential key points are discarded if they are deemed unstable or hard to locate precisely. The remaining key points should thus be relatively immune to image noise.
- The histograms are normalized at the end which means the descriptors will not store the magnitudes of the gradients, only their relations to each other. This should make the descriptors invariant against global, uniform illumination changes.
- The histogram values are also thresholded to reduce the influence of large gradients. This will make the information partly immune to local, non-uniform changes in illumination.

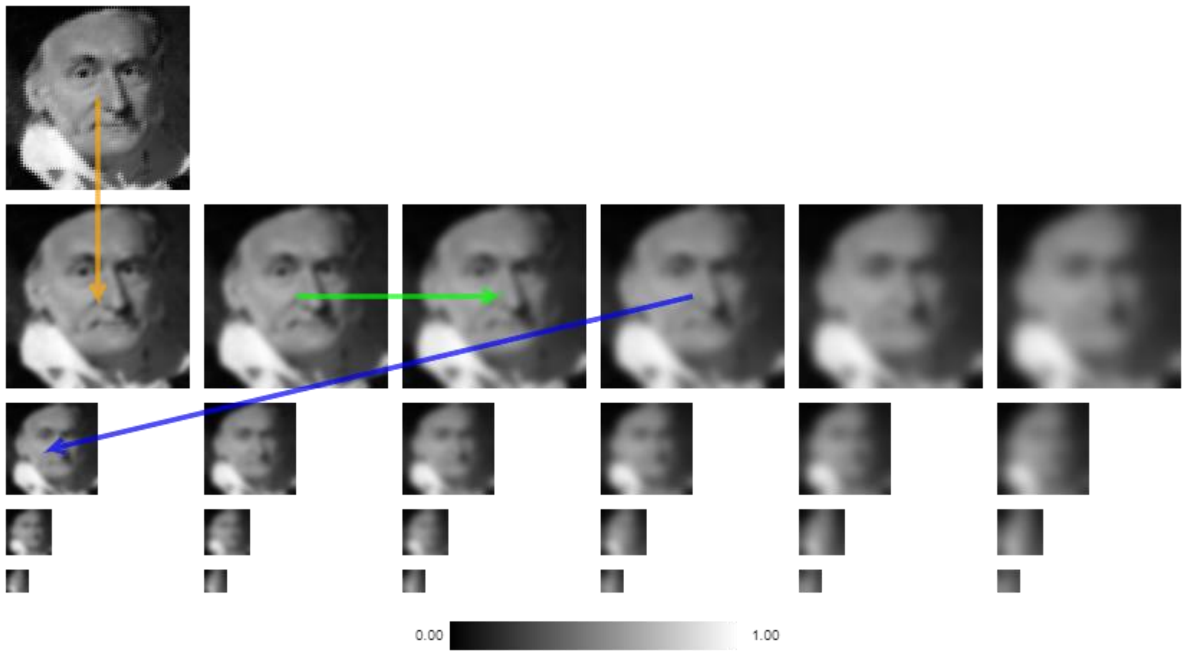


Fig6.1: Histogram values are thresholded to reduce the influence of large gradients.

We start with the picture you provided or with our default picture, a portrait of Carl Friedrich Gauß. (Or actually we start with the 64x64 version you see at the top of the page. We might have shrunk your original picture in order to keep the size of this page manageable).

The algorithm first doubles the width and height of its input using bilinear interpolation. That is the first picture above, the one in its own row.

This picture is subsequently blurred using a Gaussian convolution. That's indicated by the orange arrow.

What follows is a sequence of further convolutions with increasing standard deviation. Each picture further to the right is the result of convoluting its left neighbour, as indicated by the green arrow.

Finally, the antepenultimate picture of each row is down sampled - see the blue arrow. This starts another row of convolutions. We repeat this process until the pictures are too small to proceed.

(By the way, each row is usually called an *octave* since the sampling rate is decreased by a factor of two per stage.)

What we now have constructed is called a *scale space*. The point of doing this is to simulate different scales of observation (as you move further down in the table) and to suppress fine-scale structures (as you move to the right).

Note that the representation above has been normalized - see the gray chart at its bottom. This will be especially noticeable for low-contrast images. (An input with full contrast will have black at 0.00 and white at 1.00.)

Now for the next step. Let's imagine for a moment that each octave of our scale space were a continuous space with three dimensions: the *x* and *y* coordinates of the pixels and the standard deviation of the convolution. In an ideal world, we would now want to compute the Laplacian of the *scale-space function* which assigns gray values to each element of this space. The extrema of the Laplacian would then be candidates for the *key points* our algorithm is looking for. But as we have to work in a discrete approximation of this continuous space, we'll instead use a technique called difference of Gaussians.

For each pair of horizontally adjacent pictures in the table above, we compute the differences of the individual pixels.

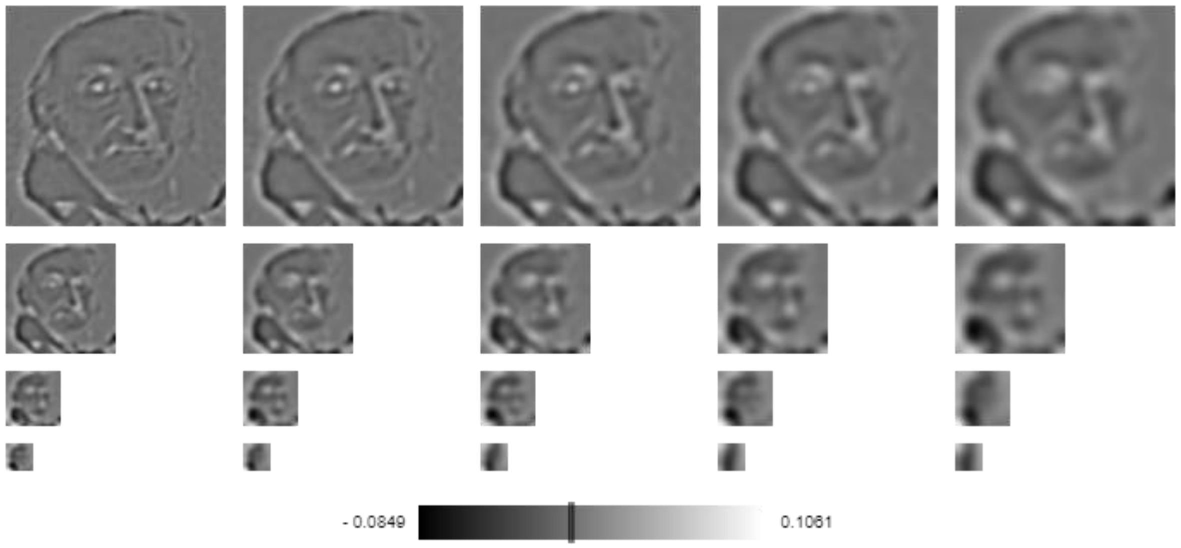


Fig 6.2: Assigning gray values to each element

If you click on one of the pixels above, you will see below how the difference for this individual pixel was calculated. You'll see a clipping of the difference image in the middle while to the left and right you'll see the corresponding clippings from the two scale space images which were subtracted. Note that bright spots in the difference image mean there was an increase in brightness while dark spots mean the opposite. Medium gray (see the marker in the gray chart above) indicates that there was no change.

(All the differences are usually comparatively small, by the way. If the difference images hadn't been normalized, we'd see mostly or only medium gray.)

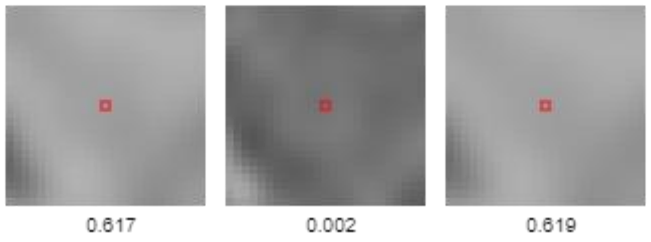


Fig 6.3: Identifying the bright spots in the difference image mean

The discrete extrema of these difference images will now be good approximations for the actual extrema we talked about above. A *discrete maximum* in our case is a pixel whose gray value is larger than those of all of its 26 neighbour pixels; and a *discrete minimum* is of course defined in an analogous way. Here we count as "neighbours" the eight adjacent pixels in the same picture, the corresponding two pixels in the adjacent pictures in the same octave, and finally *their* neighbors in the same picture.

The extrema we've found are marked below. (Some are marked with yellow circles. These are indeed extrema, but their absolute values are so small that we'll discard them before proceeding. The algorithm assumes that it's likely these extrema exist only due to image noise.)

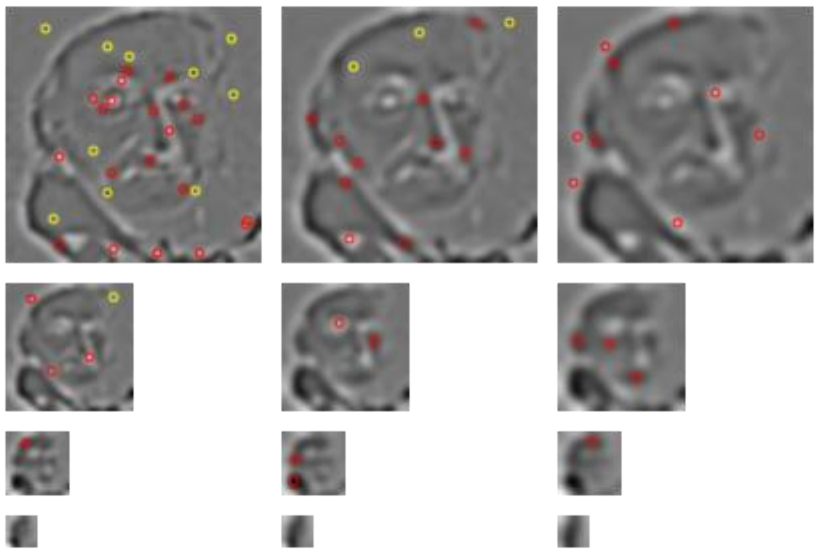


Fig 6.4: Finding the extrema values

You can click on each of the extrema above to see the pixel and its 26 neighbours rendered below. (Note that the values shown are of course rounded and thus some of the neighbouring values might look identical to the extremal value although in reality they aren't.)

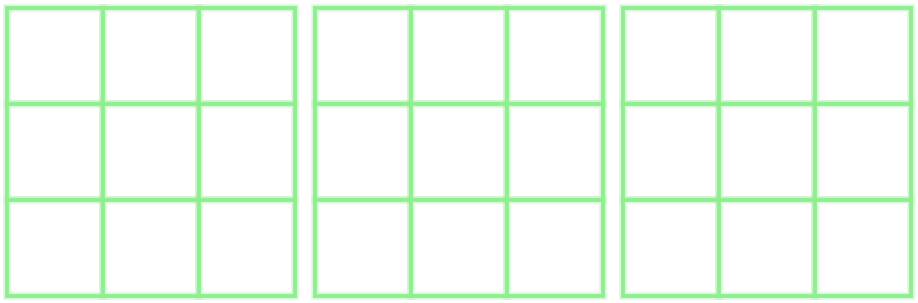


Fig5: course rounded and thus some of the neighbouring values might look identical to the extremal value

The extrema we've found so far will of course have discrete coordinates. We now try to refine these coordinates. This is done (for each extremum) by approximating the quadratic Taylor expansion of the scale-space function and computing its extrema. (The gradient and the Hessian are approximated using finite differences.) This is an iterative process and either we are able to refine the location or we give up after a couple of steps and discard the point.

Now that we have better ("continuous") coordinates, we also do a bit more. We try to identify (and discard) key point candidates which lie on edges. These aren't good key points as they are invariant to translations parallel to the edge direction. Edge extrema are found by comparing the principal curvatures of the scale-space function (or rather its projection onto the picture plane) at the corresponding locations. (This is done with the help of the trace and the determinant of the Hessian, but we won't discuss the details here.)

The remaining key points are shown below. (As we now have better estimates regarding their position, we can also discard some more low-contrast points. These are again marked with yellow colour.)

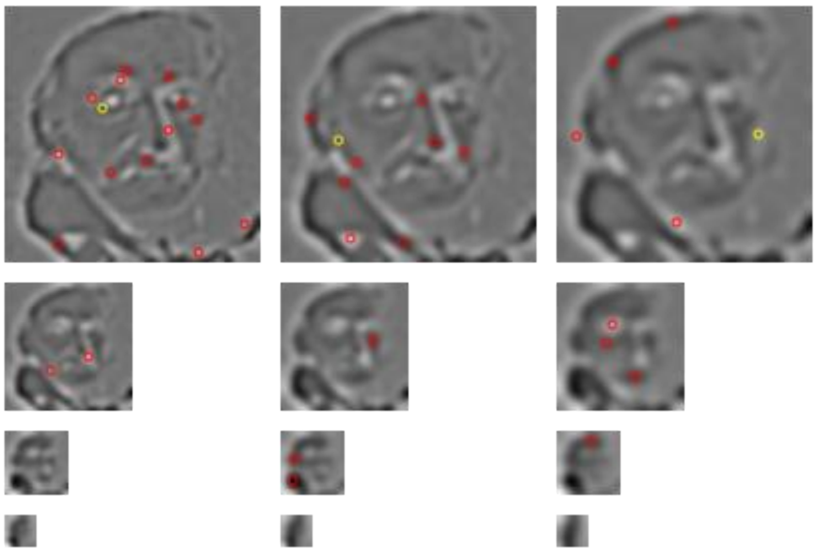


Fig 6.5 : Finding better estimates regarding their position.

You can click on the key points above to see in the table below how their scale-space coordinates have been refined.

	discrete	**interpolated**
x		
y		
scale		

You might have the impression that there are some "new" points which weren't among the extrema further above. But these will be points which moved from one scale picture to another one. (For example, if a point was originally in the middle, i.e. if its scale value had been 2, the refined value could now be 2.57. That would mean it'd now appear on the right as the nearest integer would now be 3.)

The algorithm now assigns to each remaining key point its *reference orientation*, if possible. Very roughly, we observe all gradients in the direct neighbourhood of such a point and see if many of them have approximately the same direction.

(The technical details are as follows: For each pixel in a square-shaped patch around the key point, we approximate the gradient using finite differences. Recall that the gradient points in the direction of the greatest increase and its magnitude is the slope in that direction. The interval from 0 to 360 degrees is divided into a fixed number of bins (36 by default) and the value of the bin the gradient's direction belongs to is incremented by the gradient's magnitude after it has been multiplied with a Gaussian weight. The latter is done to reduce the contribution of more distant pixels. The resulting histogram, i.e. the list of bins, is then smoothed by repeated box blurs. Finally, extrema of this histogram are identified and selected if their value exceeds a certain threshold. A better approximation for the reference orientation is then computed as the maximum of the quadratic interpolation of the histogram extremum and the values in its two neighbouring bins.)

Key points near the image border which don't have enough neighbouring pixels to compute a reference orientation are discarded. Key points without a dominating orientation are also discarded. On the other hand, key points with more than one dominating orientation might appear more than once in the next steps, namely once per orientation.

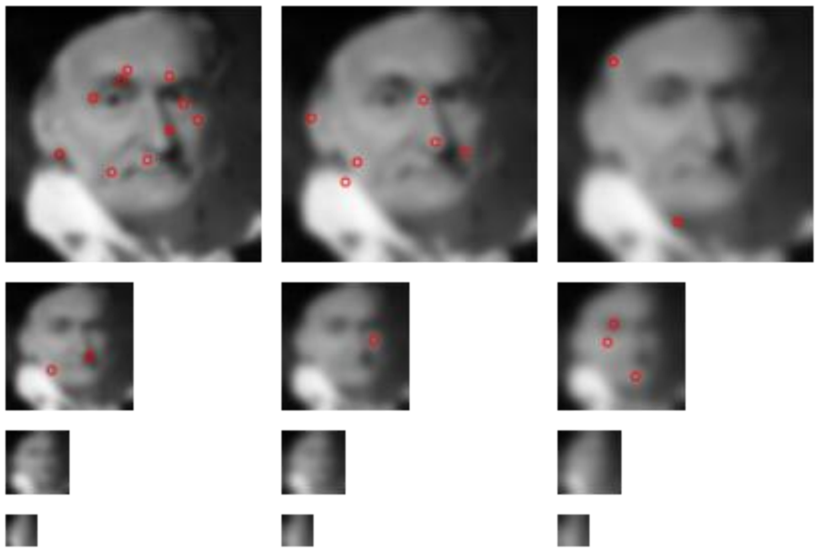


Fig 6.6 : Extrema of this histogram are identified and finding Threshold values

If you click on one of the key points above, you will see below to the left the part of its neighbourhood that was investigated and the reference orientation that was computed. To the right, you will see the (smoothed and normalized) histogram from which this orientation was derived.

We now have our final set of key points (well, almost) and will, as the last step, compute the *descriptors* for each of them.

This step is pretty similar to the one above. We will again compute a histogram for the distribution of the directions of the gradients in a neighbourhood of each key point. The difference is that this time the neighbourhood is a circle and the coordinate system is rotated to match the reference orientation. Also, the full truth is that we not only compute *one*, but rather *sixteen* histograms. Each histogram corresponds to a point near the center of the new coordinate system and the contribution of each gradient from within the circle-shaped neighbourhood is distributed over these histograms according to proximity.

(Also, as a minor technical detail, some key points might be discarded at this last step if their circle wouldn't fit into the image.)

You can click on the key points above to see the neighbourhood and the coordinate system used for the descriptor generation below. You will also see a rendering of the actual descriptor, i.e., of the histograms (which are normalized and represented internally as 4×4×8=128 8-bit integers). (Like above, one should actually imagine the histograms to be rendered as pie charts because we're talking about angles here.)

So, what do we have now? We have a potentially large set of descriptors. Practical experience has shown that these descriptors can often be used to identify objects in images even if they are depicted with different illumination, from a different perspective, or in a different size compared to a reference image. Why does this work? Here are some reasons:

- Key points are extracted at different scales and blur levels and all subsequent computations are performed within the scale space framework. This will make the descriptors invariant to image scaling and small changes in perspective.
- Computation relative to a reference orientation is supposed to make the descriptors robust against rotation.
- Likewise, the descriptor information is stored relative to the key point position and thus invariant against translations.
- Many potential key points are discarded if they are deemed unstable or hard to locate precisely. The remaining key points should thus be relatively immune to image noise.
- The histograms are normalized at the end which means the descriptors will not store the magnitudes of the gradients, only their relations to each other. This should make the descriptors invariant against global, uniform illumination changes.
- The histogram values are also thresholded to reduce the influence of large gradients. This will make the information partly immune to local, non-uniform changes in illumination.

6.1 SIFT(Scale-invariant feature transform)

The main steps of SIFT is Feature point Detection

6.1.1 Feature point detection

As its name shows, SIFT has the property of ***scale invariance***, which makes it better than Harris. Harris is not scale-invariant, a ***corner*** may become an ***edge*** if the scale changes, as shown in the following image.

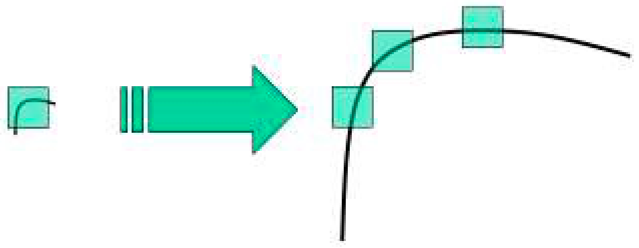


Fig 6.7: Feature point detection

An inherent property of objects in the world is that they only exist as meaningful entities over certain ranges of scale. A simple example is the concept of a branch of a tree, which makes sense only at a scale from, say, a few centimeters to at most a few meters. It is meaningless to discuss the tree concept at the nanometer or the kilometer level. At those scales it is more relevant to talk about the molecules that form the leaves of the tree, or the forest in which the tree grows. Similarly, it is only meaningful to talk about a cloud over a certain range of coarse scales. At finer scales it is more appropriate to consider the individual droplets, which in turn consist of water molecules, which consist of atoms, which consist of protons and electrons etc.

The scale of an image landmark is its (rough) diameter in the image. It is denoted by σ, which is measured in pixels, you can think scale invariance as that we can detect similar landmarks even if their scale is different.

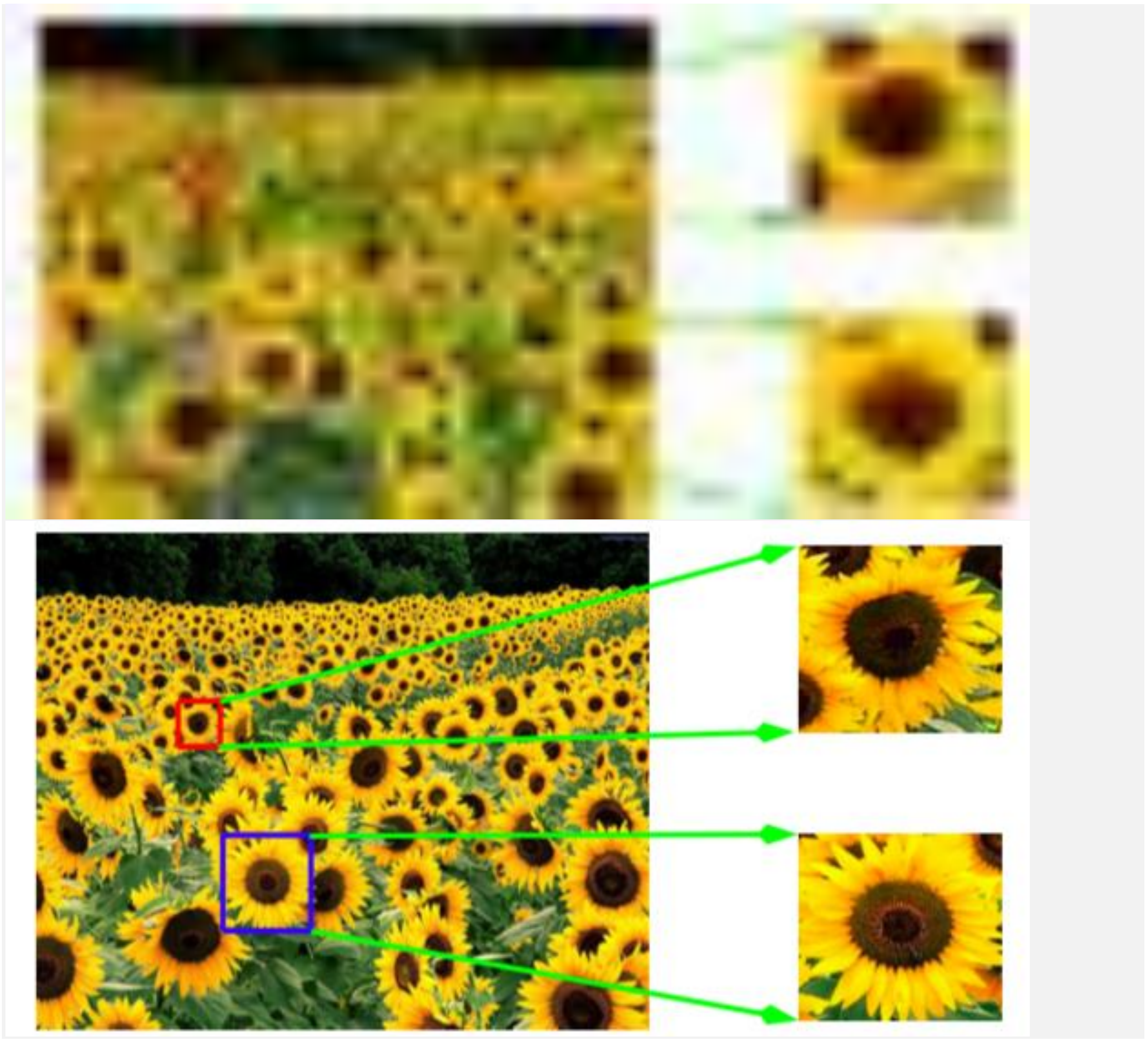


Fig 6.8: scale invariance as that we can detect similar landmarks.

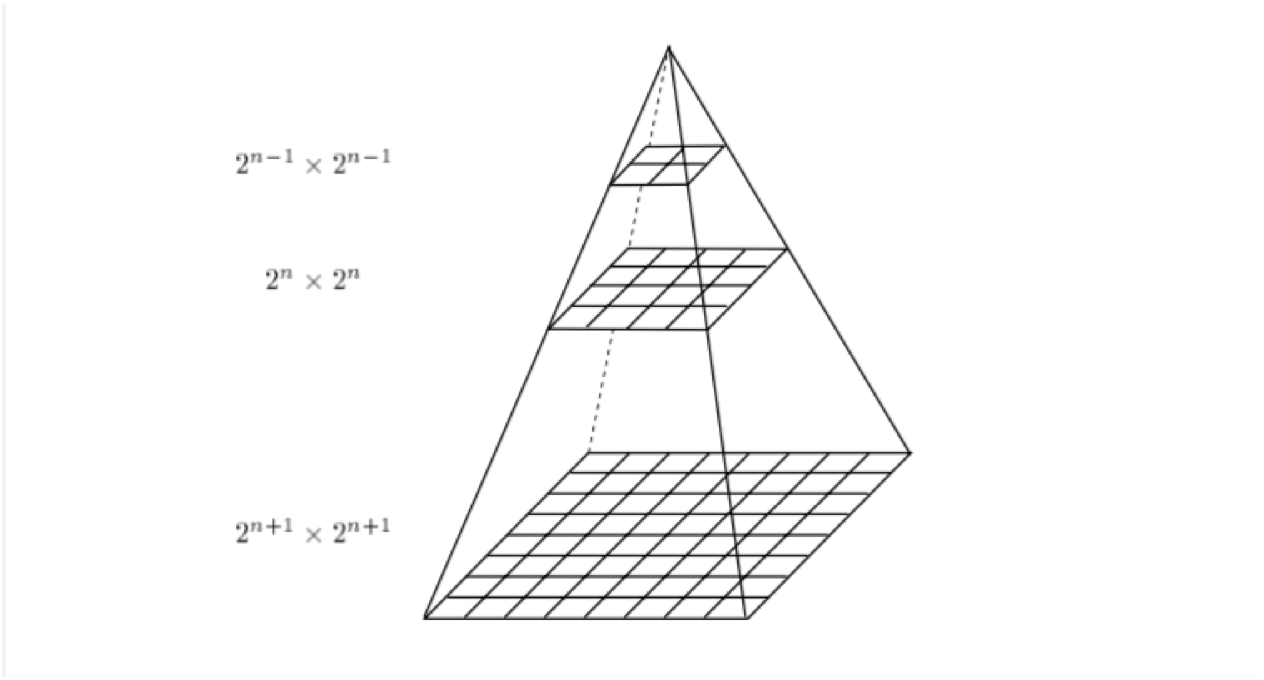


Fig 6.9: A pyramid representation is obtained by successively reducing the image size by combined smoothing and sampling.

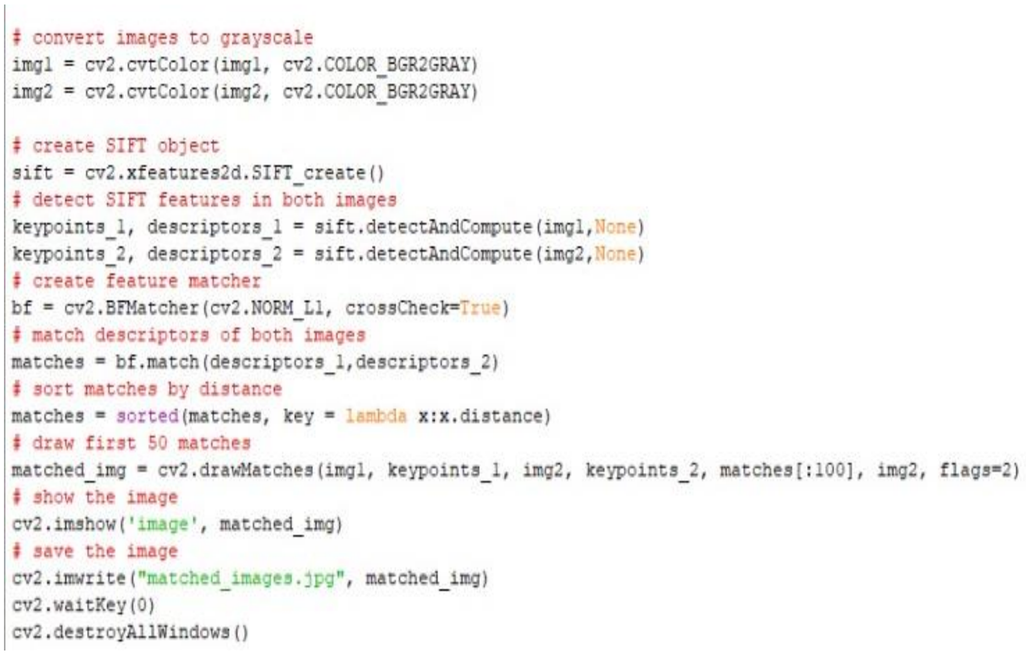
We can find the features under various image sizes. Besides, we can also use the Laplacian of Gaussian(LoG) with different σ to achieve this.Let's first have a look at LoG. As Li Yin's article indicates, the LoG operation goes like this. You take an image, and blur it a little (using Gaussian kernel). And then, you calculate the sum of second-order derivatives on it (or, the "Laplacian"). This locates edges and corners on the image. These edges and corners are good for finding keypoints (note that we want a key**point** detector, which means we will do some extra operations to suppress the edge). LoG is often used for ***blob detection*** (I will explain it later). Remember the relationship between convolution and differentiation.

$$\frac{d}{dx}(f(x) * g(x)) = (\frac{d}{dx}f(x)) * g(x)$$

Take a 1-D example, ***f*** is a scanline of an image (i.e. the pixel array from a line of an image).

In this project let us take the sample data.Let us consider the there is a huge amount of biometric data based on iris data (including colour of the iris) of people in an organization. Table 2. gives the sample table of the iris colour data of people in an organization.

The Feature matching done on Iris achieved through the code below:



```
# convert images to grayscale
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# create SIFT object
sift = cv2.xfeatures2d.SIFT_create()
# detect SIFT features in both images
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)
# create feature matcher
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)
# match descriptors of both images
matches = bf.match(descriptors_1,descriptors_2)
# sort matches by distance
matches = sorted(matches, key = lambda x:x.distance)
# draw first 50 matches
matched_img = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2, matches[:100], img2, flags=2)
# show the image
cv2.imshow('image', matched_img)
# save the image
cv2.imwrite("matched_images.jpg", matched_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Table.6.1. Sample data of people in an organization with their iris colour data present against their name

S no.	Name of the person	Iris colour
1.	John	red
2.	Rahul	gray
3.	Raju	hazel
4.	Rahim	red

Let us consider john is the person on which we need to find the biometrics. firstly, after detecting the colour from the data base, we pick the data off all the members with the matched colour. Table 3. shows the reduced data on which we need to scan to find the biometric of john (i.e., all the columns with red iris colour gets selected).

Table.6.2 Reduced data table based on the identified colour

S no.	Name of the person	Iris colour
1.	John	red
4.	Rahim	red

Later, using input iris data of the person to be matched is matched with the data present in the reduced data through sift. The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images [8],[9]. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors. From the full set of matches, subsets of key points that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches.

CHAPTER 7

RESULTS

The displayed colour along with the RGB values will be displayed on the image (In an image the place where we want to identify the colour) .

In the entire input frame when we are selecting the region where we want to know the exact colour. The precise colour along with their RGB values are displayed. The following figures shows the results

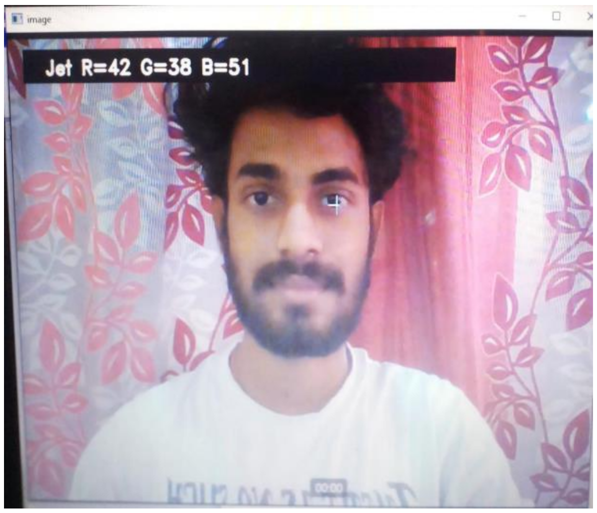


Fig.7.1. Iris colour detected of a person1

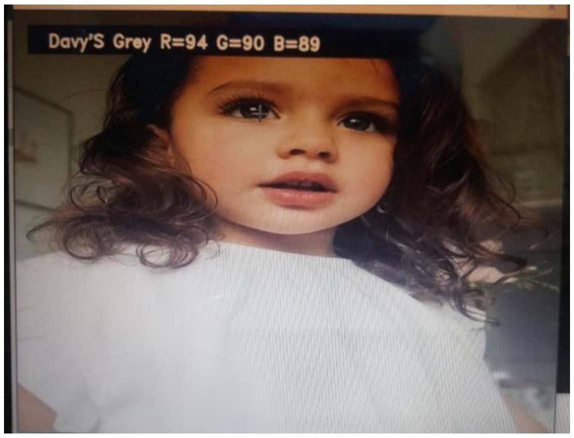


Fig.7.2. Iris colour detected of a person2

After detecting the colour of the Iris, through the SIFT algorithm, the Iris pattern (i.e., rings, crypts, furrows) is matched and checked are for authenticity as shown on the figure below.

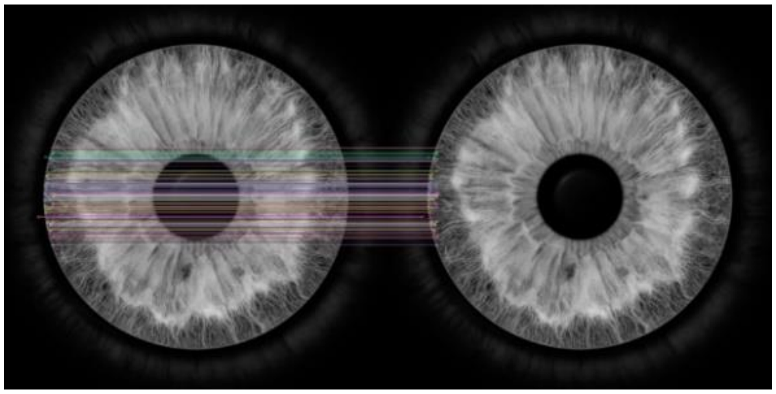


Fig.7.3. Iris feature matching done through SIFT algorithm on an Iris (sample 1)

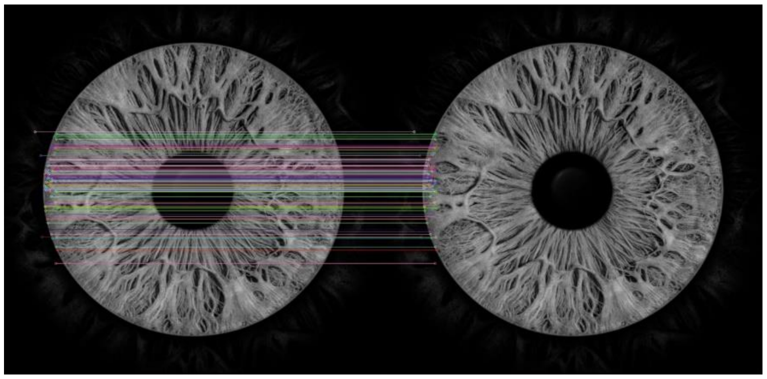


Fig.7.4. Iris feature matching done through SIFT algorithm on an Iris (sample 2)

CONCLUSIONS

Therefore, through SIFT algorithm we have identified the identical match and found the particular person's biometrics. "Fig. 4" shows the feature matching of Iris using SIFT algorithm. Also, because we have developed a process to identify the exact name of the colour in the first place, the data to be matched is reduced to maximum extent, thereby improving the system efficiency and therefore the load on the server gets reduced resulting in faster response.

In the above sample process, we have seen that the results are obtained 50% faster since the iris of john is tested only with the two persons instead of four. The system improved efficiency can be seen more evident if we consider large data like data of people in an entire state or the data of an entire country.

The work carried out in this paper is to improve system efficiency during Iris detection by preliminary screening of data on basis of colour. From the result obtained we could see that the process we have developed can identify the colour of the Iris. On the basis of these results, it is clear that we can improve the efficiency of the biometric systems as a particular person's biometric information is searched against only a limited set of data where the result colour matches with the colour of the Iris in the database, thereby improving the system efficiency. The motivation behind the formulating of this process is to develop more efficient biometrics system.

FUTURE WORK

The process formulated is a prototype. A large database is created to maintain the records of the Iris colour and Iris data using a more detailed images of the eye with a sophisticated, high-resolution digital camera at visible or infrared wavelengths. This process can be extended to applications like biometric attendance system, it can also be used in voting systems due to its high efficiency and precision. It can also be used in identifying the

REFERENCES

[1] A. K. Jain, R. Bolle, and S. Pankanti, "Biometrics: personal identification in networked society",vol. 1. 1999, p. 434.

[2] A. K. Jain, A. Ross, and S. Prabhakar, "An Introduction to Biometric Recognition," IEEE Trans. Circuits Syst. Video Technol., vol. 14, pp. 4–20, 2004.

[3] Principles of Digital Image Processing Advanced Methods,Wilhelm Burger, Mark J. Burge.

[4] Deep Learning with Python and OpenCV,A beginner's guide to perform smart image processing techniques using TensorFlow and Keras Viral Thakar.

[5] OpenCV 2 Computer Vision Application Programming Cookbook,Robert Laganière.

[6] Python 3 Image Processing,Learn Image Processing with Python 3, NumPy, Matplotlib, and Scikit-image,Ashwin Pajankar.

[7] October 2018 International Journal of Electrical and Computer Engineering 8(5):3278-3284 DOI:10.11591/ijece.v8i5.pp.3278-3284 Project: pattern recognition , image processing

[8] Angela Chau, Ezinne Oji, Jeff Walters Dept. of Electrical Engineering Stanford University Stanford, CA 94305 angichau, ezinne, jwalt@stanford.edu

[9] Colour-based Object Detection To cite this article: Hani Hunud A Kadouf and Yasir Mohd Mustafah 2013 IOP Conf. Ser.: Mater. Sci.