Smart Surveillance Parking Camera

Thesis submitted by

Jeet Thakore (181020011008)
Satyajeet Kumar Verma (181020012015)

Under the Guidance of
Dr. Raghavendra H. Bhalerao
Assistant Professor, EECS (IITRAM)

in partial fulfillment of the requirements for the award of

Bachelor of Technology

Department of Electrical Engineering and Computer Science
Institute of Infrastructure Technology Research and Management, Ahmedabad, Gujarat - 380026.

May,2022

Self-declaration

We declare that this written submission represents our ideas in our own words, and where ideas or words of others have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken.

Place:	Signature(s):
Date:	Jeet P Thakore, Satyajeet K Verma
	(**181020011008, 181020012015**

Approval sheet

The thesis entitled "**Smart Surveillance Parking Camera**" submitted by **Jeet Thakore, Satyajeet K Verma** is the outcome of the work done during the academic period 2021 - 2022 under my supervision for the partial fulfilment of the requirements for the award of degree of **Bachelor of Technology**. The extent of similarity content found in this submission is acceptable for me.

Place: Signature:

Date Name of the supervisor: **Dr. Raghavendra Bhalerao**

The examination committee hereby approves that the thesis entitled "**Smart Surveillance Parking Camera**" submitted by **Jeet Thakore, Satyajeet K Verma** can be considered for submission in the department of Electrical Engineering at Institute of Infrastructure Technology Research and Management, Ahmedabad.

Name of the supervisor: Dr. Raghavendra Bhalerao
Affiliation: Dept. of Electrical and Computer Science Engineering, IITRAM, Ahmed-abad
Signature with date:

Name of the examiner 1: Dr. Ram Narayan Yadav
Affiliation: Dept. of Electrical and Computer Science Engineering, IITRAM, Ahmed-abad
Signature with date:

Name of the examiner 2: Dr. Manish Chaturvedi
Affiliation: Dept. of Electrical and Computer Science Engineering, IITRAM, Ahmed-abad
Signature with date:

Name of the examiner 3: Dr. Manish Sharma
Affiliation: Dept. of Electrical and Computer Science Engineering, IITRAM, Ahmed-abad
Signature with date:

Acknowledgements

We wish to express our heartfelt gratitude to our guide, Dr. Raghavendra Bhalerao, Department of Electrical and Computer Science Engineering, IITRAM Ahmedabad, for encouraging us to work on our present topic and for his inspiring guidance, constructive criticism, as well as valuable suggestions throughout the course of this project. We would also like to thank the Institute for giving us access to various resources and material, thus enabling us to work on this project. Last but not the least, we would like to thank our families for their constant encouragement and moral support throughout.

Dedicated to our parents and almighty

"An Image is worth a thousand words..."
-Henrik Ibsen

Abstract

A tremendous amount of electrical energy is invested in the illumination of underground Parking lots. While this illumination is immensely crucial to navigate through the parking lot, it does lead to a significant loss of electricity. Also, completely stopping the use of artificial lighting to illuminate the parking space may not be a viable option because most parking lots are situated underground, where the natural lighting is non-accessible. Artificial lighting, hence, is an inevitable component in such cases, and cannot be completely disregarded. However, it can be controlled. One way to mitigate this issue is by utilizing the lights in a systematic manner. The "Smart Surveillance Parking Camera" is one such possible implementation. This project aims at reducing the overall energy consumption of the parking/basement lights by incorporating computer vision. Rather than having all the lights turned on, only those lights will be turned on which have a vehicle underneath them. The vehicle's movement will be tracked, and only the light(s) corresponding to that location will be turned on, hence eliminating the need to always have all the lights on, which would have caused a lot of energy loss. At the rudimentary level of this project, a computer would be used to perform the required image processing but in future, this project can also be reconciled with a stand-alone microcontroller, in order to eliminate the need of a computer. This would improve its portability drastically. This project predominantly deals with underground parking lots but can also be extended to situations where a similar illumination control is desired. With increasing awareness about effective energy utilization and sustainable development, this project will definitely be an essential contributor in the overall process of energy conservation.

Keywords: Digital Image Processing, Computer Vision, Object Detection, Contour Detection, Image Moments, Contour Centroid Computation, OpenCV, PyFirmata, Trackbars, Warp Perspective Numpy Arrays, Relays, Arduino UNO, Python, Illumination Control.

List Of Figures

Chapter 1

Introduction

1.1 Overview

Light and vision have always been areas of avid interest since the dawn of time. Numerous curiosity-driven attempts have been made by eminent scholars and researchers in this field which have led to fine discoveries and astonishing results. Humans have been curious around the notion of illumination since time immemorial. Light or illumination plays a pivotal role in our daily lives. At present, our illumination demands are either met by natural sunlight or artificial lighting. In night-time conditions or in places where light is hard to pass/reflect, one has to solely rely upon artificial lighting. One such scenario is underground parking lots. Located below the ground level, this place thrives solely on electrical lights. Needless to say, A significant amount of energy is spent to illuminate such places. However, there could be a way through which artificial illumination can be achieved in this case along with minimal energy usage, and that is by using controlled illumination. A Trade-Off is needed to be made between precise lighting control and energy saving. Its worth noting that the lighting needs to be precisely controlled in order to avoid fatal vehicular accidents. The idea is that rather than having all the lights turned on, only those lights will be turned on which have a vehicle underneath them. The vehicle's movement will be tracked, and only the light(s) corresponding to that location will be turned on, hence eliminating the need to have all the lights on at all times, which would have caused a lot of energy loss.

On a broader scale, with urbanization rate increasing rapidly, we are no strangers to the fact that underground parking lots will be an even more common sight in almost all the leading cities of our nation. If dedicated efforts are made to minimize the energy consumption of all these parking lots even by a marginal amount, then there would be a humongous chunk of energy that will be saved from getting wasted which might suffice the energy requirements of hundreds if not thousand small villages and municipalities.

A humble attempt has been made in this direction using the combined knowledge of digital image processing and computer vision.

1.2 Literature Survey

Paper 1

Study on Object Detection using Open CV - Python

Object detection is a well-known computer technology that focuses on finding objects or instances of a given class (such as persons, flowers, and animals) in digital photos and videos. Face detection, character recognition, and vehicle calculation are just a few examples of well-researched object detection applications. The fundamental goal of studying and researching computer vision is to use a computer to directly replicate the behavior and manner of human eyes, and then design a system that can do so. The purpose of this research is to use several ways to identify things placed over a surface from a complex backdrop image. The main goal is to distinguish a specific object among a vast number of objects in real time. When there are a lot of recognizable items, most recognition methods aren't very scalable. As the number of items grows, so does the computational cost. Because two objects may have the same qualities, comparing and querying photos based on colour, texture, and shape is insufficient. It's challenging to create a recognition system that can work in a dynamic environment and behave like a human. Lighting, dynamic backgrounds, the existence of shadows, camera motion, the speed of moving objects, and intermittent object motion are some of the major issues in designing an object recognition system. The potential for applying computer vision to real-world situations is enormous. The fundamentals of object detection, as well as numerous methods for attaining it and its scope, have been covered. The primary processes in object detection are feature interpretation and matching, which should be done well and with high accuracy. Deep Face is the most effective face detection method, and it is favoured by most social media apps such as Facebook, Snapchat, and Instagram over Haar-Cascade. OpenCV will become extremely popular among coders in the near future, as well as a top demand for IT firms

Paper 2:

Face Detection and tracking using OpenCV

The goal of computer vision is to directly imitate the behaviour of human eyes using computers. This paper discusses an application for automatic face detection and tracking on video streams from surveillance cameras in public or commercial areas. It is useful to detect where people are looking in numerous circumstances, such as exhibits, shopping malls, and public areas of buildings. Prototype is built on open source platforms Arduino and OpenCV and is designed to operate with web cams for face detection and tracking. The system is based on AdaBoost algorithm and abstracts faces Haar-Like features. The AdaBoost algorithm trains the key category features to the weak classifiers, and cascades them into a strong classifier for face detection. A program is developed using OpenCV that can detect people's face and also track from the web camera.

The image of the face captured by web-cam with the help of Processing, OpenCV undergoes different steps which are generating rectangle class which keeps track of the face coordinates. Create an instance of the OpenCV library. This serial library is needed to communicate with the Arduino. Adjust Screen Size Parameters on contrast/brightness values. Convert the image coming from webcam to greyscale format. Find out if any faces were detected. If a face is found, find the midpoint of the first face in the frame. Manipulate these values to find the midpoint of the rectangle.

By using this approach it was found that time taken to detect the face was less than 1 second which means that this setup can be used in real time. The detection efficiency was greatly improved by using OpenCV. The average frame rate was found to be 15 fps.

Paper 3

Real time object detection and tracking using Deep Learning and OpenCV

Deep Learning has had a significant impact on how the world has adapted to artificial intelligence in recent years. Region-based convolution Neural Networks (RCNN), Faster RCNN, single shot detector (SSD), and You Only Look Once are some of the most popular object detection techniques (YOLO). Faster- RCNN and SSD have superior accuracy, however YOLO works better when speed is prioritized over accuracy. Deep learning combines SSD and Mobile Nets for effective detection and tracking implementation. This algorithm detects objects quickly and efficiently without sacrificing performance. In this paper, detection and tracking algorithms based on SSD and Mobile Nets are implemented in a Python environment. Object detection is the process of finding an object's region of interest from a set of images. Frame differencing, Optical flow, and Background subtraction are examples of different ways. With the use of a camera, this is a way of identifying and finding a moving object. The properties of picture and video are extracted to explain detection and tracking algorithms for security applications. CNN and deep learning are used to extract features. Image classification and counting are done with classifiers. Using deep learning techniques, a YOLO-based method with a GMM model will provide good feature extraction and classification accuracy. This model performed well on the object trained in terms of detection and tracking, and it can be used in certain circumstances to identify, track, and respond to specific targeted objects in video surveillance.

Paper 4

Object detection system using Arduino and Android application for Visually Impaired people

Humans receive the majority of their information through sight, hence vision is the most significant element of their body functions. According to the World Health Organization (WHO), there are 285 billion visually handicapped persons in the world, with around 13% being blind and the remainder having low vision. The suggested technology's operation is based on two major tasks: monitoring the surrounding environment and detecting impediments using sonar sensors.

This technology can detect the position of an obstruction inside the detecting range. It can be easily worn as a flexible, lightweight, and comfy clothing that is also washable. The proposed smart clothing system has the potential to become an integral component of the lives of visually impaired persons. The operating principle and applications of an Arduino board in conjunction with Android are explored in this paper. This paper also looks at how they may be utilized to develop an object detection system that can respond in voice. Many blind assistive systems have been deployed at this time, however in this research, we suggest an android-based system that identifies the object in front of the blind person. According to the barrier position, the blind person is instructed using English voice commands provided by the application. Our suggested approach will be useful in applications such as assistive robots for people with visual disabilities, as well as industrial applications like work robots. The system's strength rests in the fact that it can prove to be a very low-cost solution for millions of blind people around the world. The proposed combination of multiple functioning units creates a real-time system that can identify and inform the user to an object in front of them, making navigation safer and more secure.

1.3 Report Overview

In the initial sections of the report, a brief idea of the project is given alongside a literature survey followed by an in-depth discussion of the basic concepts involved in the project like images, histograms, filters to more advanced topics such as image operation, thresholding, noise and contour detection. Methodology has been articulated in the later sections of the report where the entire project flow has been discussed. Towards the end of the report, a conclusion of the project has been provided along with results and challenges faced.

Chapter 2

Image Acquisition and Digitization

2.1 Images and Vision

Vision is one of the most advanced sensors of living beings. The set of human eyes are responsible for efficient capturing of the world around us. The images which we perceive, however are not digital in nature. One needs to convert such naturally occurring analog image to digital format via digitization (to be discussed in the immediate next section). The discipline of digital image processing deals with developing a digital system which performs operations on the digital image. An image is nothing more than a two-dimensional signal. It is defined by the mathematical function f(x,y) where x and y are the two co-ordinates horizontally and vertically. The value of f(x,y) at any point is gives the pixel value at that point of an image. In most mammals and humans, the light enters the eye through cornea which is then focused by the lens on the retina. The retina is a light sensitive membrane located at the back of the eye. The retina acts as a transducer which converts the received photonic signals into neuronal signals. The rods and cones are the special photoreceptive cells of the retina which are responsible for this transduction. These rods and cones detect the photons of the light and respond by producing neural impulses. These signals are transmitted by the optic nerve, from the retina upstream to central ganglia in the brain.

2.2 Image Acquisition Techniques

Image Acquisition is the action of retrieving an unprocessed image from source. It is the first step in the workflow sequence because, without an image, no processing is possible.

The input energy is transformed into a voltage by the combination of input electrical power and sensor material that is responsive to a particular type of energy being detected. The output voltage waveform is the response of the sensor, and a digital quantity is obtained from each sensor by digitizing its response. The below image denotes a single image sensor.

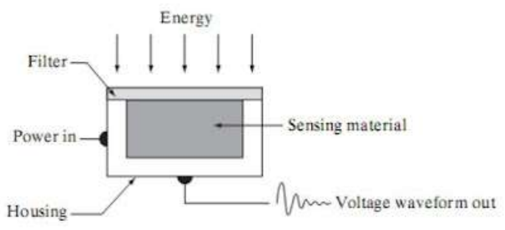


Fig.1: Single Image sensing cell

There are multiple ways through which an image may be acquired.

1. Using a Single Sensor

A single sensor is moved in 1D or 2D to get an image in (x,y) co-ordinates. The sensor is often moved using a mechanical drum or any other form of mechanical displacement. This method yields high resolution pictures and is also inexpensive, but the main drawback is that it requires mechanical displacement.

2. Using a Strip Sensor

This method involves multiple sensors connected in such a way that they either form a ring or a stick-like strip. This method also requires relative motion between the sensors and the source. The strip type method is used in flatbed scanners and the ring type is used in the MRI scanning facilities.

3. Using a Charged Couple Device Array (CCD)

This method involves an array of numerous charged couple sensors which cohesively detect the entire source which is to be captured. The best thing about this method is that no relative motion is needed between the source and the capturing device.

2.3 Digitization Process

Once an image is acquired by a sensor, it is subjected to digitization. Digitization is necessary as the image which is received from the sensor is analog in nature. Digital image processing operations can only be carried on images which are digital in nature. Continuous image data is converted to finite digital format. Digitization involves 2 processes: Sampling and quantization. Sampling is the process of discretising the independent variable i.e., the X axis (Co-ordinate Values). For sampling, a Sampling function of appropriate parameters is superimposed on the grey level plot of the given image. The number of samples used determine the overall quality of the image and noise presence in that image. Post Sampling, the image is subjected to Quantisation. Quantisation is the process of discretising the amplitude values i.e., the Y axis. Concisely speaking, Quantisation involves assignment of levels to the sampled values. The number of quantization levels should be high enough for human perception of fine shading details in the image. The occurrence of false contours is the main problem in image which has been quantized with insufficient brightness levels.

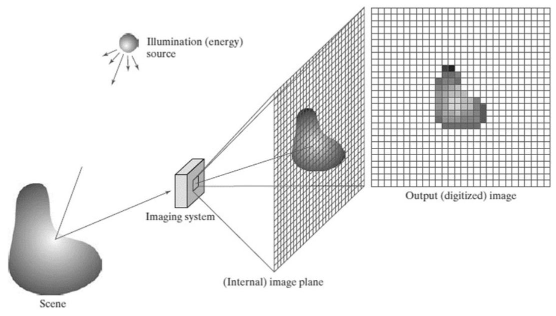
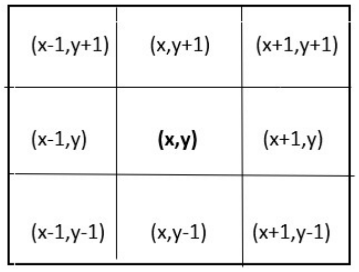


Fig.2 The Process Of Digitization

2.4 Pixels

The word “Pixel” means “Picture Element”. It’s the smallest element of any digital image. Pixel is the most rudimentary concept as far as digital image processing is concerned. The value of a pixel at any point correspond to the intensity of the light photons striking at that point. Each pixel stores a value proportional to the light intensity at that particular location. Every pixel stores an information pertaining to the overall image. If the image has been subjected to a good enough digitization process, then the pixels will be more in number. More number of pixels infer that the image will be clearer, as more details can be captured with better efficacy. Images can be classified into various categories based on the type of pixels used. An image is said to be a binary image if each of its pixel stores either black (0) or white (1) gray level. In such images one pixel holds one single bit; either a binary 0 or a binary 1. One thing to note is that a binary image doesn’t have any gray levels. The only 2 possible levels are either black or white. A grayscale image is an image in which each pixel assumes a value in the range 0 -255. 0 stands for black, and 255 stands for white, and 127 stands for gray colour. Each pixel in a grayscale image stores 1 byte (8 bits) of data. In such images, the desired colour is achieved by choosing specific mix values of R, G, B (Red, Green, Blue). Such images occupy 3 bytes per pixel.

The relation between 2 pixels is given by adjacency. Consider the following figure. The pixel count of the below image is 9. Considering their respective positions, each pixel can be assigned a certain value in the spatial domain. All the pixels in the below image have been mapped with respected to the 5th pixel (the centre of the image) as the origin.



(x-1,y+1)	(x,y+1)	(x+1,y+1)
(x-1,y)	**(x,y)**	(x+1,y)
(x-1,y-1)	(x,y-1)	(x+1,y-1)

Fig.3: Pixel Adjacency

4- Adjacency: $N_4(p)$: [(x+1,y),(x-1,y),(x,y+1),(x,y-1)]
Diagonal Adjacency: $N_D(p)$: [(x+1,y+1),(x-1,y+1),(x-1,y-1),(x+1,y-1)]
8-Adjacency: $N_8(p)$: $N_4(p)$ U $N_8(p)$

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image.

2.5 Filters

Spatial Filters

Spatial Filtering technique is used directly on pixels of an image. Mask is usually considered to be added in size so that it has specific centre pixel. This mask is moved on the image such that the centre of the mask traverses all image pixels. Spatial filters are of 2 kinds; Linear and Non- linear.

Linear:

Linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask. The idea is replacing the value of every pixel in an image by the average of the grey levels in the neighborhood define by the filter mask.

Types of Linear filter:

1. Averaging filter: It is used in reduction of the detail in image. All coefficients are equal.
2. Weighted averaging filter: In this, pixels are multiplied by different coefficients. Center pixel is multiplied by a higher value than average filter.

Non-Linear

It is based on the ordering the pixels contained in the image area encompassed by the filter. It replaces the value of the centre pixel with the value determined by the ranking result. Edges are better preserved in this filtering.

Types of Non- Linear filter:

1.Median

Replaces the value of a pixel by the median of the pixel values in the neighbourhood of that pixel

2.Max

The max filtering is achieved using the following equation

f(x,y) = max g(s,t)

3. Min

The min filtering is achieved using the following equation

f(x,y) = min g(s,t)

4 .Midpoint

Replaces the value of a pixel by the midpoint between the maximum and minimum pixels in a neighborhood.

2.6 Histograms

A Histogram is a graphical means of representing a digital image. It is a plot denoting number of pixels for each gray level/intensity. It is used to analyse an image. Properties of an image can easily be determined by the use of a histogram. The brightness and contrast of an object can also be adjusted using the knowledge of histograms. More applications of histograms involve thresholding, which improves the appearance of the image and image equalization. The below image shows a histogram of a RGB image.

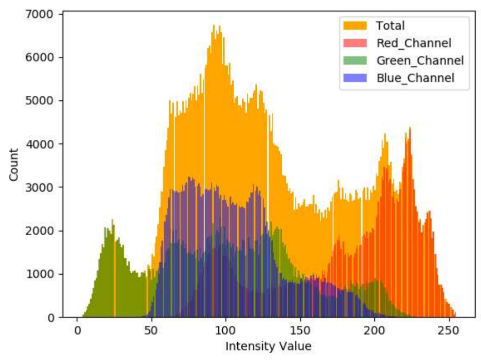


Fig.4: Histogram of an RGB image

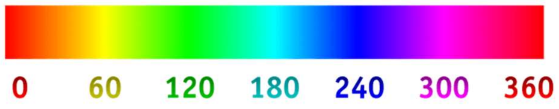
2.7 HSV vs RGB Images

RGB Colour Space

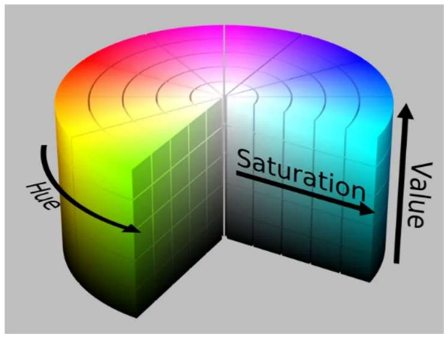
When working on image processing or computer vision, it is observed that the RGB (Red, Green, and Blue) colour space is frequently used as the default for loading frames (or images). It is the most frequently understood colour space, a three-dimensional model made up of three main colours that may be combined to create any other colour.

HSV Colour Space

HSV is a popular colour space made up of three components: hue, saturation, and value. In simple terms, Hue represents the real pure colour observed by human eyes, Saturation represents the colourfulness of that pure colour (i.e. decreasing Saturation reduces the colour's colourfulness), and Value reflects the colour's intensity, which correlates with its darkness.



(a)



(b)

Fig.5: (a) HSV Colour Space and (b) distribution

The distribution of colors in Hue component aligns well with color of the visible light spectrogram.

- Red ranges between 0 and 60 degrees.
- Yellow ranges between 61 and 120 degrees.
- Green ranges between 121-180 degrees.
- Cyan ranges between 181-240 degrees.
- Blue ranges between 241-300 degrees.
- Magenta ranges between 301-360 degrees

Why HSV is usually preferred in Image processing tasks

People prefer to use the HSV model over the RGB model in most image processing and computer vision applications. Examining the goals for which each model was created is one approach to see the differences. It was officially created/optimized for the display screen for the RGB model, but it was built to mirror how humans process colours for the HSV model. Because HSV can separate pure colours from their lightness, it may be used to conduct many operations on either the colour itself (Hue) or its intensity (Intensity) (Value). People are more interested in the colour of an image in computer vision applications than in its lightness; in fact, lightness is considered noise in most applications because it can affect the image uniqueness without causing a real change (except for the lightness); this explains why HSV makes the algorithm more robust and less sensitive to image lightness.

2.8 Object Detection

Object detection is an application of Computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Well-researched domains of object detection include face detection and pedestrian detection.

Object detection has applications in many areas of computer vision, including image retrieval and video surveillance. Object tracking is used for a variety of use cases involving different types of input footage. Whether or not the anticipated input will be an image or a video, or a real-time video vs. a pre-recorded video, impacts the algorithms used for creating object tracking applications.

2.9 Image Operations

Grayscale: The reason why the image is converted into grayscale is that grayscale image is of 2 dimensions, and some algorithm can only be applied on 2-D image rather than 3-D. The size of grayscale is also small, that enables us to do more complex operations in a shorter time.

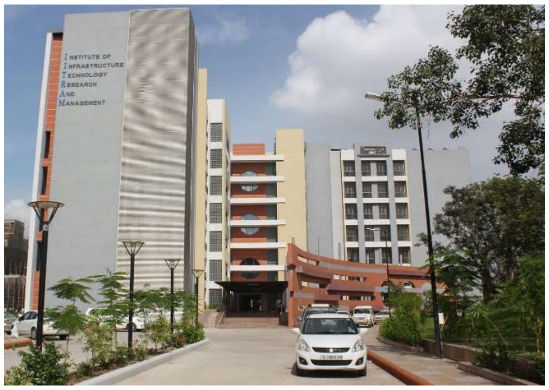


Fig.6: Original figure

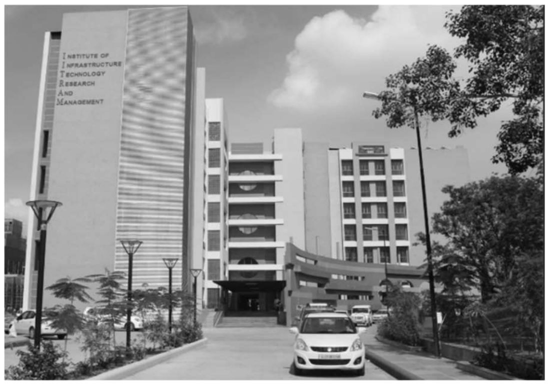


Fig.7: Grayscale Image

Blurring: The blur, or smoothing, of an image removes "outlier" pixels that may be noise in the image. Blurring is an example of applying a *low-pass filter* to an image. In computer vision, the term "low-pass filter" applies to removing noise from an image while leaving the majority of the image intact. A blur is a very common operation we need to perform before other tasks such as edge detection

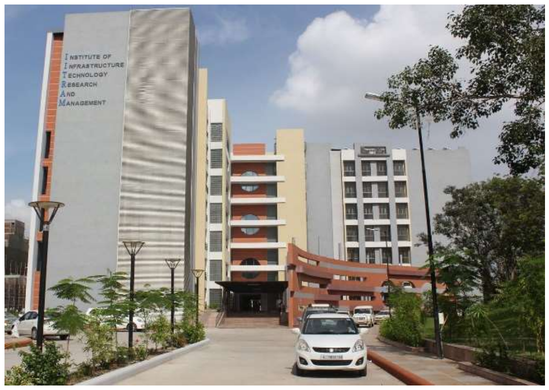


Fig.8: Original Image

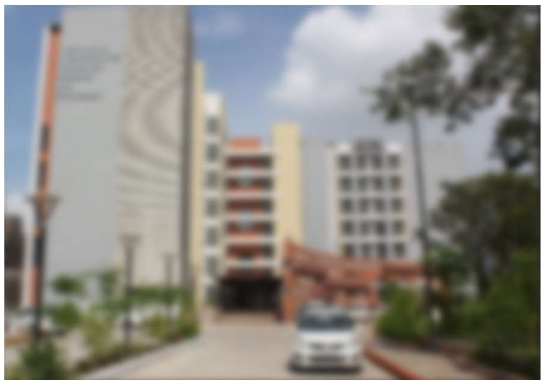


Fig.9: Blurred Image

Thresholding: Thresholding can be used as a way to select areas of interest of an image, while ignoring the parts we are not concerned with. The process of thresholding involves, comparing each pixel value of the image (pixel intensity) to a specified threshold. This divides all the pixels of the input image into 2 groups:

1.Pixels having intensity value lower than threshold.

2.Pixels having intensity value greater than threshold.

In order to threshold colour images, One approach is to designate a separate threshold for each of the RGB components of the image and then combine them with an AND operation. This reflects the way the camera works and how the data is stored in the computer, but it does not correspond to the way that people recognize colour. Therefore, the HSL and HSV colour models are more often used; note that since hue is a circular quantity it requires circular thresholding. It is also possible to use the CMYK colour model

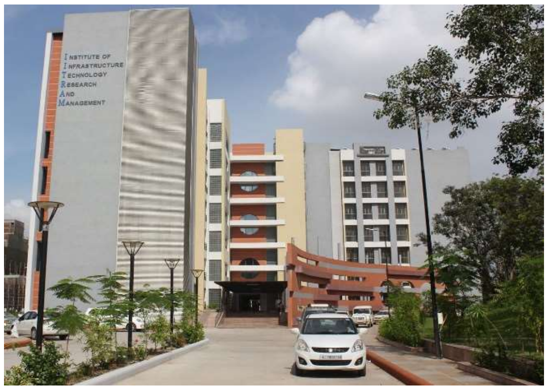


Fig.10: Original Image

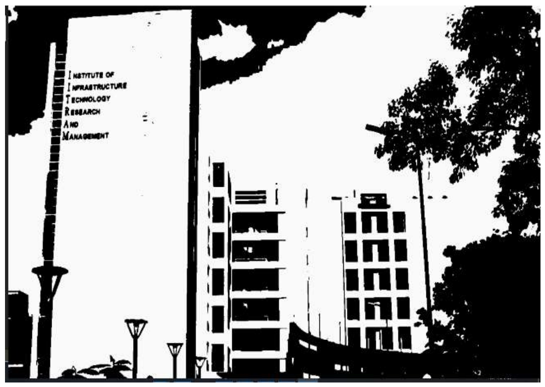


Fig.11: Thresholded Image

Dilation: Expands the image pixels , dilation adds pixels to the object boundary. The value of the output pixel is the maximum value of all the pixels in the neighbourhood. A pixel is set to 1 if any of the neighbouring pixels have the value 1. Dilation (usually represented by $\bigoplus$) is one of the basic operations in mathematical morphology. Originally developed for binary images, it has been expanded first to grayscale images, and then to complete lattices. The dilation operation usually uses a structuring element for probing and expanding the shapes contained in the input image. In the field of mathematical morphology, dilation is one of two basic operators, the other being erosion. It's most commonly used on binary images, but there are grayscale versions as well. The operator's basic impact on a binary image is to gradually extend the boundaries of foreground pixels areas (i.e. white pixels, typically). As a result, foreground pixel areas grow in size, while gaps within those regions shrink. Two bits of data are fed into the dilation operator. The image that will be dilated is the first. The second is a structural element, which is a (typically tiny) group of coordinate points (also known as a kernel). The precise effect of the dilation on the input image is determined by this structural element.

The Dilation of A by B is defined as:

$$A \oplus B = \bigcup_{b \in B} A_b, \qquad \text{.......Eq.(1)}$$

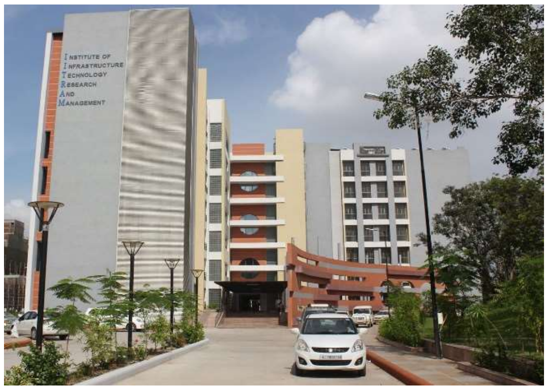


Fig.12: Original Image

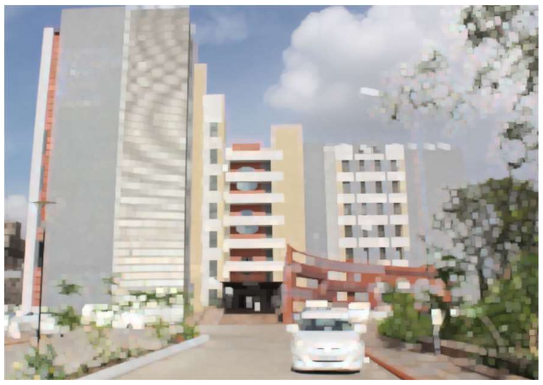


Fig.13: Dilated Image

2.10 Some More Morphological Operations

Morphological image processing is a set of non-linear processes that deal with the shape or morphology of picture features. Morphological operations are well suited to the processing of binary images since they rely solely on the relative ordering of pixel values rather than their numerical values. Greyscale images can also be subjected to morphological treatments in which the light transfer functions are unknown and the absolute pixel values are of no or small importance.

Erosion: Erosion is one of two fundamental operations in morphological image processing (the other being dilation), on which all other morphological operations are built. It was first defined for binary images, but it was later expanded to grayscale images, and then to entire lattices. A structuring element is frequently used in the erosion procedure to probe and reduce the forms in the input image.

Let A be a binary image in E and E be a Euclidean space or an integer grid. The structural element B's erosion of the binary picture A is defined as follows:

$$A \ominus B = \{z \in E | B_z \subseteq A\} \qquad \text{.......Eq.(2)}$$

where B_z is the translation of B by the vector z

Boundary Extraction:

The boundary of a set A, denoted by $\beta(A)$ can be obtained by first eroding A by B and then performing the set difference between A and its erosion.

That is,

$$\beta(A) = A - (A \ominus B)$$

Where B is a suitable structuring element

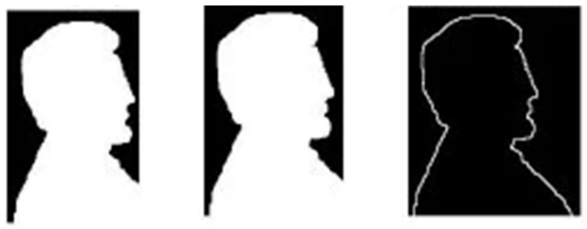


Fig.14: Stages of boundary extraction

Hole filling

A hole can be defined as a background region with a connected border of foreground pixels around it. Let A denote a set with eight connected bounds, each of which encloses a background region. The goal is to fill all of the holes with 1s, starting with a point in each hole.

The method is started by forming an array, X_0, of 0s(the same size as the array containing A) except at the location in X_0 corresponding to the given point in each hole, which is set to 0. Then the following procedure fills all the holes with 1s:

$$X_k = (X_{k-1} \oplus B) \cap A^c \qquad k=1,2,3,.... \qquad \text{.......Eq.(3)}$$

where B is the symmetric structuring element. The algorithm terminates at iteration step k if $X_k = X_{k-1}$. The set X_k then contains all the filled holes. The set union of X_k and A contains all the filled holes and their boundaries.

Extraction of connected components:

Extraction of connected components from a binary image is central to many automated image analysis applications. Let A be a set containing one or more connected components, and form an array X_0 (of the same size as the array containing A) whose elements are 0s (background values), except at each location known to correspond to a point at each connected component in A, which is set to 1 (foreground value). The objective is to start with X_0 and find all the connected components. The following iterative procedure accomplishes this objective:

$$X_k = (X_{k-1} \oplus B) \cap A \qquad k=1,2,3,\ldots \qquad \text{.......Eq.(4)}$$

where B is a suitable structuring element. The procedure terminates when $X_k = X_{k-1}$, with X_k containing all the connected components of the input image.

2.11 Canny Edge Detection

A Canny edge detector is a multi-step technique that detects edges in any input image. It entails following the methods listed below while detecting edges in an image:

- Using a Gaussian filter, noise in the input image is removed
- Calculating the gradient of picture pixels using the derivative of a Gaussian filter to obtain magnitude along the x and y dimensions
- Suppress the non-max edge contributor pixel points when considering a group of neighbours for any curve in a direction perpendicular to the provided edge
- Finally, utilise the Hysteresis Thresholding method to keep pixels that are larger than the gradient magnitude and ignore those that are smaller

<u>Steps to be followed for canny edge detection</u>

Noise removal or image smoothening: During the presence of noise, the pixel may not be comparable to its neighboring pixels. As a result, edges may be detected incorrectly or inappropriately. To avoid this, we employ a Gaussian filter, which is convolved with the image and removes noise, preventing the desired edges from appearing in the output images.

$$S = I * g(x,y) = g(x,y) * I \qquad \text{.......Eq.(5)}$$

$$g(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{x^2+y^2}{2\sigma^2}} \qquad \text{.......Eq.(6)}$$

g(x,y) = Gaussian distribution

I = Input Image

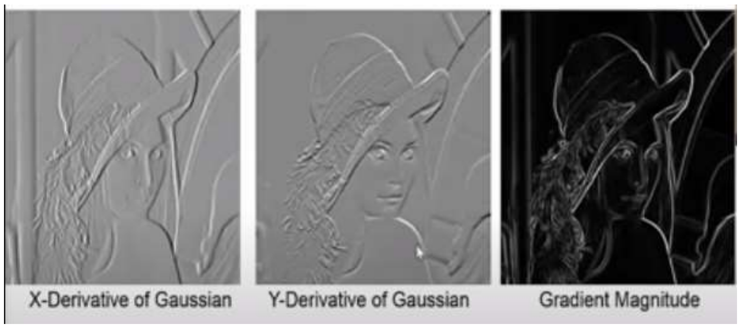


Fig.15: Stages of Canny Edge Detection

Derivative: Calculate the filter's derivative in the X and Y dimensions, then convolve it with I to get the gradient magnitude across the dimensions. The tangent of the angle between the two dimensions can also be used to compute the image's direction.

Below is an example of Gaussian Derivatives, which finally contribute to edges in output images

Non-Max Suppression: Few spots along an edge are often noted to enhance the visibility of the edge clearer. As a result, we can ignore those edge locations that don't contribute much to feature visibility. We employ the Non-Maximum Suppression approach to achieve this. The spots on the curve of the edge where the magnitude is greatest are marked here. This can be found by looking for a maximum and a slice that is perpendicular to the curve.

Hysteresis Thresholding:

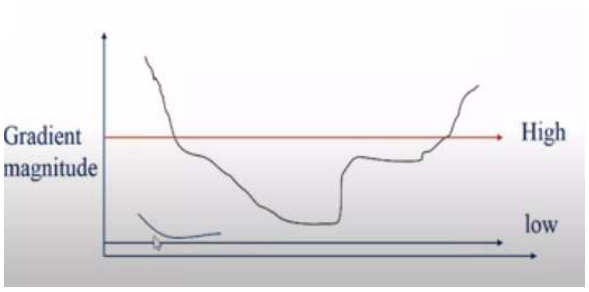


Fig.16: Hysteresis thresholding (Graphical Interpretation)

If the gradient at a pixel is :

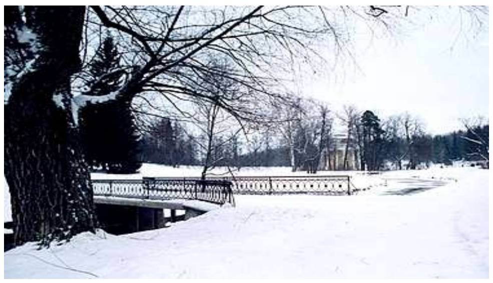
-Above “High” declare it as an ‘edge pixel.’

-Below, “Low” declares it as a ‘non-edge pixel.’

-Between “low” and “high: consider its neighbours iteratively then declare it an “edge pixel” if it’s connected to an “edge pixel” or via pixels between “low” and “high.

2.12 Thresholding

Thresholding is the most basic method of segmenting images in digital image processing. Thresholding can be used to create binary pictures from a grayscale image. If the image intensity is less than a fixed number called the threshold T, the simplest thresholding algorithms replace each pixel in an image with a black pixel, or a white pixel if the pixel intensity is greater than that threshold.


(a)

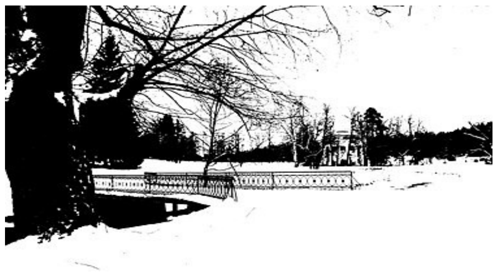

(b)

Fig.17,18 Original Image (a) and Binarized Image (b)

Automatic Thresholding: While the user can or should select the threshold T explicitly in some circumstances, there are numerous cases where the user wishes the threshold to be established automatically by an algorithm. In those circumstances, the threshold should be the "best" threshold, separating the brighter things thought to be in the foreground from the darker objects deemed to be in the background into two classes. There are several different types of automatic thresholding systems, with Otsu's method being the most well-known and commonly utilised.
The following list divides thresholding algorithms into broad categories based on the data that the algorithm manipulates. However, such a classification is inherently ambiguous, as certain methods might fit into many categories (for example, Otsu's method can be classified as both a histogram-shape and a clustering algorithm)

Histogram shape-based methods: where the smoothed histogram's peaks, valleys, and curvatures are examined. It's worth noting that these algorithms, more than others, require assumptions regarding the probability distribution of image intensity (i.e., the shape of the histogram)

Clustering-based methods: The gray-level samples are divided into two groups: background and foreground

Entropy-based methods: The entropy of the foreground and background regions, as well as the cross-entropy between the original and binarized image, are used in algorithms

Object attribute-based methods: Find a similarity metric between the gray-level and binarized images, such as fuzzy shape similarity, edge coincidence, and so on

Spatial methods: Use a higher-order probability distribution and/or pixel correlation

Global vs Local thresholding: Most approaches apply the same threshold to every pixel in an image. However, depending on the local information of the pixels, it may be desirable in some situations to apply a different threshold to different portions of the image. Local or adaptive thresholding is the name for this type of approach. They're especially well-suited to photographs with inhomogeneous illumination. A user-specified neighbourhood is defined in these cases, and a threshold is determined for each pixel and its vicinity.

Otsu's method for Image thresholding

The Otsu method is a variance-based strategy for determining the threshold value with the least weighted variance between foreground and background pixels. The main concept is to go through all of the potential threshold settings and measure the spread of background and foreground pixels. Then figure out where the spread is the smallest

The procedure iterates until it finds a threshold that minimises the within-class variance, which is defined as the weighted sum of the two classes' variances (background and foreground). Grayscale hues are usually between 0-255. (0-1 in case of float). If we set a threshold of 100, all pixels with values less than 100 become the image's background, while all pixels with values greater than or equal to 100 form the image's foreground

The formula for finding the within-class variance at any threshold *t* is given by:

$$\sigma^2(t) = \omega_{bg}(t)\sigma_{bg}^2(t) + \omega_{fg}(t)\sigma_{fg}^2(t)$$ …….Eq.(7)

where $\omega_{bg}(t)$ and $\omega_{fg}(t)$ represents the probability of number of pixels for each class at threshold t and σ2 represents the variance of colour values.

$P\_bg\ (t)$ is the count of background pixels at threshold t, $P_{fg}(t)$ is the count of foreground pixels at threshold t, and P_{all} is the total number of pixels in an image

So the weights are given by:

$$\omega_{bg}(t) = \frac{P_{bg}(t)}{P_{all}}$$ …….Eq.(8)

$$\omega_{fg}(t) = \frac{P_{fg}(t)}{P_{all}}$$ …….Eq.(9)

The variance can be calculated using the below formula:

$$\sigma^2(t) = \frac{\Sigma(x_i - x)^2}{N-1}$$ …….Eq.(10)

Where, x_i is the value of pixel at in in group (bg or fg)

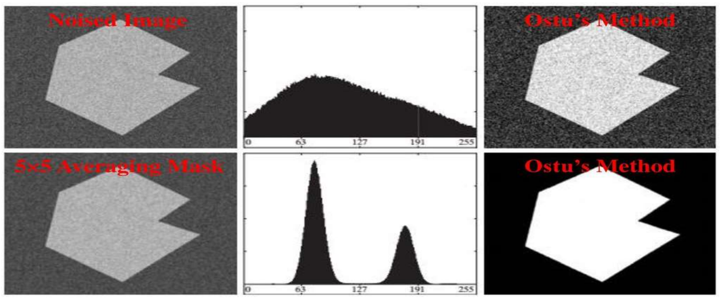
X is the value of pixel values in the group (bg or fg)

N is the number of pixels

Image smoothening to Improve threshold

If there is noise in the image, the histogram's modality will change. The steep dips between the bimodal histogram's peaks begin to deteriorate. The otsu's approach, or any other global thresholding method, will fail in that circumstance. To obtain the global threshold, reduce the noise using any smoothening filters such as Gaussian, etc., and then use any automatic thresholding method such as otsu, etc

(a) (b) (c)



(d) (f)

(e)

Fig.19: Threshold Improvement via Image Smoothening

Fig(a) is the noisy image and fig(b) is its histogram, fig(c) is the image thresholded using Otsu's method.Every black point in the white region and every white point in the black region is a thresholding error, so the segmentation was highly unsuccessful. Fig(d) shows the result of smoothing the noisy image with an average mask of size 5*5 and fig(e) is its histogram. The improvement in the shape of the histogram due to smoothing is evident, and thresholding of the smoothed image would be nearly perfect. As, it can be observed in the fig(f). The slight distortion of the boundary between the object and background in the segmented, smoothed image was caused by the blurring of the boundary.

Using edges to improve global thresholding

If the histogram peaks are tall, narrow, symmetric, and separated by deep valleys, the chances of selecting a "good" threshold are much improved. Considering only those pixels that lie on or near the boundaries between objects and the background is one way to improve the form of the histogram. The histogram would be less dependent on the relative sizes of objects and the background, which would be an instant and noticeable improvement. Because of the high concentration of one type of pixel, the histogram of an image made of a small object on a big background area (or vice versa) would be dominated by a massive peak.

If only the pixels on or near the edges between objects and background were used, the resulting histogram would have peaks of approximately the same height. In addition, the probability that any of those pixels on an object would be approximately equal to the probability that it lies on the background, thus improving the symmetry of the histogram modes. Finally, using pixels that satisfy some simple measures based on gradient and Laplacian operators has a tendency to deepen the valley between histogram peaks.

The approach discussed assumes that the edges between objects and background are known. This information clearly is not available during segmentation, as finding a division between objects and background is precisely what segmentation is all about. An indication of whether a pixel is on an edge may be obtained by computing its gradient or Laplacian. For example, the average value of the Laplacian is 0 at the transition of an edge, so the valleys of histograms formed from the pixels selected by a Laplacian criterion can be expected to be sparsely populated. This property tends to produce the desirable deep valleys.

Summarized Algorithm

1. Compute an edge image as either the magnitude of the gradient or absolute value of the Laplacian of F(x,y)
2. Specify a threshold value
3. Threshold the image from step 1 using the threshold from step 2 to produce a binary image, $g_T(x,y)$. This image is used as a mask image in the following step to select pixel from f(x,y) corresponding to "strong" edge pixels.
4. Compute a histogram using only the pixels in f(x,y) that correspond to the locations of the 1- valued pixels in $g_T(x,y)$.
5. Use the histogram from step 4 to segment f(x,y) globally using, for example, Otsu's method.

If T is set to a value less than the edge image's minimum value, $g_T(x,y)$ will be all 1s, meaning that the image histogram will be computed using all pixels of f(x,y). In this scenario, the preceding procedure is renamed global thresholding, and the original image's histogram is employed. It is common practise to supply a percentile value for T, which is often set high so that only a few pixels from the gradient/Laplacian image are used in the computation. We can improve cell region segmentation by changing the percentile at which the threshold is set.

Variable Threshold

Image Partitioning: One of the simplest approaches to variable thresholding is to subdivide an image into nonoverlapping rectangles. This approach is used to compensate for non- uniformities in illumination and/or reflectance. The rectangle are chosen small enough so that the illumination of each is approximately uniform.

Image subdivision generally works well when the objects of interest and the background occupy regions of reasonably comparable size. When this is not the case, the method typically fails because of the likelihood of subdivisions containing only object or background pixels. Although this situation can be addressed by using additional techniques to determine when a subdivision contains both types of pixels the logic required to address different scenarios can get complicated.

Variable thresholding based on local image properties: A more broad approach than image subdivision is to compute a threshold at each point(x,y) in the image depending on one or more specified properties determined in the neighbourhood of the image (x,y). Despite the fact that this may appear to be a time-consuming process, modern algorithms and hardware enable quick neighbourhood processing, particularly for simple functions like logical and arithmetic operations.

The basic approach to local thresholding is using the standard deviation and mean of the pixels in a neighbourhood of every point in an image. These two quantities are quite useful for determining local thresholds because they are descriptors of local contrast and average intensity. Let σ_{xy} and m_{xy} denote the standard deviation and mean value of the set of pixels contains in a neighbourhood,

…….Eq.(11)

S_{xy}, centred at coordinates (x,y) in an image . The following are common forms of variable, local thresholds:

$$T_{xy} = a\,\sigma_{xy} + b\,m_{xy}$$

where a and b are nonnegative constants, and

…….Eq.(12)

$$T_{xy} = a\,\sigma_{xy} + b\,m_G$$

where m_G is the global image mean. The segmented image is computes as:

$$g(x,y) = \begin{cases} 1 & , \quad f(x,y) > T_{xy} \\ 0 & , \quad f(x,y) \leq T_{xy} \end{cases}$$ …….Eq.(13)

where f(x,y) is the input image. This equation is evaluated for all pixel location in the image, and a different threshold is computed at each location (x,y) using the pixels in the neighbourhood S_{xy}. Significant power (with a modest increase in computation) can be added to local thresholding by using predicates based on parameters computed in the neighbourhood of (x,y):

$$g(x,y) = \begin{cases} 1 & , \text{ if Q(local parameters) is true} \\ 0 & , \text{if Q(local parameters) is false} \end{cases}$$ …….Eq.(14)

where Q is a predicate based on parameters computed using the pixels in neighbourhood S_{xy}.

2.13 Edge Linking and Boundary Detection

Edge detection systems should ideally provide pixels that are exclusively on the borders of regions. Because of noise, breaks in the border owing to non-uniform illumination and other phenomena that generate spurious discontinuities, in practise, this pixel set rarely completely characterises a boundary. As a result, edge detection methods are frequently followed by linking and other boundary detection procedures that combine edge pixels into meaningful boundaries.

Local processing:

Basic idea:

Analyse the features of pixels in a small neighbourhood (3x3, 5x5, etc.) for each edge-detected location (x,y)

All "similar" points are linked together, establishing a boundary of pixels with some common attribute.

2 principal properties for establishing similarity:

The strength of the response of the gradient operator used to produce the edge pixels

The direction of the gradient

$$\alpha(x,y) = \tan^{-1}\frac{G_y}{G_x} \qquad \text{.......Eq.(15)}$$

The magnitude of an edge pixel at (x',y') in the neighborhood centred at (x,y) is similar to the pixel at (x,y) if

$$|\nabla f(x,y) - \nabla \mathrm{f}(\mathrm{x}',\mathrm{y}')| \leq T \qquad \text{.......Eq.(16)}$$

Where T is predetermined threshold

The angle of an edge pixel at (x',y') in the neighborhood centred at (x,y) is similar to the pixel at (x,y) if

$$|\alpha(x,y) - \alpha(\mathrm{x}',\mathrm{y}')| \leq A \qquad \text{.......Eq.(17)}$$

Where A is predetermined angle threshold

If both the magnitude and angle conditions are met, a point in the vicinity of (x,y) is linked to (x,y)

Global processing: Hough transform

- Consider the case where we want to locate all subsets of n points in an image that lie on straight lines

- Consider a point (x_i,y_i) and the equation for a straight line y_i=ax_i+b
- For various values of a and b, an infinite number of lines pass through the point (x_i,y_i), all satisfying the equation
- When the equation is written as b=-x_ia + y_i and the ab plane (also known as the parameter space) is taken into account, the equation of a single line for a fixed point is obtained (x_i,y_i)
- A second point (x_j, y_j) also has a line in the parameter space associated with it.

This line intersects the line associated with (x_i,y_i) at (a' ,b')

a'is the slope and b' is the intercept of the line containing both (x_i,y_i) and (x_j, y_j) in the xy plane

In fact, all points that lie on this line have corresponding lines in the parameter space that intersect at (a',b')

The computational attractiveness of the Hough transform arises from the subdivision of the parameter space into accumulator cells

(a_{min},a_{max}) and (b_{min},b_{max}) are expected ranges of slope and intercept values

The cell at coordinates (i,j), with cell value A(i,j), corresponds to the square associated with parameter space coordinates (ai,bj)

The collection of accumulator cells is commonly called the Hough matrix (or Hough array) and are computed as

1. Set all cells to zero.

2. For every point (x_k,y_k) in the image plane, let a equal each of the allowed subdivision values on the a axis and solve for busing b=-ax_i+y_i 3. The resulting b's are rounded off to the nearest allowed value in the b axis

4. If a choice of a_p results in solution b_q, we let A(p,q)=A(p,q)+1

5. At the end of the procedure, a value of M in A(i,j) corresponds to M points in the xy plane lying on the line y=-xa_i+b_j

2.14 Noise

In digital images, noise conveys undesired information. Noise causes artefacts, unrealistic edges, unseen lines, corners, blurred objects, and affects background scenery, among other things. In digital images, the main causes of noise occur during image acquisition and/or transmission. Imaging sensor performance is influenced by a number of factors, including the environment during images acquisition and the quality of the sensing elements themselves. For example, when using a CCD camera to acquire images, light levels and sensor temperature are important factors in determining the amount of noise in the final image. During transmission, images are corrupted mostly due to interference in the transmission channel.

Gaussian Noise

It is also called as electronic noise because it arises in amplifiers or detectors . Natural causes of Gaussian noise include atom thermal vibrations and the discontinuous character of heated object radiation. The grey values in digital photographs are typically disturbed by Gaussian noise. As a result, the Gaussian noise model is primarily defined by its PDF, which normalises the histogram with regard to grey value.

Rayleigh noise

In radar range images, Rayleigh noise can be seen. The probability density function for Rayleigh noise is given as

$$P(g) = \begin{cases} \frac{2}{b}(g-a)e^{\frac{-(g-a)^2}{b}}, for & g \geq a \\ 0 \quad , & g < a \end{cases}$$ …….Eq.(18)

Where mean $\mu = a + \sqrt{\frac{\pi b}{4}}$ and variance $\sigma^2 = \frac{b(4-\pi)}{4}$

Gamma Noise

Gamma noise is generally seen in the laser-based images. It obeys the Gamma distribution.

$$P(g) = \begin{cases} \dfrac{a^b g^{b-1} e^{-a}}{(b-1)!}, for & g \geq 0 \\ 0 \qquad\qquad , & g < 0 \end{cases} \qquad \text{…….Eq.(19)}$$

Where mean $\mu = \frac{b}{a}$ and variance $\sigma^2 = \frac{b}{a^2}$

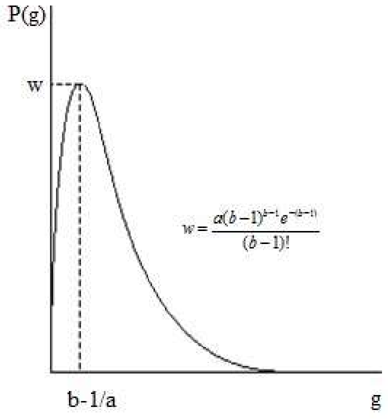


Fig.20 P(g) vs g plot for gamma noise

Impulse Valued Noise (Salt and Pepper Noise)

This is also known as data drop noise since it reduces the original data values statistically. Salt and pepper noise is another name for this phenomenon. However, the image is not completely destroyed by salt and pepper noise; rather, some pixel values in the image are altered. Although there is a possibility of some neighbours not changing in a noisy image. This noise is seen in data transmission. Image pixel values are replaced by corrupted pixel values either maximum ‘or’ minimum pixel value i.e., 255 ‘or’ 0 respectively, if number of bits are 8 for transmission.

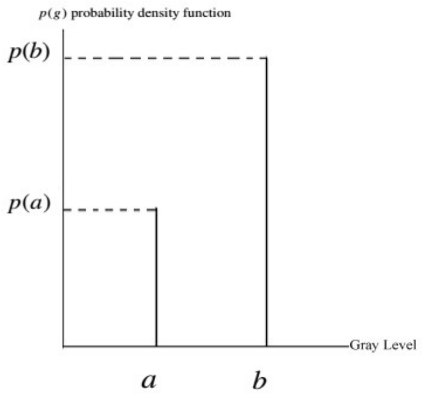


Fig.21: P(g) vs g plot for Salt and Pepper noise

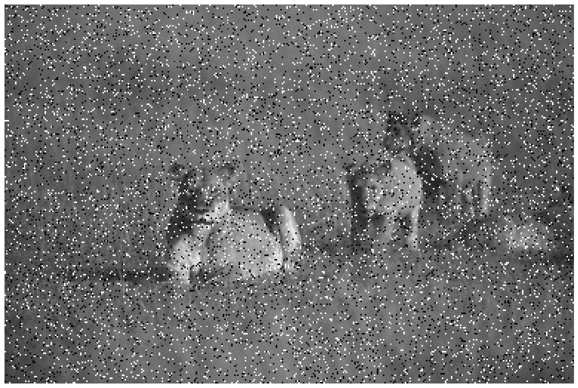


Fig.22: 10% Salt and Pepper Noise

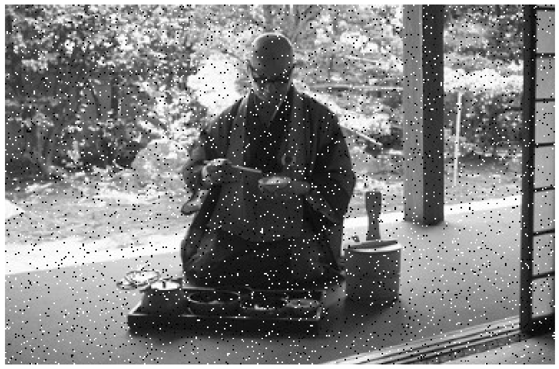


Fig.23: 5% Salt and Pepper Noise

Periodic Noise

Electronic interferences, particularly in the power signal during image acquisition, cause this noise. At multiples of a given frequency, this noise exhibits unique properties such as being spatially dependent and sinusoidal in nature. In the frequency domain, it appears as conjugate spots. A narrow band reject filter or notch filter can be used to easily eliminate it.

2.15 Contour Detection

A contour line indicates a curved line representing the boundary of the same values or the same intensities. Contours are different from Edges. Edges are points whose values change significantly compared to their neighboring points. Contours, on the other hand, are closed curves which are obtained from edges and depicting a boundary of figures. Processed image is fed to contour detection. For contour detection in OpenCV, we use the following functions:

1. findContours()
2. drawContours()

Also, it has two different algorithms for contour detection:

1. CHAIN_APPROX_SIMPLE
2. CHAIN_APPROX_NONE

CHAIN_APPROX_SIMPLE And CHAIN_APPROX_NONE have some rudimentary differences. CHAIN_APPROX_SIMPLE removes all redundant points and compresses the contour, thereby saving memory. The CHAIN_APPROX_SIMPLE algorithm compresses horizontal, vertical, and diagonal segments along the contour and leaves only their end points. This means that any of the points along the straight paths will be dismissed, and we will be left with only the end points. For example, consider a contour, along a rectangle. All the contour points, except the four corner points will be dismissed. This method is faster than the CHAIN_APPROX_NONE because the algorithm does not store all the points, uses less memory, and therefore, takes less time to execute.

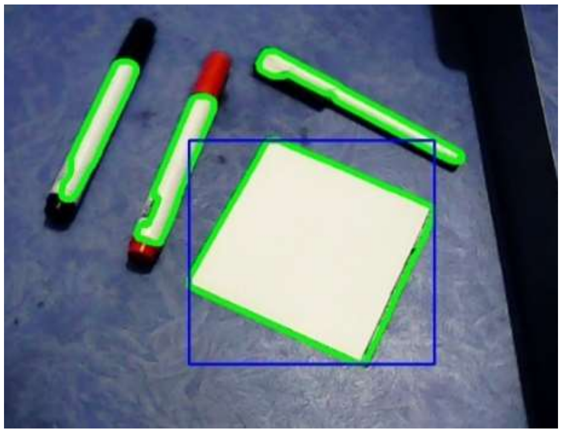


Fig.24: Detected and drawn: Contours(Green), Bounding Contour Box(Blue)

In other words When we join all the points on the boundary of an object, we get a contour. Typically, a specific contour refers to boundary pixels that have the same colour and intensity.

Once contours have been identified, we will overlay the contours on the original RGB image.

Fig.25: Original Image

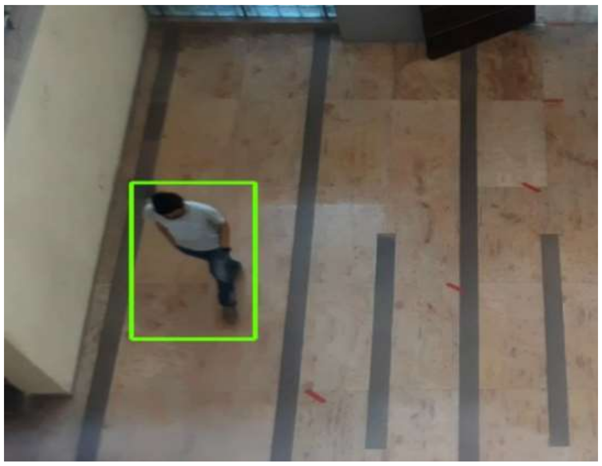


Fig.26: Original(Left) and Contour Box drawn on the moving object(Right)

Chapter 3

Implementation and Methodology

3.1 Software Implementation

PyFirmata

In order to abridge the communication gap between the Arduino Microcontroller and OpenCV python code, pyFirmata was employed. pyFirmata is a python-based library which works on the firmata protocol. Firmata is a protocol for communicating with microcontrollers from software on a computer (or smartphone/tablet, etc). The Firmata protocol could theoretically be implemented for any microcontroller platform. Currently however, the most complete implementation is for Arduino (including Arduino-compatible microcontrollers). Firmata is based on the midi message format in that commands bytes are 8 bits and data bytes are 7 bits. For example the midi Channel Pressure (Command: 0xD0) message is 2 bytes long, in Firmata the Command 0xD0 is used to enable reporting for a digital port (collection of 8 pins). Both the midi and Firmata versions are 2 bytes long, but the meaning is obviously different. In Firmata, the number of bytes in a message must conform with the corresponding midi message. Midi System Exclusive (Sysex) messages, however, can be any length and are therefore used most prominently throughout the Firmata protocol.

There are two main models of usage of Firmata. In one model, the author of the Arduino sketch uses the various methods provided by the Firmata library to selectively send and receive data between the Arduino device and the software running on the host computer. For example, a user can send analog data to the host using Firmata.sendAnalog(analogPin, analogRead(analogPin) or send data packed in a string using Firmata.sendString(stringToSend).

The second and more common model is to load a general purpose sketch called StandardFirmata (or one of the variants such as StandardFirmataPlus or StandardFirmataEthernet depending on the application) on the Arduino board and then use the host computer exclusively to interact with the Arduino board. The latter model of usage has been used in this project.

pyFirmata solves a crucial problem since it allows the user to access and control Arduino from python language console itself, hence eliminating the need to program on Arduino separately.

OpenCV

Open CV or Open-Source Computer Vision is the pivotal component of this project. It's a library of programming functions which can be utilized to perform operations pertaining to computer vision. 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. Its most popular applications involve Object detection, 2D and 3D feature toolkits, Egomotion estimation, Facial recognition, system Gesture recognition, Human–computer interaction (HCI), Mobile robotics, Segmentation and recognition, Stereopsis stereo vision: depth perception from 2 cameras, Structure from motion (SFM), Motion tracking and Augmented reality. OpenCV was originally developed in C++ Language but extends to other popular programming languages like Python, Java, MATLAB/Octave as well. OpenCV mainly emphasises on providing a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The following operations can be performed using the OpenCV library –

1. Read and write images
2. Capture and save videos
3. Process images (filter, transform)
4. Perform feature detection
5. Detect specific objects such as faces, eyes, cars, in the videos or images.
6. Video Analysis, i.e., estimate the motion in it, subtract the background, and track objects in it.

   Other Libraries Used: Numpy, Matplotlib, PyFirmata.

3.2 Hardware Implementation

Relays

The electronic relay is a type of an electronic switch that opens or close the circuit contacts by using electronic component without any mechanical operation. The absence of mechanical operation allows faster switching as compared to conventional switching apparatus. It consists of a set of input terminals for a single or multiple control signals, and a set of operating contact terminals. The switch may have any number of contacts in multiple contact forms, such as make contacts, break contacts, or combinations thereof. Relays are used where it is necessary to control a circuit by an independent low-power signal, or where several circuits must be controlled by one signal.

The traditional form of a relay uses an electromagnet to close or open the contacts, but other operating principles have been

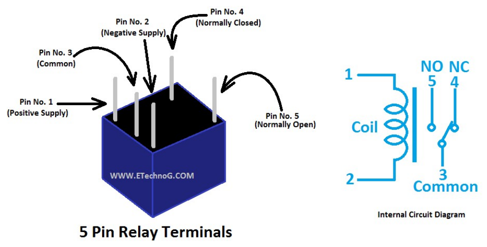

Pin No. 4 (Normally Closed)
Pin No. 2 (Negative Supply)
Pin No. 3 (Common)
Pin No. 1 (Positive Supply)
Pin No. 5 (Normally Open)
WWW.ETechnoG.COM
5 Pin Relay Terminals
1
2
Coil
NO NC
5 4
3
Common
Internal Circuit Diagram

Fig.27: Schematic of a Relay

invented, such as in solid-state relays which use semiconductor properties for control without relying on moving parts. Relays with calibrated operating characteristics and sometimes multiple operating coils are used to protect electrical circuits from overload or faults; in modern electric power systems these functions are performed by digital instruments still called protective relays.

Arduino UNO Microcontroller

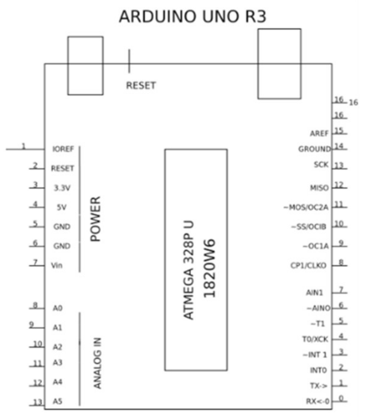


Fig.28: Arduino UNO Schematic Circuit Diagram

Arduino consists of both a physical programmable circuit board (microcontroller) and a software environment, or IDE (Integrated Development Environment) that is used to write, and upload computer code to the physical board. Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards ('shields') or breadboards (for prototyping) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs. The microcontrollers can be programmed using the C and C++ programming languages, using a standard API which is also known as the Arduino language, originated from the Processing language. In addition to using traditional compiler toolchains, the Arduino project provides an integrated development environment (IDE) and a command line tool developed in Go. The 14 digital input/output pins can be used as input or output pins by using pinMode(), digitalRead() and digitalWrite() functions in arduino programming. Each pin operates at 5V and can provide or receive a maximum of 40mA current, and has an internal pull-up resistor of 20-50 KOhms which are disconnected by default.

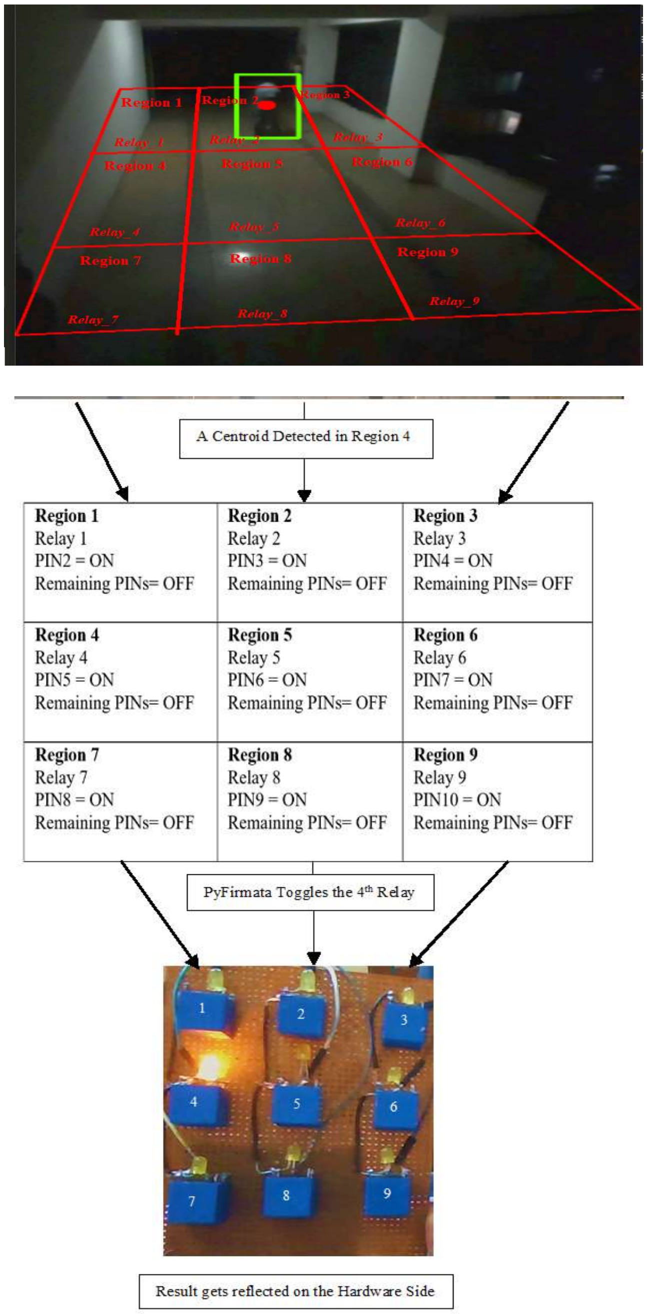
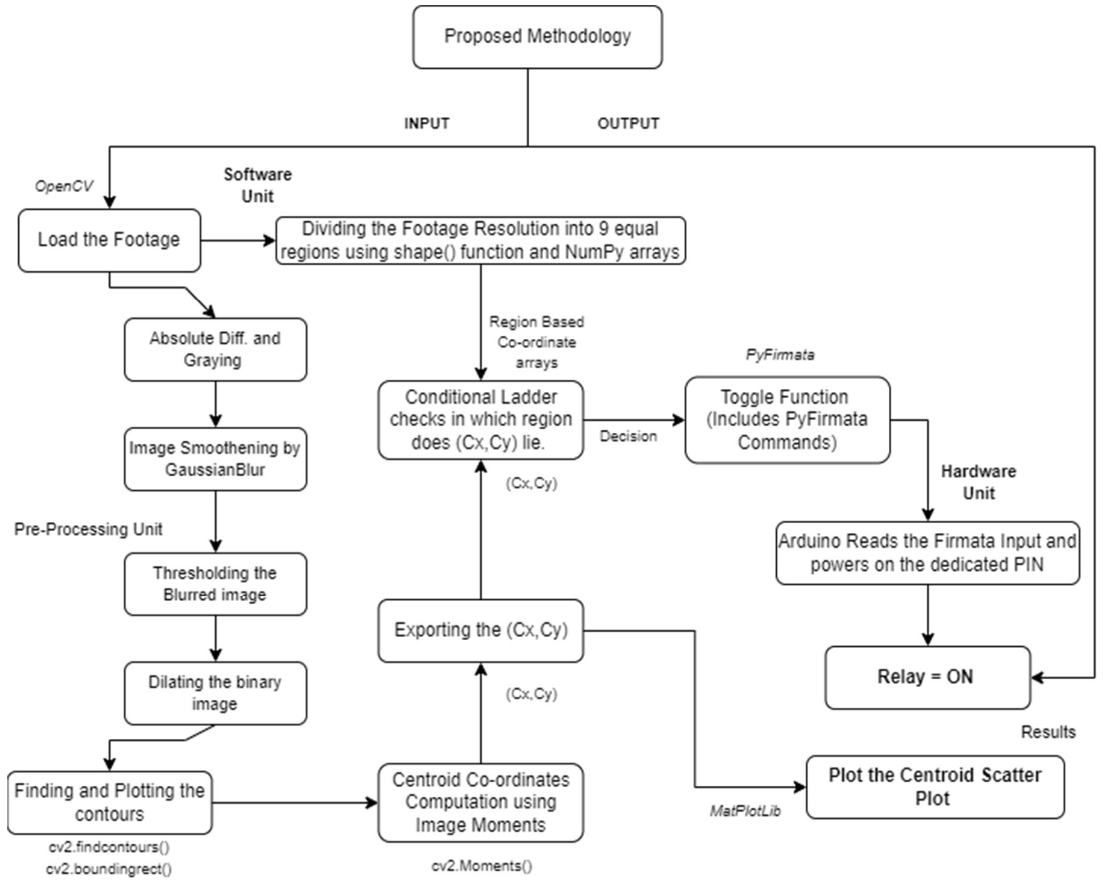

Region 1
Region 2
Region 3
Relay_1
Relay_2
Relay_3
Region 4
Region 5
Region 6
Relay_4
Relay_5
Relay_6
Region 7
Region 8
Region 9
Relay_7
Relay_8
Relay_9
A Centroid Detected in Region 4
Region 1 Relay 1 PIN2 = ON Remaining PINs= OFF
Region 2 Relay 2 PIN3 = ON Remaining PINs= OFF
Region 3 Relay 3 PIN4 = ON Remaining PINs= OFF
Region 4 Relay 4 PIN5 = ON Remaining PINs= OFF
Region 5 Relay 5 PIN6 = ON Remaining PINs= OFF
Region 6 Relay 6 PIN7 = ON Remaining PINs= OFF
Region 7 Relay 7 PIN8 = ON Remaining PINs= OFF
Region 8 Relay 8 PIN9 = ON Remaining PINs= OFF
Region 9 Relay 9 PIN10 = ON Remaining PINs= OFF
PyFirmata Toggles the 4th Relay
1
2
3
4
5
6
7
8
9
Result gets reflected on the Hardware Side

Fig.29: An Illustration of entire implementation

3.3 Proposed Methodology


Proposed Methodology
INPUT
OUTPUT
OpenCV
Software Unit
Load the Footage
Dividing the Footage Resolution into 9 equal regions using shape() function and NumPy arrays
Absolute Diff. and Graying
Region Based Co-ordinate arrays
PyFirmata
Conditional Ladder checks in which region does (Cx,Cy) lie.
Decision
Toggle Function (Includes PyFirmata Commands)
Image Smoothening by GaussianBlur
(Cx,Cy)
Hardware Unit
Pre-Processing Unit
Thresholding the Blurred image
Arduino Reads the Firmata Input and powers on the dedicated PIN
Exporting the (Cx,Cy)
Relay = ON
Dilating the binary image
(Cx,Cy)
Results
Finding and Plotting the contours
Centroid Co-ordinates Computation using Image Moments
Plot the Centroid Scatter Plot
MatPlotLib
cv2.findcontours()
cv2.boundingrect()
cv2.Moments()

1. Image Acquisition

This is the most fundamental step of the project. The image is acquired through stock/offline footage (for understanding purpose) and live-CCTV camera (for real-time implementation). VideoCapture is an inbuild function of OpenCV that allows the user to load the video into the program from the dedicated address. The attributes of loaded video are stored in a variable “cap”. The argument of VideoCapture method either seeks an address or an integer denoting the connected camera, for example, 0 for the in-build webcam.

2. Processing:

Step1: ***Absolute Difference* and *Gray Scaling***

The video is loaded and, then 2 frames from the video, namely frame 1 and frame 2 are extracted. Then the difference of the two frames was taken and then the difference was converted into the grayscale. A grayscale (or graylevel) image is simply one in which the only colors are shades of gray.

cap.read() is the function used. It takes 2 variables, namely ret and frame (here). "Frame" will get the next frame in the camera (via "cap"). "Ret" will obtain return value from getting the camera frame, either true of false. The "absdiff" method takes 2 static frames; frame 1 and frame 2 and computes its absolute difference and stores it in a variable "diff". Later this variable "diff" is fed to cvtColour method with a present of COLOR_BGR2GRAY which converts a Blue-Green-Red Colour image to a Grayscale image. Further operations are done in a loop.

Step 2: **Image Smoothening by *GaussianBlurring* Operation.**

When we blur an image, we make the colour transition from one side of an edge in the image to another smooth rather than sudden. The effect is to average out rapid changes in pixel intensity. Blurring is an example of applying a *low-pass filter* to an image. In computer vision, the term "low-pass filter" applies to removing noise from an image while leaving the majority of the image intact. A blur is a very common operation we need to perform before other tasks such as edge detection. The Method GaussianBlur is used. "gray" is the source for this operation. (5,5) is the tuple denoting the kernel size for the blurring to take place. And, 0 denotes the border size.

Step 3 : ***Thresholding* the Blurred Image.**

Thresholding is a type of *image segmentation*, where we change the pixels of an image to make the image easier to analyse. In thresholding, we convert an image from colour or grayscale into a *binary image*, i.e., one that is simply black and white

Step 4: ***Dilating* the Binary Image.**

Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the *structuring element* used to process the image.

Step 5: **Plotting the Contour**

When we join all the points on the boundary of an object, we get a contour. Typically, a specific contour refers to boundary pixels that have the same colour and intensity.

Once contours have been identified, we will overlay the contours on the original RGB image.

Step 6: Generalized algorithm to divide Resolution into Trapezoidal Grids

The entire resolution was divided into a 3x3 grid. This division of x*y pixel into 9 regions resolution was carried out as below:

Masking was done of all the 9 regions . Followed by which, 4 variables were declared for denoting the boundary of each square region. Post typecasting these into integers, 6 more variables were declared as a range using the arrange method of the NumPy library. These ranges were then fed to the main conditional statements which in order to check in which of the 9 regions the centroid pair (Cx, Cy) of the contour at T=t seconds lies. As a result, the relay corresponding to that particular region was toggled. Bitwise operators were also used in the process.

Step 7: Toggle Function

Merely switching ON the dedicated relay using the conditional output is not enough. When one relay is given high pulse (ON status), the rest of the relays need to be OFF for that particular instant of time to avoid mis-triggering of the Relays. A function "toggle" was defined in order to ease this rapid operation of switching ON/OFF the various relays present in the matrix.

As it can be inferred from the above code fragment, there are 9 relays, connected from PINS2 to PIN10 of the Arduino. (i.e. PIN2 corresponds to RELAY1, PIN3 corresponds to RELAY2 and so on)

The input for the function is an integer X. This integer in this context is the relay number (The PIN at which that relay has been connected). Lets suppose, the conditional ladder determines that the centroid is detected in the 5^{th} region at the given second. This would imply that the 5^{th} relay needs to be ON, meaning PIN6 needs to go HIGH pulse (As discussed already, the relays are connected to PINS 2-10). To do so , board.digital[x].write(1) is executed, which turns on the relay corresponding to PIN X.

At the very same time, all the other Relays apart from the one pertaining to X need to be OFF. To do so, X is removed from the list of all the relays and then the board.digital[i].write(0) is iterated across the list of the remaining relays. (Where i is the iterable loop variable). This ensures that the rest of the relays remain OFF while one relay, Relay X is being provided HIGH signal (ON status).

Step 8: Moment and Centroid Computation

Centroid of the contour is the arithmetic mean of all the positions of a contour.

$$\mathbf{c} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$$ …….Eq.(20)

In the context of image processing and computer vision, each shape is made of pixels, and the centroid is simply the weighted average of all the pixels constituting the shape. In order to detect the centroid, first the image is subjected to grayscale operation followed by binary thresholding. Post this, The image moments are calculated. Image Moment is a particular weighted average of image pixel intensities, with the help of which we can find some specific properties of an image, like radius, area, centroid etc. The centroid is given by the formula: -

$$C_x = \frac{M_{10}}{M_{00}} \qquad C_y = \frac{M_{01}}{M_{00}}$$ …….Eq.(21),(22)

The knowledge of Image moments is used to compute centroids using OpenCV. Image Moment is a particular weighted average of image pixel intensities, with the help of which specific properties of an image, like radius, area, centroid etc can be determined. For a 2D continuous function *f*(*x*,*y*) the moment (sometimes called "raw moment") of order (*p* + *q*) is defined as

$$M_{pq} = \int\limits_{-\infty}^{\infty}\int\limits_{-\infty}^{\infty} x^p y^q f(x,y)\,dx\,dy$$ …….Eq.(23)

for *p*,*q* = 0,1,2,... Adapting this to scalar (greyscale) image with pixel intensities *I*(*x*,*y*), raw image moments M_{ij} are calculated

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$ …….Eq.(24)

The zeroth moment is denoted by M00 and corresponds to the area of the binary image since its nothing but the summation of every non-zero pixel present in the image. Similarly, M10 denotes the sum of all the non-zero pixels along the X-axis and M01 denotes the sum of all the non-zero pixels along the Y-axis.

C_x is the x coordinate and C_y is the y coordinate of the centroid and M denotes the Moment. Once the C_x and C_y was calculated, it was stored in dedicated tuples. Moments in image processing are average values from the single pixels` intensities of an image. With this moments physical properties like orientation, eccentricity, the area or the centroid of an object in the image can be identified. These tuples were later used to graph the centroid plot. In the results, the Contours have been denoted by green colored rectangular bounding box and its corresponding centroid has been denoted by red colored dot.

A centroid is the point which corresponds to the mean position of all the points in a figure. The centroid is the term for 2-dimensional shapes. The centre of mass is the term for 3-dimensional shapes.

For instance, the centroid of a circle and a rectangle is at the middle. In this scenario, the term centroid refers to the Arithmetic mean of all the positions of a contour. (Weighted average of all the pixels).

Step 9: Drawing the Centroid

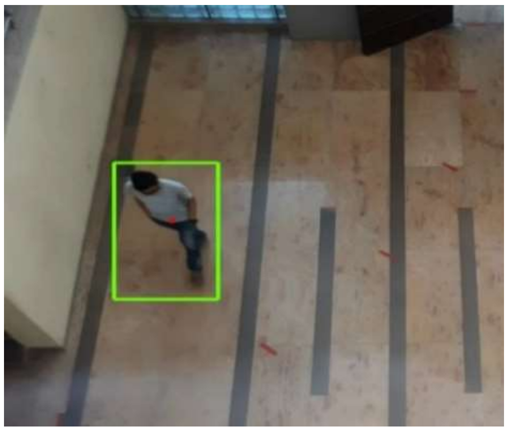


Fig.30: Centroid detected and drawn on the object

Step 10: Creating Trackbars

For hands-on live control of parameters, trackbars were introduced in the system. The goal here was to improve the overall accessibility of the project. At present, 2 Trackbars namely

Brightness control and Manual Relay Control were added but more can be added if needed.

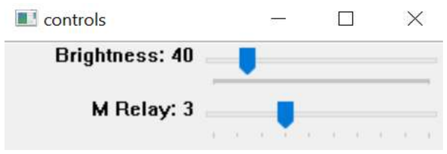


Fig.31: Trackbars

Step 11: Warp Perspective

How the warpPerspective() function in OpenCV works:
The perspectives of the photographs or videos can be aligned to extract more valuable information from the images or videos. In OpenCV, a function called getPerspective Transform() is used to adjust the perspective of pictures or movies. The getPerspective Transform() function delivers an image that provides additional information about the given image or video. The warpPerspective() function is used to fit the size of the produced image to the size of the original image or video by utilising the getPerspectiveTransform() function. The warpPerspective() function generates an image or video that is the same size as the original image or video. The following is the syntax for defining the warpPerspective() function in OpenCV:

warpPerspective(source_image, result_image, outputimage_size)

where source_image is the image whose perspective is to be changed using the getPerspectiveTransform() function
result_image is the image returned by using getPerspectiveTransform() on the source_image
outputimage_size is the size of the output image that fits the size of the original image using the warpPerspective() function

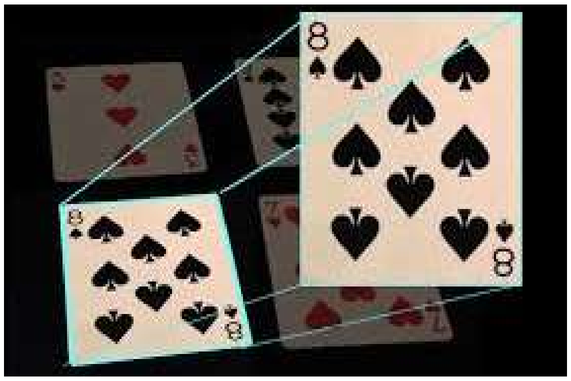


Fig.32: Warp Perspective: An illustration

Chapter 4

Challenges and Shortcomings

4.1 Results

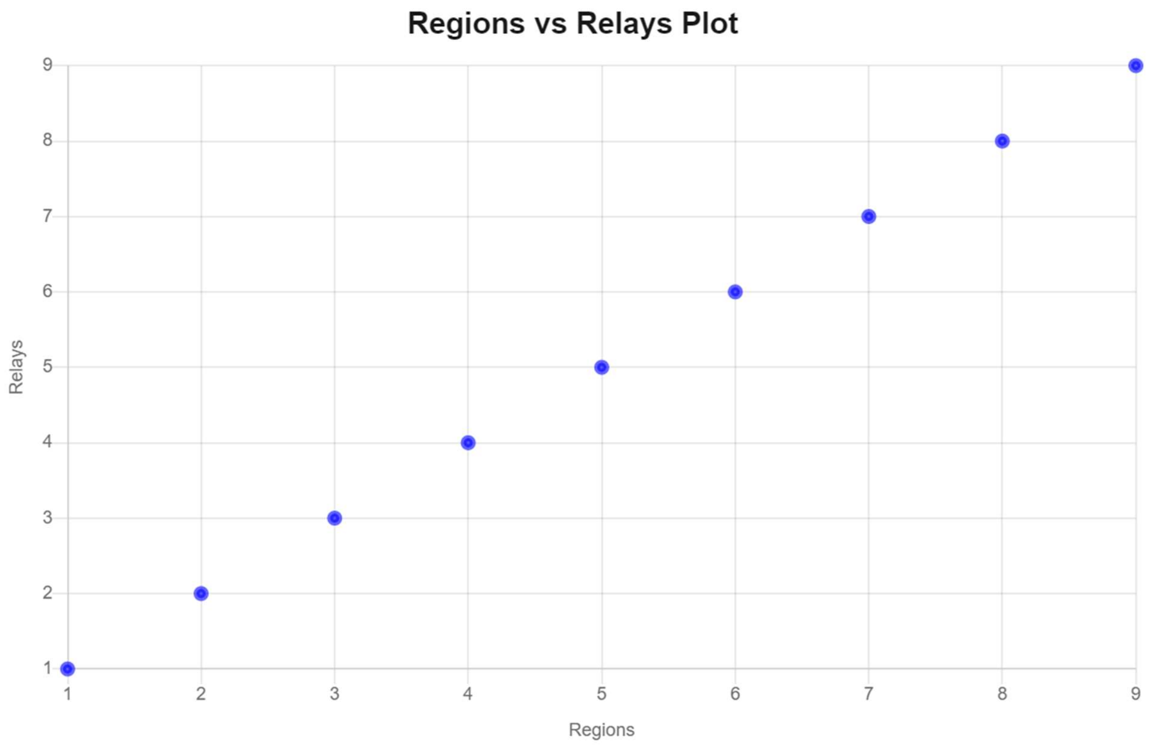


Fig 33: Regions vs Relays Plot

4.2 Conclusions

9. Precision drops marginally in low-lighting scenarios as compared to well-lit conditions.

10. The process of contour detection can be improved by improving the pre-processing unit of the project.

11. Noise or unwanted entities might cause mis-triggering. Hence its imperative to reduce them to the best extent possible.

12. Different footages have different parametric demands (i.e pre-processing parameters like kernel size, dilation iterations and so on).

4.3 Challenges Faced

The contour detection is a highly subjective process. It depends on numerous factors. Many times, the source CCTV footage quality is poor which might contain motion blur. This might leave a so called “Tailing effect” after an object in motion, which may hinder the process of object detection. Another crucial factor is the lighting condition or the availability of light in the footage. Due to dearth of light, the algorithm might face trouble in deciphering between the pixels pertaining to the background and that of the moving object. Low light scenarios often incur noises which might cause faulty detection of contour and hence might lead to mis-triggering of the lights. This also leads to the “Blinking” or “Flickering” of the bounding box of the detected contour. To mitigate this issue, we need to make sure that the contour detection is immune to the changes in light as much as possible. One possible way to do so is by changing the parameters of the image enhancement operations used such as graying, blurring and thresholding. For example, median blur with kernel size of 3 works best for salt and pepper noise. But this would, at best, cause a microscopic enhancement in the contour detection accuracy. The most optimum way is to use advanced algorithms like Fast R-CNN (Region based convolution neural network), Histogram of oriented gradients (HOG), Spatial Pyramid Pooling (SSP- net) and YOLO.

Appendix

Summary

The Smart Surveillance Parking Camera is a niche idea that is expected to save energy by many folds. Not only does it involve simple execution and infrastructure, but also involves lesser components than the conventional systems of illumination control. Fetching the footage directly from the CCTV camera eliminates the need of having multiple sensors connected in an array and this is what makes it unique from the rest of the pre-existing illumination control techniques. Once the footage is fetched, it is fed directly to the image processing block/stage. Image processing is the most essential stage of this project. The live video footage is parsed frame by frame and various image operations are implemented on each frame with the help of OpenCV commands. For efficient contour detection, the frame is subjected to operations like blurring, dilation,thresholding,graying and absolute difference followed by which contours are detected. Image moments were utilized to calculate the centroid of the detected contour(s). The footage is divided into a grid using NumPy arrays which were later used to compare/verify where centroid co-ordinates lie. (i.e Which region is the object currently present in). This is done using a robust conditional ladder which involved masking via bitwise operator. The output is obtained via this decision-making ladder is an attribute. The processed output is not sufficient to control the lights independently. To abridge this gap of hardware and software, a microcontroller is used as an interface. PyFirmata acts as an excellent communicator between Python and Arduino. The light pertaining to the object's current location is then turned ON by the microcontroller as per the intimation received through the OpenCV programme.

References

[1] Gupta, Bhumika & Chaube, Ashish & Negi, Ashish & Goel, Umang. (2017). Study on Object Detection using Open CV - Python. International Journal of Computer Applications. 162. 17-21. 10.5120/ijca2017913391.

[2] S.V, Viraktamath, Mukund Katti, Aditya Khatawkar and Pavan Kulkarni. "Face Detection and Tracking using OpenCV." (2016).Guennouni, Souhail & Ali, Ahaitouf & Mansouri, Anass. (2015). Multiple Object Detection using OpenCV on an Embedded Platform. 2015. 374-377. 10.1109/CIST.2014.7016649.

[3] Turkay, Mustafa. (2017). OBJECT DETECTION AND TRACKING FOR REAL TIME FIELD SURVELLIANCE APPLICATIONS. 10.13140/RG.2.2.34716.49282

[4] S., Suthagar & Ponmalar, Augustina & Benita, & Banupriya, & Beulah,. (2016). SMART SURVEILLANCE CAMERA USING RASPBERRY PI 2 AND OPENCV. International Journal of Advanced Research Trends in Engineering and Technology (IJARTET). 3. 177-181.

[5] Shabbir Bhusari, Sumit Patil, Mandar Kalbhor. ―Traffic Control System using raspberry-pi,‖ International Journal Of Advanced Engineering Technologies. [2] Rajeshwar Kumar Dewangan, Yamini Chouhan. A Review on Object Detection using Open CV Method. International Research Journal of Engineering and Technology (IRJET)

[6] G. Chandan, A. Jain, H. Jain and Mohana, "Real Time Object Detection and Tracking Using Deep Learning and OpenCV," 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), 2018, pp. 1305-1308, doi: 10.1109/ICIRCA.2018.8597266.

[7] A. Sharma, J. Pathak, M. Prakash and J. N. Singh, "Object Detection using OpenCV and Python," 2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), 2021, pp. 501-505, doi: 10.1109/ICAC3N53548.2021.9725638.

[8] Khan, Farid & Khattak, Atal & Masood, Ahmad. (2021). Arduino Based Control And Data Acquisition System Using Python Graphical User Interface (GUI). International Journal of Scientific & Technology Research. 10.

[9] Gehlot, Anita & Singh, Rajesh & Gupta, Lovi & Singh, Bhupendra & Swain, Mahendra. (2019). Python and Arduino with Pyfirmata. 10.1201/9780429284564-7.

[10] S., Manjula & Krishnamurthy, Lakshmi & Ravichandran, Manjula. (2016). A STUDY ON OBJECT DETECTION.

[11] Kaymak, Çağrı & Uçar, Ayşegül. (2018). Implementation of Object Detection and Recognition Algorithms on a Robotic Arm Platform Using Raspberry Pi. 1-8. 10.1109/IDAP.2018.8620916.

[12] S. S. Walam, S. P. Teli, B. S. Thakur, R. R. Nevarekar and S. M. Patil, "Object Detection and seperation Using Raspberry PI," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018, pp. 214-217, doi: 10.1109/ICICCT.2018.8473068.

[13] N. Gupta and A. K. Agarwal, "Object Identification using Super Sonic Sensor: Arduino Object Radar," 2018 International Conference on System Modeling & Advancement in Research Trends (SMART), 2018, pp. 92-96, doi: 10.1109/SYSMART.2018.8746951.

[14] A. U. Kulkarni, A. M. Potdar, S. Hegde and V. P. Baligar, "RADAR based Object Detector using Ultrasonic Sensor," 2019 1st International Conference on Advances in Information Technology (ICAIT), 2019, pp. 204-209, doi: 10.1109/ICAIT47043.2019.8987259.

[15] L. Koraqi and F. Idrizi, "“Detection, identification and tracking of objects during the motion”," 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), 2019, pp. 1-6, doi: 10.1109/ISMSIT.2019.8932833.

[16] S. M. R, F. Khan, G. G. R and H. S, "Object Detection and Human Identification using Raspberry Pi," 2019 1st International Conference on Advances in Information Technology (ICAIT), 2019, pp. 135-139, doi: 10.1109/ICAIT47043.2019.8987398.

[17] Khair, Ummul & Lestari, Yuyun & Perdana, Adidtya & Budiman, Arief. (2019). CCTV Camera Controller Simulation Using Arduino Uno and Joystick. Journal of Physics: Conference Series. 1361. 012086. 10.1088/1742-6596/1361/1/012086.

[18] Aditya, M. Sharma and S. Chand Gupta, "An Internet of Things Based Smart Surveillance and Monitoring System using Arduino," 2018 International Conference on Advances in Computing and Communication Engineering (ICACCE), 2018, pp. 428-433, doi: 10.1109/ICACCE.2018.8441725.

[19] S, Shilpashree & Hlmg, Lokesha & Shivkumar, Hadimani. (2015). Implementation of Image Processing on Raspberry Pi. IJARCCE. 4. 199-202. 10.17148/IJARCCE.2015.4545.

[20] V. K. Bhanse and M. D. Jaybhaye, "Face Detection and Tracking Using Image Processing on Raspberry Pi," 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), 2018, pp. 1099-1103, doi: 10.1109/ICIRCA.2018.8597246.

[21] Gupta, Ishita & Patil, Varsha & Kadam, Chaitali & Dumbre, Shreya. (2016). Face detection and recognition using Raspberry Pi. 83-86. 10.1109/WIECON-ECE.2016.8009092.

[22] A. A. Wazwaz, A. O. Herbawi, M. J. Teeti and S. Y. Hmeed, "Raspberry Pi and computers-based face detection and recognition system," 2018 4th International Conference on Computer and Technology Applications (ICCTA), 2018, pp. 171-174, doi: 10.1109/CATA.2018.8398677.

[23] Benito-Picazo, Jesús & Domínguez, Enrique & Palomo, Esteban & López-Rubio, Ezequiel & Ortiz-De-Lazcano-Lobato, Juan. (2018). Deep learning-based anomalous object detection system powered by microcontroller for PTZ cameras. 10.1109/IJCNN.2018.8489437.

[24] Leonard, Juan. (2019). Image Classification and Object Detection Algorithm Based on Convolutional Neural Network. Science Insights. 31. 85-100. 10.15354/si.19.re117.

[25] Goel, Rohini & Sharma, Avinash & Kapoor, Rajiv. (2019). Object Recognition Using Deep Learning. Journal of Computational and Theoretical Nanoscience. 16. 4044-4052. 10.1166/jctn.2019.8291.

Other References :

1. https://learnopencv.com/contour-detection-using-opencv-python-c/
2. https://www.tutorialspoint.com/dip/index.htm
3. https://www.javatpoint.com/digital-image-processing-tutorial
4. Digital Image Processing by Rafael C. Gonzalez and Richard E. Woods (4th Edition)
5. https://www.google.com/search?q=contour+detection+open+cv&rlz=1C1CHBF_enIN915IN915&sxsrf=AOaemvLPe0FbjEMkcCVCaNrUGtlM8YQIoQ:1640067701470&source=lnms&tbm=isch&sa=X&ved=2ahUKEwixrIOmoPT0AhXh63MBHcifAAAQ_AUoAnoECAEQBA#imgrc=_Ny_hHWt5UbSsM
6. http://www.fit.vutbr.cz/~vasicek/imagedb/
7. https://iitram.ac.in/front_picture
8. https://www.etechnog.com/2021/07/relay-wiring-diagram.html
9. https://circuitdigest.com/sites/default/files/circuitdiagram/Simple-relay-switch-circuit-diagram_0.png
10. https://www.google.com/url?sa=i&url=https%3A%2F%2Fepochabuse.com%2Fgamma-noise%2F&psig=AOvVaw0SrtoIece1SJDJjxRzXSDY&ust=1651251513127000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCJDx7rqdt_cCFQAAAAAdAAAAABAO
11. https://www.google.com/url?sa=i&url=https%3A%2F%2Fepochabuse.com%2Fsalt-and-pepper-noise%2F&psig=AOvVaw3CrOc112piThHFwyYgvsQs&ust=1651253562824000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCLi9zYOlt_cCFQAAAAAdAAAAABAD
12. https://www.google.com/url?sa=i&url=https%3A%2F%2Fgraphicdesign.stackexchange.com%2Fquestions%2F142508%2Fmapping-3d-rgb-color-space-to-1d-visible-light-wavelength&psig=AOvVaw2zBODgjk2A55uB8JALrggd&ust=1651253682969000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCNDjzsOlt_cCFQAAAAAdAAAAABAD
13. https://www.google.com/imgres?imgurl=https%3A%2F%2Fupload.wikimedia.org%2Fwikipedia%2Fcommons%2Fthumb%2F3%2F33%2FHSV_color_solid_cylinder_saturation_gray.png%2F197px-HSV_color_solid_cylinder_saturation_gray.png&imgrefurl=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FHSL_and_HSV&tbnid=Vhc4Xnik66j8RM&vet=12ahUKEwiB9sbwpbf3AhVR_DgGHU3XAs8QMygCegUIARDIAQ..i&docid=aQl2g-rSAw73VM&w=197&h=148&q=hsv%20image&ved=2ahUKEwiB9sbwpbf3AhVR_DgGHU3XAs8QMygCegUIARDIAQ
14. https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.led-professional.com%2Fresources-1%2Farticles%2Fled-light-spectrum-enhancement-with-transparent-pigmented-glazes-by-light-spectrum-glazes%2Fscreen-shot-2016-06-20-at-15-39-33.png%2F%40%40images%2F06fd791f-07cb-464a-bd93-973c113c8be3.png&imgrefurl=https%3A%2F%2Fwww.led-professional.com%2Fresources-1%2Farticles%2Fled-light-spectrum-enhancement-with-transparent-pigmented-glazes-by-light-spectrum-glazes&tbnid=w51_WLUstA685M&vet=12ahUKEwjR9--jp7f3AhWlyaACHfQzAsoQMygLegUIARDVAQ..i&docid=5S0IsZLovfvDjM&w=1280&h=473&q=color%20of%20visible%20light%20spectrogram&ved=2ahUKEwjR9--jp7f3AhWlyaACHfQzAsoQMygLegUIARDVAQ
15. https://www.google.com/imgres?imgurl=http%3A%2F%2F4.bp.blogspot.com%2F-zo6GV-bNkbI%2FT5XFuL-0wPI%2FAAAAAAAAABk%2FetVIn89lLzg%2Fs1600%2Fresult-2.jpg&imgrefurl=http%3A%2F%2Fhappystudybymaria.blogspot.com%2F2012%2F04%2Fproject-

boundary-extraction-by-dilation.html&tbnid=OX91gHukKEgeEM&vet=12ahUKEwiWxYrLp7f3AhWryaACHfGsB9IQMygLegUIARDQAQ..i&docid=dwHwTAZPBPCE8M&w=716&h=313&q=boundary%20extraction%20in%20image%20processing&ved=2ahUKEwiWxYrLp7f3AhWryaACHfGsB9IQMygLegUIARDQAQ

16. https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.mathworks.com%2Fmatlabcentral%2Ffileexchange%2F41221-canny-edge-detector&psig=AOvVaw1CYiMgZZ0q3iAf2d0F43M7&ust=1651254304911000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCNCtx--nt_cCFQAAAAAdAAAAABAE
17. https://www.google.com/imgres?imgurl=https%3A%2F%2Fcdn-images-1.medium.com%2Fmax%2F519%2F1*XycAd2NLLaAQ5hyQkhtRCQ.png&imgrefurl=https%3A%2F%2Ftowardsai.net%2Fp%2Fcomputer-vision%2Fwhat-is-a-canny-edge-detection-algorithm&tbnid=EaM5RhvpJLwWsM&vet=12ahUKEwjW_N2GqLf3AhXZx6ACHcsLCxUQMygiegUIARD-AQ..i&docid=7mEXdZR156OLwM&w=519&h=254&q=hysteresis%20thresholding&ved=2ahUKEwjW_N2GqLf3AhXZx6ACHcsLCxUQMygiegUIARD-AQ
18. https://www.google.com/imgres?imgurl=https%3A%2F%2Fmiro.medium.com%2Fmax%2F641%2F1*-gSrDDEVKw30MpW7jGghHA.png&imgrefurl=https%3A%2F%2Fmedium.com%2Fspinor%2Fa-straightforward-introduction-to-image-thresholding-using-python-f1c085f02d5e&tbnid=Cf6GhFwKQ0uROM&vet=12ahUKEwj47pChqLf3AhUT_TgGHXBtCfwQMygmegUIARCnAg..i&docid=9ulCSmaMmD9LEM&w=641&h=219&q=thresholding&ved=2ahUKEwj47pChqLf3AhUT_TgGHXBtCfwQMygmegUIARCnAg
19. https://www.google.com/url?sa=i&url=https%3A%2F%2Fslideplayer.com%2Fslide%2F13066226%2F&psig=AOvVaw0OriAuhcSagdpXKStvg20w&ust=1651254520087000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCIifjcyot_cCFQAAAAAdAAAAABAD