

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'Hamed BOUGARA – Boumerdes



Institute of Electrical and Electronics Engineering
Department of Electronics

Third Year Mini Project Report Presented in Partial Fulfillment of the
Requirements for the Degree of

LICENCE

In Electrical and Electronic Engineering

Title:

**Detection of pills using classic algorithms and
machine learning approach.**

Presented by:

- **BETTOUCHE Zineddine Mohamed Islem**
- **LOUAHCHI Zakarya**

Supervisor:

- **Dr.CHERIFI Dalila**

Registration Number: 2018/2019

Acknowledgement

All praise is due to Allah the most gracious and the most merciful.

We would like to express our special gratitude to our supervisor Dr. CHERIFI Dalila for her patience and her support on the way, and for her useful comments, remarks, and engagement through the learning process of this license project.

We would like to express our deepest appreciation to all those who provided us with the possibility to complete this report.

Our gratitude also goes to our parents, family and friends for their invaluable support in our entire life.

Abstract

The industry of medical pill manufacturing takes a big share in the modern industry market. But due to the immense amount of production, flaws happen during tablet compression which is critical for consumer health and convenience.

The aim of our project is to study the detection of pills using "*Python*" and recognition using both classic algorithms in image processing and the machine learning approach.

In the experimental part, we tested our algorithms on some examples, which are able to detect their count, thus checking whether the appropriate number is produced.

Content

Acknowledgment	02
Abstract	03
Content	04
List of Figures	06
Introduction	07
Chapter I: Review about Image Processing	08
I.1 Introduction	09
I.2 Image Processing	09
I.3 Application of Image Processing	10
I.4 Medical Pills Manufacturing	12
I.5 Summary	13
Chapter II: Classic Algorithms in Image Processing	14
II.1 Median Filter	15
II.2 Canny Edge Detection	16
II.3 Circle Hough Transform	17
II.4 Blob Detection	19
II.5 Summary	19
Chapter III: Machine Learning	20
III.1 Definition	21
III.2 Types of Machine Learning	21
III.3 Application of Machine Learning Algorithms	21
III.4 Steps Involved in Machine Learning	22
III.5 Haar Cascade Classifier	22
III.6 Training Machine Learning Algorithms	22
III.7 Summary	27
Chapter IV: Experimental Part	28
IV.1 Experiment Overview	29
IV.2 Python	29

IV.3 Algorithm Implementation	30
IV.4 Experiment Sampled in Desktop Application	34
IV.5 Further Application	37
IV.6 Summary	38
Conclusion	39
Reference	40

List of Figures

Fig 1.1: Electromagnetic Spectrum

Fig 1.2: Blurred Image

Fig 1.3: Brain Activity

Fig 1.4: QR Code Scanning

Fig 1.5: Face Recognition Pattern

Fig 1.6: Fingerprint Recognition

Fig 1.7: Pills Manufacturing

Fig 1.8: Typical Tablet

Fig 2.1: Median Filter

Fig 2.2: Naïve Implementation of Median Blur

Fig 2.3: Circle Hough Transform

Fig 3.1: Types of Machine Learning

Fig 3.2: Haar Cascade Classifier

Fig 3.3: GUI Trainer –Input Tab

Fig 3.4: GUI Trainer –Common Tab

Fig 3.5: GUI Trainer –Cascade Tab

Fig 3.6: GUI Trainer –Boost Tab

Fig 3.7: GUI Trainer –Log Tab

Fig 4.1: Sample of Medical Pills

Fig 4.2: Experiment 01

Fig 4.3: Experiment 01 Results

Fig 4.4: Experiment 02

Fig 4.5: Experiment 02 Results

Fig 4.6: Experiment 03

Fig 4.7: Experiment 03 Results

Fig 4.8: Execution Flowchart

Fig 4.9: GUI Application Screen Shots

Fig 4.10: Raspberry Pi

Fig 4.11: Servomotor

Fig 4.12: Realization Flowchart

Introduction

Image processing is the development of computer systems to be able to process digital images, to make modifications or to extract data from them.

This project combines the field of image processing, machine learning, and software development, in order to find a solution to the problems faced in medical pills manufacturing.

The main objective is to implement a routine of algorithms to detect flaws, i.e. recognizing pills using a camera, then to make a decision whether or not a mistake was made.

This report is organized into four chapters: Chapter 1 presents a brief introduction to image processing, its application, and medical pills manufacturing, which will be the scope of our experiment. Chapter 2 introduces the algorithms used to achieve our goal, these algorithms are classics in the field of image processing. Chapter 3 presents one of the techniques used to detect medical pills, which is machine learning. We will give a general overview about machine learning, its different concepts, and how our model is trained. Finally, Chapter 4 contains the experiment with the different approaches to determine the most efficient one. In addition to that, we will experiment medical pills detection using machine learning base on Python script and demonstrate our implementation.

Chapter I

Review about Image Processing

In this chapter we are giving an overview about image processing and medical pill manufacturing. Where image processing will be our tool a medical pill will be our field of interest.

I.1. Introduction:

This chapter contains a review and some definitions about: image processing. We will also see some applications and examples about in this field.

I.2. Image Processing:

Electromagnetic waves can be thought of as steam of particles, where each particle is moving with the speed of light. Each particle contains a bundle of energy. This bundle of energy is called a photon. The electromagnetic spectrum according to the energy of photon is shown in the following figure:

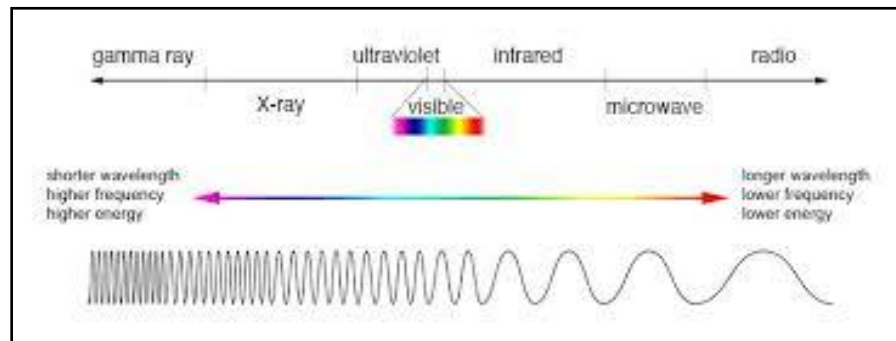


Fig 1.1: Electromagnetic Spectrum

In this electromagnetic spectrum, we are only able to see the visible spectrum, which includes seven different colors that are violet, indigo, blue, green, orange, yellow and red. But that does not nullify the existence of other stuff in the spectrum. Our human eye can only see the visible portion, in which we saw all the objects. But a camera can see the other things that a naked eye is unable to see. For example: x rays, gamma rays...etc. hence the analysis of all that stud is done in digital image processing, because it is used a lot in the field of medical, nuclear medicine, and astronomical observation. In order to study how digital images work, we have to understand the image formation in human eyes. When light falls upon the particular object, it is reflected back after striking through the object. The rays of light when passed through the lens of eye, form a particular angle, and the image is formed on the retina which is the back side of the wall. The image that is formed is inverted. This image is then interpreted by the brain and that makes us able to understand things.

In a digital camera an array of small analog sensors senses the intensity of light at each point. When photons of light strike on the chip, it is held as a small electrical charge in each photo sensor. The response of each sensor is directly equal to the amount of light struck on the surface of the sensor.

The limited number of sensors in the camera defines the resolution of the produced image which affects the image quality, ie: the higher the number of pixels, the more the amount of details in the image. A digital image can be seen as a 2D array of pixels, where each pixel holds the amount of light caught by one sensor.

Digital-image-processing deals with the manipulation of digital images through a digital computer. It is a subfield of signals and systems but focuses particularly on images. DIP focuses on developing a computer system that is able to perform processing on an image to extract information from it. The input of that system is a digital image and the system process that image using efficient algorithms, and gives an output.

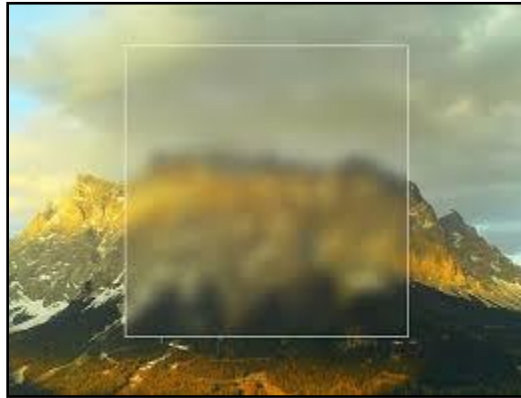


Fig 1.2: Blurred image

I.3. Applications of Image Processing:

Image processing is used in many areas of science and engineering; it has applications in computer vision, medical imaging, radar processing, text classification, and many other fields.

- **Brain Activity Observation:**

One of the applications of pattern recognition in medical field is the brain imaging. It is the use of various techniques to either directly or indirectly visualize the structure of the nervous system. We may understand patterns of brain activity and find relationships between brain activities, cognition, and behaviors. It can also be used to detect cerebral abnormalities and diseases like tumors, cancers...etc.

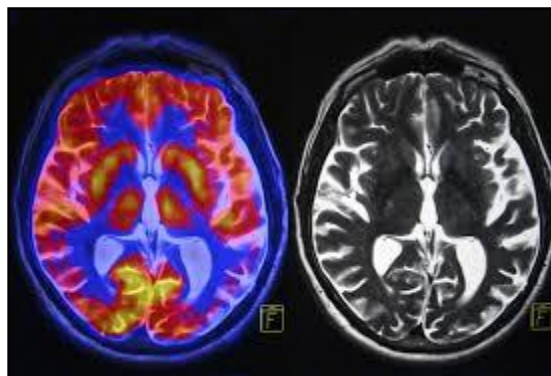


Fig 1.3: Brain Activity

- **QR Codes:**

A QR code is a machine-readable optical label that contains information about the item it is attached to. It consists of black squares arranged in a square grid on a white background, which can be read by an imaging device such as cameras, and processed until the image can be appropriately interpreted. The required data is then extracted from patterns that are present in both horizontal and vertical components of the image.



Fig 1.4: QR Code Scanning

- **Biometric Applications:**

In modern technologies, image processing and computer vision have been used a lot in biometrics which is the measurement and analysis of unique physical or behavioral characteristics to provide more secure systems for identifying humans (face recognition).

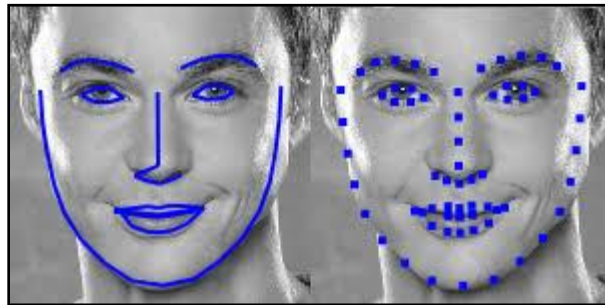


Fig 1.5: Face Recognition Pattern

- ✚ Face Recognition: it allows a machine to correctly recognize a person by providing an image of his face, it is used in security cameras in public places, as in airports, to track dangerous persons stored in a database. One of the ways to do it is by comparing selected facial features from the image and a face database.

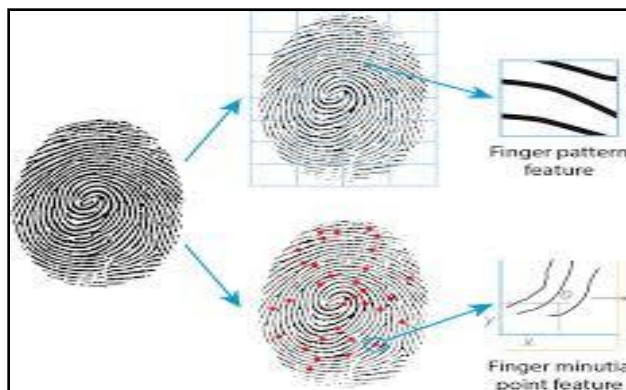


Fig 1.6: Fingerprint Recognition

- ✚ Fingerprint Recognition: Human fingerprints are detailed, nearly unique, difficult to alter, and durable over the life of an individual, making them suitable as long-term markers of human identity. Fingerprints are used as recognition system to identify persons. One of its applications is the attendance systems in companies; we can find it also in bank's cash distributors, and in smart phones unlocking systems. Fingerprint analysis is used since the early 20th century as it has led to solving many crimes investigations.

I.4. Medical Pills Manufacturing:



Fig 1.7: Pills Manufacturing

In the tablet pressing process, the appropriate amount of active ingredient must be in each tablet. Hence, all the ingredients should be well-mixed. If a sufficiently homogenous mix of the components cannot be obtained with simple blending processes, the ingredients must be granulated prior to compression to assure an even distribution of the active compound in the final tablet. Two basic techniques are used to granulate powders for compression into a tablet: wet granulation and dry granulation. Powders that can be mixed well do not require granulation and can be compressed into tablets through direct compression[2].

However, the immense daily tablet production leads to flaws when pills are being compressed, and detecting the flaw of a missing pill in a tablet is the scope of this project.



Fig 1.8: Typical tablets

I.4.1. Detecting the Flaw of a Missing Pill:

To detect a missing pill, we are going to take the approach of counting pills and then comparing the count to the desired number in a tablet. For this, we need algorithms in image processing to detect circular and elliptical shapes, since all pills come in either of these two shapes. Therefore, in the next two chapters, we're going to introduce some of the algorithms to detect the desired shapes.

I.5. Summary:

In this chapter we introduced the overall image processing techniques, and the purpose of our project, which is detecting medical pills. Our next step is studying the different approaches in image processing to achieve the desired results.

Chapter II

Classic Algorithms in Image Processing

In this chapter we are going to introduce some famous algorithms in image processing, which are going to be used in the experimental part to result the desired outcome.

Introduction:

In this chapter, we are going to introduce the algorithms used to achieve our goal. These algorithms are classics in the field of image processing, and they are listed as follows:

- Median Filter
- Canny Edge Detection
- Circle Hough Transform
- Blob Detection

II.1. Median Filter:

The Median Filter is a linear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing. Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise.

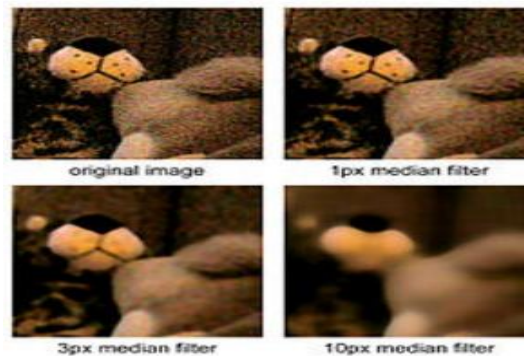


Fig 2.1: Median Filter

2D Median Filter Pseudo Code:

Note that this algorithm: processes one color channel only, and takes the "not processing boundaries" approach.

```
allocate outputPixelValue[image width][image height]
allocate window>window width * window height
edgex := (window width / 2) rounded down
edgey := (window height / 2) rounded down
for x from edgex to image width - edgex
  for y from edgey to image height - edgey
    i = 0
    for fx from 0 to window width
      for fy from 0 to window height
        window[i] := inputPixelValue[x + fx - edgex][y + fy - edgey]
        i := i + 1
    sort entries in window[]
    outputPixelValue[x][y] := window>window width * window height / 2]
```


The majority of the computational effort and time is spent on calculating the median of each window. Because the filter must process every entry in the signal, for large signals such as images, the efficiency of this median calculation is a critical factor in determining how fast the algorithm can run. The naïve implementation described above sorts every entry in the window to find the median.



Fig 2.2: Naïve Implementation of Median Blur

All smoothing techniques are effective at removing noise, but adversely affect edges. At the same time as reducing the noise, it is important to preserve the edges. Edges are of critical importance to the visual appearance of images, for speckle noise and salt-and-pepper noise, it is particularly effective. Because of this, median filtering is very widely used in digital image processing[3].

II.2. Canny Edge Detection:

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. The general criteria for edge detection include:

- 1) Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible.
- 2) The edge point detected from the operator should accurately localize on the center of the edge.
- 3) A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

Among the edge detection methods developed so far, Canny edge detection algorithm is one of the most strictly defined methods that provides good and reliable detection[4].

- **Process of Canny edge detection algorithm:**

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

- 1) Apply smoothing filter to the image in order to remove the noise.
- 2) Find the intensity gradients of the image.
- 3) Apply non-maximum suppression to get rid of spurious response to edge detection.
- 4) Apply double threshold to determine potential edges.
- 5) Track edge by hysteresis.

II.3. Circle Hough Transform:

The circle Hough Transform (CHT) is a feature extraction technique for detecting circles in a digital image. It is a specialization of Hough Transform. The purpose of the technique is to find circles in imperfect image inputs. The circle candidates are produced by “voting” in the Hough parameter space and then select the local maxima in a so-called accumulator matrix.

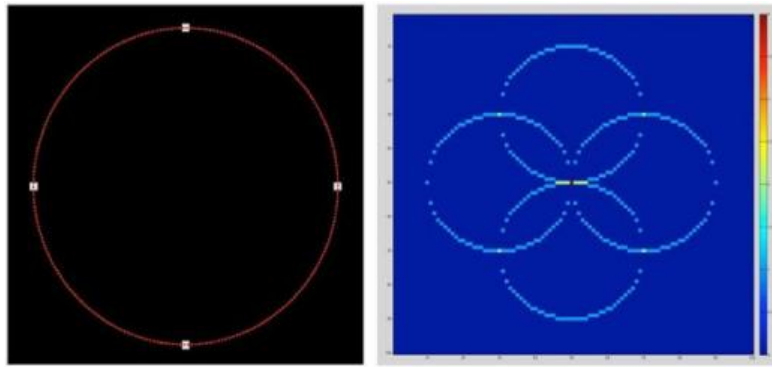


Fig 2.3: Circle Hough Transform

In a two-dimensional space, a circle can be described by: $(x - a)^2 + (y - b)^2 = r^2$ where (a, b) is the center of the circle, and r is the radius. If a 2D point (x, y) is fixed, then the parameters can be found. The parameter space would be three dimensional, (a, b, r) and all the parameters that satisfy (x, y) would lie on the surface of an inverted right-angled cone whose apex is at $(x, y, 0)$. This process can be divided into two stages. The first stage is fixing radius then find the optimal center of circles in a 2D parameter space. The second stage is to find the optimal radius in a one dimensional parameter space [5].

- **Find parameters with known radius R:**

If the radius is fixed, then the parameter space would be reduced to 2D (the position of the circle center). For each point (x, y) on the original circle, it can define a circle centered at (x, y) with radius R . The intersection point of all such circles in the parameter space would be corresponding to the center point of the original circle. An accumulator matrix is used for tracking the intersection point. In the parameter space the voting number of points, through which the circle is passing, would be increased by one; then the local maxima point can be found. The position (a, b) of the maxima would be the center of the original circle.

- **Accumulator matrix and voting:**

Accumulator matrix is introduced to find the intersection point in the parameter space. First, we need to divide the parameter space into “buckets” using a grid and produce an accumulator matrix according to the grid. The element in the accumulator matrix denotes the number of “circles” in the parameter space that passing through the corresponding grid cell in the parameter space. The number is also called “voting number”. Initially, every element in the matrix is zeros. Then for each “edge” point in the original space, we can formulate a circle in the parameter space and increase the voting number of the grid cell which the circle passing through. This process is called “voting”. After voting, we can find local maxima in the accumulator matrix. The positions of the local maxima are corresponding to the circle centers in the original space.

- **Find circle parameter with unknown radius:**

Since the parameter space is 3D, the accumulator matrix would be 3D, too. We can iterate through possible radii; for each radius, we use the previous technique. Finally, find the local maxima in the 3D accumulator matrix. Accumulator array should be $A[x, y, r]$ in the 3D space. Voting should be for each pixels, radius and theta $A[x, y, r] += 1$

The algorithm:

- 1) For each $A[a, b, r] = 0$;
- 2) Smoothen the image with a blurring filter (Median), convert the image to grayscale, make Canny operator (producing edges).
- 3) Vote all possible circles in accumulator.
- 4) The local maximum voted circles of accumulator A gives the circle Hough space.
- 5) The maximum voted circle of Accumulator B gives the circle.

The Voting:

```

For each pixel(x,y)
  For each radius r = 10 to r = 60 // the possible radius
    For each theta t = 0 to 360 // the possible theta 0 to 360
      a = x - r * cos(t * PI / 180); //polar coordinate for center
      b = y - r * sin(t * PI / 180); //polar coordinate for center
      A[a,b,r] +=1; //voting
    end
  end
end

```

II.4. Blob Detection:

Blob detection methods are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. A blob is a region of an image in which some properties are constant or approximately constant. The most common method for blob detection is convolution[6].

II.4.1. Laplacian of Gaussian:

One of the first and also most common blob detectors is based on the Laplacian of the Gaussian (LoG). Given an input image $f(x, y)$, this image is convolved by a Gaussian kernel:

$$g(x, y, z) = \frac{1}{2\pi t} e^{-\frac{x^2+y^2}{2t}} \quad (2.1)$$

at a certain scale to give a scale space representation $L(x, y, t) = g(x, y, t) * f(x, y)$. Then, the result of applying the Laplacian operator $\nabla^2 L = L_{xx} + L_{yy}$ is computed, which usually results in strong positive responses for dark blobs of radius $r = \sqrt{2t}$ (for a two-dimensional image, for a d-dimensional image $r = \sqrt{dt}$) and strong negative responses for bright blobs of similar size. Automatically capture blobs of different (unknown) size in the image domain, a multi-scale approach is therefore necessary.

A straightforward way to obtain a multi-scale blob detector with automatic scale selection is to consider the scale-normalized Laplacian operator $\nabla_{\text{norm}}^2 L = t(L_{xx} + L_{yy})$ and to detect scale-space maxima/minima, (points that are simultaneously local maxima/minima of $\nabla_{\text{norm}}^2 L$ with respect to both space and scale). Thus, simultaneous selection of interest points (\hat{x}, \hat{y}) and scales \hat{t} is performed according to

$$(\hat{x}, \hat{y}, \hat{t}) = \underset{(x,y,t)}{\operatorname{argmaxminlocal}} \left((\nabla_{\text{norm}}^2 L)(x, y, t) \right) \quad (2.2)$$

II.5. Summary:

In this chapter, we have introduced the used algorithms in our project with their different type and different parameters. In the next chapter we will present the machine learning approach.

Chapter III

Machine Learning

In this chapter we are going to see one of the techniques used to detect medical pills which is machine learning. We will give a general overview about machine learning, its different concepts, and how our model is trained.

III.1. Definition:

Machine learning is a discipline that deals with programming the systems so as to make them automatically learn and improve with experience. Here, learning implies recognizing and understanding the input data and taking informed decisions based on the supplied data. It is very difficult to consider all the decisions based on all possible inputs. To solve this problem, algorithms are developed that build knowledge from a specific data and past experience by applying the principles of statistical science, probability, logic, mathematical optimization, reinforcement learning, and control theory[7].

III.2. Types of Machine Learning:

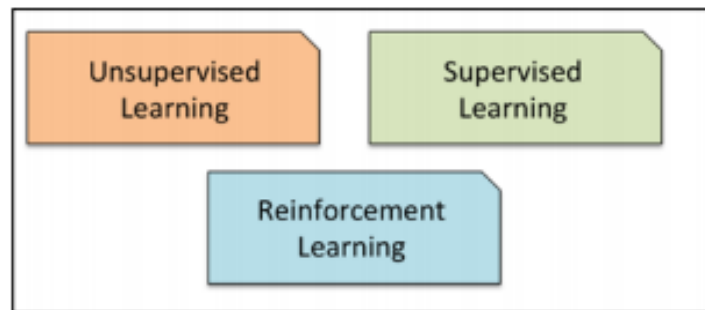


Fig 3.1: Types of Machine Learning

- 1) **Supervised Learning:** The main goal in supervised learning is to learn a model from labeled training data that allows us to make predictions about unseen or future data. Here, the term supervised refers to a set of samples where the desired output signals (labels) are already known.
- 2) **Unsupervised Learning:** In unsupervised learning, we are dealing with unlabeled data. Using unsupervised learning techniques, we are able to explore the structure of our data to extract meaningful information without the guidance of a known outcome variable or reward function.
- 3) **Reinforcement Learning:** In reinforcement learning, the goal is to develop a system (agent) that improves its performance based on interactions with the environment. The feedback is not the correct ground truth label or value, but a measure of how well the action was measured by a reward function. Through the interaction with the environment, an agent can then use reinforcement learning to learn a series of actions that maximizes this reward via an exploratory trial-and-error approach or deliberative planning.

III.3. Application of Machine Learning Algorithm:

The developed machine learning algorithms are used in various applications such as:

- Vision processing
- Language processing
- Forecasting things like stock market trends, weather
- Pattern recognition
- Games

III.4. Steps Involved in Machine Learning:

A machine learning project involves the following steps:

- Defining a Problem
- Preparing Data
- Evaluating Algorithms
- Improving Results
- Presenting Results

III.5. Haar Cascade Classifier:

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

We need to extract features from images. For this, Haar features shown in the image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

We apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications[8].

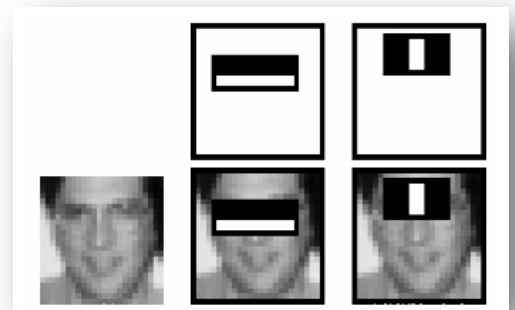
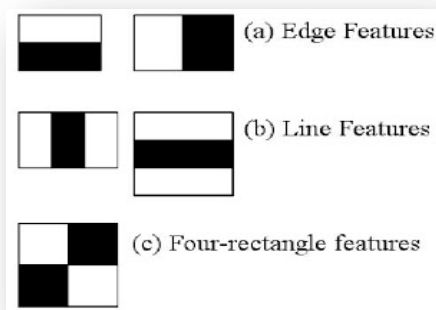


Fig 3.2: Haar Cascade Classifier

III.6. Training Machine-Learning Algorithms:

The process of training a model involves providing an algorithm with training data to learn from. The term ML model refers to the model artifact that is created by the training process. The training data must contain the correct answer, which is known as a target. Learning algorithm finds patterns in the training data that map the input data attributes to the target and it outputs an ML model that captures these patterns.

To train an ML model, you need to specify the following:

- Input training data source
- Name of the data attribute that contains the target to be predicted
- Required data transformation instructions
- Training parameters to control the learning algorithm

To train Haar cascade classifiers specifically, there are 3 ways:

- 1- Use the CMD (Windows) and documentation by Auckland University, this method however is outdated and does not guarantee the best accuracy.
- 2- Sentdex's approach to make your own cascade classifier, but the problem with this is that you need to have a paid Digital Ocean server.
- 3- The best method by far (quick, easy, best accuracy) is using the GUI application (Windows OS). It takes very short time to train with the best results, and that is what we will be using to make our own Haar cascade to detect custom objects.

III.6.1 GUI Cascade Trainer:

GUI Cascade Trainer is a program that can be used to train, test, and improve cascade classifier models. It uses a graphical interface to set the parameters and make it easy to use OpenCV tools for training and testing classifiers[9].

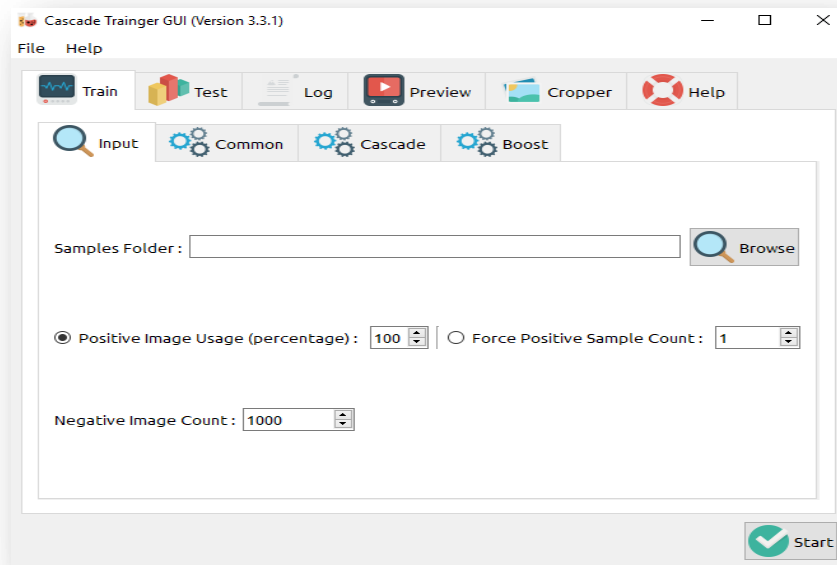


Fig 3.3: GUI Trainer –Input Tab

How to use:

When Cascade Trainer GUI is first started you will be presented with the following screen. This is the starting screen and it can be used for training classifiers. To train classifiers usually you need to provide the utility with thousands of positive and negative image samples, but there are cases when you can achieve the same with less samples.

To start the training, you need to create a folder for your classifier. Create two folders inside it. One should be “p” (for positive images) and the other should be “n” (for negative images). For example, you should have a folder named “Car” which has the mentioned folders inside it. Positive image samples are the images of the object you want to train your classifier and detect. For example, if you want to train and detect cars then you need to have many car images and you should also have many negative images. Negative images are anything but cars.

Important Note 1: Negative images must NEVER include any positive images. Not even partially.

Important Note 2: In theory, negative images can be any image that is not the positive image but in practice, negative images should be relevant to the positive images. For example, using sky images as negative images is a poor choice for training a good car classifier, even though it doesn't have a bad effect on the overall accuracy:

Start by pressing the browse button in train tab. Select the folder you have created for the classifier.

- Common, cascade and boost tabs can be used for setting numerous parameters for customizing the classifier training. Cascade Trainer GUI sets the most optimized and recommended settings for these parameters by default, still some parameters need to be modified for each training. Note that detailed description of all these parameters are beyond the scope of this help page and require deep knowledge about cascade classification techniques.
- You can set pre-calculation buffer size to help with the speed of training process. You can assign as much memory as you can for these but be careful to not assign too much or too low. For example if you have 8 GB of RAM on your computer then you can safely set both of the buffer sizes below to 2048.

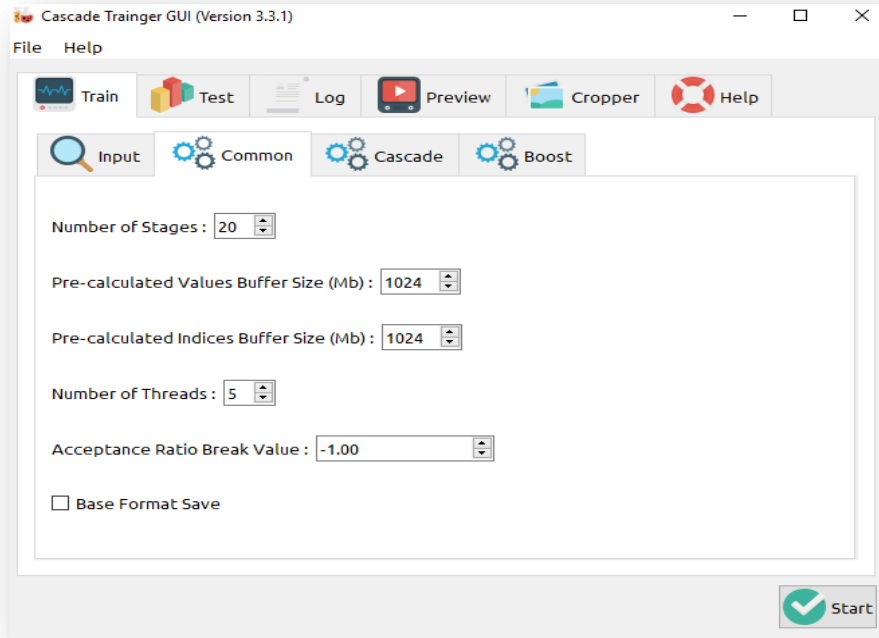


Fig 3.4: GUI Trainer –Common Tab

Next you need to set the sample width and height. Make sure not to set it to a very big size because it will make your detection very slow. Actually it is quite safe to always set a small value for this. Recommended settings for sample width and height is that you keep one aspect on 24 and set the other accordingly. For example, if you have sample images that are 320×240 then first calculate the aspect ratio which in this case is 1.33:1 then multiply the bigger number with 24. You'd get 32×24 for sample width and height.

You can also set the feature type to HAAR or LBP. Make sure to use HOG only if you have OpenCV 3.1 or later. HAAR classifiers are very accurate but require a lot more time to train so it is much wiser to use LBP if you can provide your classifiers with many sample images. LBP classifiers on the other hand are less accurate but train much quicker and detect almost 3 times faster.

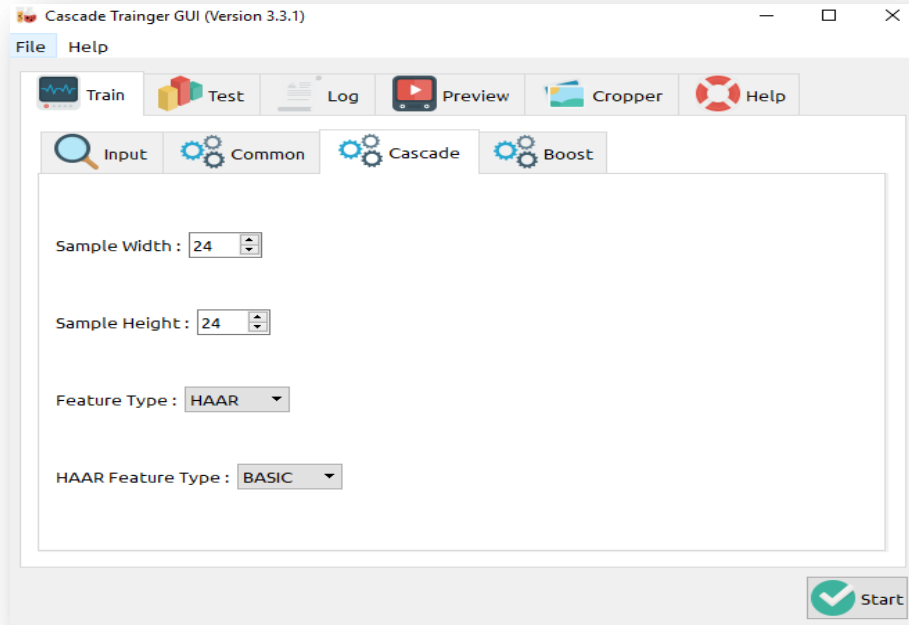


Fig 3.5: GUI Trainer –Cascade Tab

As for the parameters in the Boost tab, it is recommended to keep the default values unless you are quite sure about what you're doing.

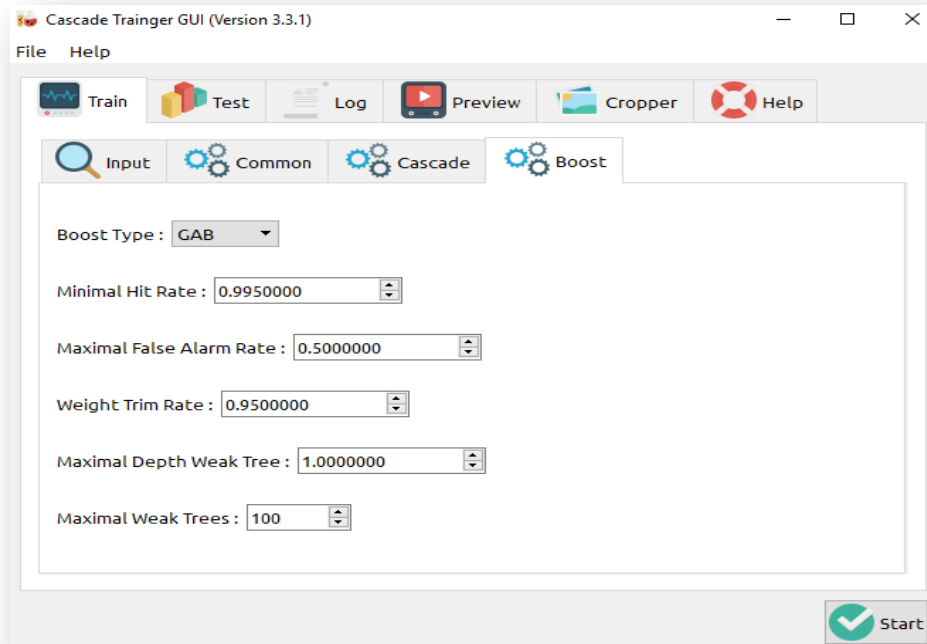


Fig 3.6: GUI Trainer –Boost Tab

After all the parameters are set, press Start button at the bottom to start training your cascade classifier. You'll see the following log screen while training is going on.

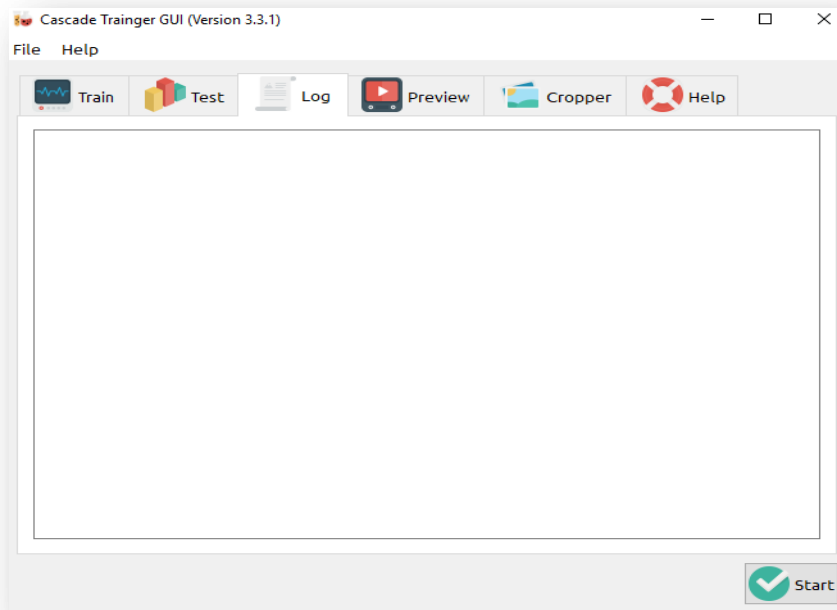


Fig 3.7: GUI Trainer –Log Tab

Wait for the training to complete.

Now if you exit Cascade Trainer GUI and go to the classifier folder you will notice that there are new files and folders are created in this folder.

“n” and “p” are familiar but the rest are new. “classifier” folder contains XML files that are created during different stages of training. If you check inside “classifier” folder, you’ll notice something similar to the following. “stage#.xml” files are temporary files that won’t be needed anymore. “params.xml” contains the parameters you have used for the training. (Just a reminder file) “cascade.xml” is the actual cascade classifier and if the training completed successfully then you should have this file inside classifier folder. “neg.lst”, “pos.lst” and “pos_samples.vec” are temporary files created for training the classifier and they can also be removed without having any effect.

III.7. Summary:

In this chapter, we introduced a presentation about machine learning: definition, types, and different examples of application. Then we passed to presenting the specific Haar cascade classifier and how the model is trained using the GUI Cascade Trainer. We have studied machine learning, because it is our tool in order to detect medical pills in the experimental part.

Chapter IV

Experimental Part

In this chapter we will experiment with the different approaches to determine the most efficient one. In addition to that, we will experiment medical pills detection using machine learning base on Python script and demonstrate our implementation.

IV.1. Experiment Overview:

In the experiments we used an image that contains some medical pills, in order to make the experiment as close as possible to the real life.



Fig 4.1: Sample of Medical Pills

The algorithms previously-introduced are going to be applied using the Python programming language.

IV.2. Python Programming Language:

IV.2.1. Definition:

Python is an interpreter, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale projects[10].







IV.2.2. Computer Vision with Python:

During our implementation, we have been using two Python libraries: OpenCV and Scikit-learn.

- **OpenCV** is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source BSD license. OpenCV supports the deep learning frameworks TensorFlow and Torch/PyTorch [11].
- **Scikit-Learn** is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and BDSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy[12].

IV.3. Algorithms Implementation:

First of all, the experiment has three paths. The algorithms are applied in the following three experiments:

- 1- Gray Scale  Median Blur  Edge Detection  Hough Circle Transform.
- 2- Gray Scale  Median Blur  Blob Detection.
- 3- Gray Scale  Haar Cascade Classifier.

Now we introduce the Python code implementation for each experiment:

- Experiment 1:

```
1 import cv2
2 import numpy as np
3
4 image = cv2.imread("n.jpg", 1)
5
6 blurred_image = cv2.medianBlur(image, 5)
7
8 edges = cv2.Canny(blurred_image, 100, 255)
9
10 circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 2, 20,
11                             param1=50, param2=30, minRadius=0, maxRadius=30)
12 circles = np.uint16(np.around(circles))
13 for i in circles[0, :]:
14     cv2.circle(image, (i[0], i[1]), i[2], (0, 0, 255), 2)
15
16 cv2.imshow("image", image)
17 cv2.waitKey(0)
18 cv2.destroyAllWindows()
```

Fig 4.2: Experiment 1

Line 1-2: importing libraries.

Line 4: Reading the image as gray-scaled.

Line 6: Applying the Median blur (kernel 5x5)

*Line 8: Applying Canny Edge Detection with
a threshold value of 100*

Line 10: Applying Hough Circle Transform

Line 12-14: Drawing the obtained circles

Line 16-18: Displaying the resulting image

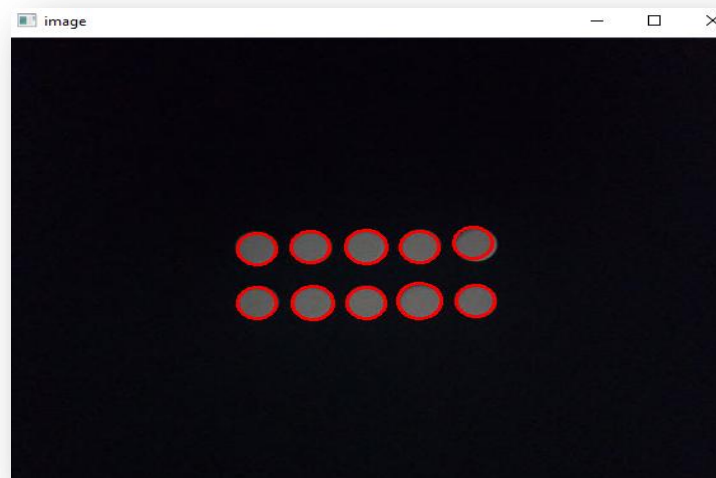


Fig 4.3: Experiment 01 Results

Experiment 2:

```
1  import cv2, numpy as np
2
3  image = cv2.imread("n.jpg", 1)
4  blurred_image = cv2.medianBlur(image, 5)
5
6  params = cv2.SimpleBlobDetector_Params()
7  params.filterByConvexity = False
8  params.filterByCircularity = False
9  params.filterByColor = False
10 params.filterByArea = True
11 params.minArea = 300
12 params.minDistBetweenBlobs = 20
13 params.minThreshold = 200
14
15 detector = cv2.SimpleBlobDetector_create(params)
16 keypoints = detector.detect(blurred_image)
17
18 processedImage = cv2.drawKeypoints(image, keypoints, np.array([]),
19                                     (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
20
21 cv2.imshow("processed", processedImage)
22
23 cv2.waitKey(0)
24 cv2.destroyAllWindows()
```

Fig 4.4: Experiment 2

Line 1: Importing Libraries

Line 3: Reading the image as gray-scaled

Line 4: Applying Median Blur (kernel 5x5)

*Line 6-13: Defining the parameters of blob
detector*

Line 15-16: Applying blob detector and extracting key-points of resulted blobs

Line 18: Drawing circles around resulted blobs

Line 21-24: Displaying the resulted image

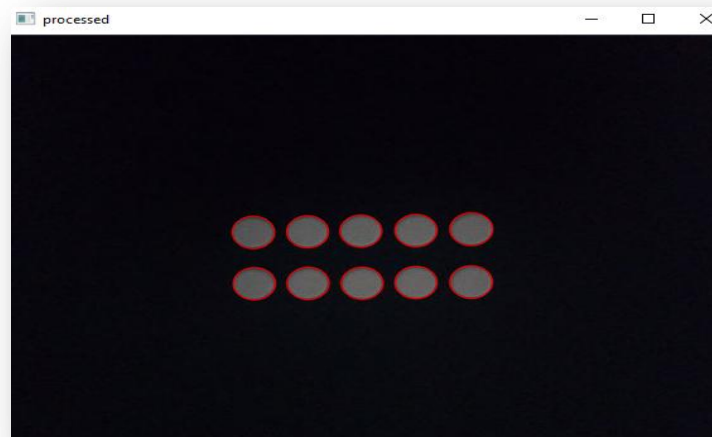


Fig 4.5: Experiment 2 Results

Experiment 3:

```
1  import cv2
2
3  image = cv2.imread("n.jpg", 1)
4
5  blurred_image = cv2.medianBlur(image, 5)
6
7  cascade = cv2.CascadeClassifier("cascade.xml")
8  pill = cascade.detectMultiScale(blurred_image, 1.3, 5)
9
10 for (gx, gy, gw, gh) in pill:
11     cv2.rectangle(image, (gx, gy), (gx + gw, gy + gh), (100, 100, 0), 2)
12
13 cv2.imshow("image", image)
14 cv2.waitKey(0)
15 cv2.destroyAllWindows()
```

Fig 4.6: Experiment 3

Line 1: Importing libraries

Line 3: Reading image as gray-scaled

*Line 7-8: Defining the cascade classifier and
applying it on the blurred image*

Line 10-11: Drawing the detected pills

Line 13-15: Displaying the resulted image

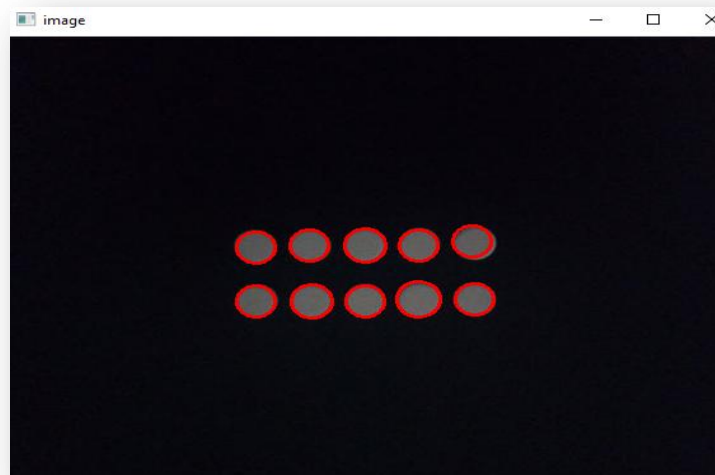


Fig 4.7: Routine 03 Results

IV.3.1. Experiments comparison:

As the three routines come into real-time practical implementation, there are some trade-offs concerning favoring one over the others.

- **Speed-accuracy trade-off:** Classic algorithms in image processing take less time than machine learning to execute due to simple stages involved in implementation. However, if one parameter is changed, results of classic algorithms have a terrible accuracy comparably to machine learning. This is the result of pixel dependency. Since Python is known for its high-speed execution, and since the microprocessors used these days are fast, the speed gap between classic algorithms and machine learning is unnoticeable, therefore we stick with the machine learning implementation (Experiment 3).

However we are going to build a desktop application to provide every routine on demand.

IV.4. Experiment Sampled in Desktop Application:

To provide a user-friendly experience, we have built a GUI desktop application to ensemble all routines and functionalities of this project with a real-time display of the detected medical pills.

IV.4.1. PyQt as Python Binding for Qt Library:

PyQt is one of the most popular Python bindings for the Qt cross-platform C++ framework. PyQt developed by Riverbank Computing Limited. Qt itself is developed as part of the Qt Project. PyQt provides bindings for Qt 4 and Qt 5. PyQt is distributed under a choice of licenses: GPL version 3 or a commercial license[13].

PyQt is available in two editions: PyQt4 which will build against Qt 4.x and 5.x and PyQt5 which will only build against 5.x. Both editions can be built for Python 2 and 3. PyQt contains over 620 classes that cover graphical user interfaces, XML handling, network communication, SQL databases, Web browsing and other technologies available in Qt.

Notice that our application runs on the main thread, where the image processing flow runs on a parallel thread built using QThread class in PyQt.

IV.4.2. Application Flowchart:

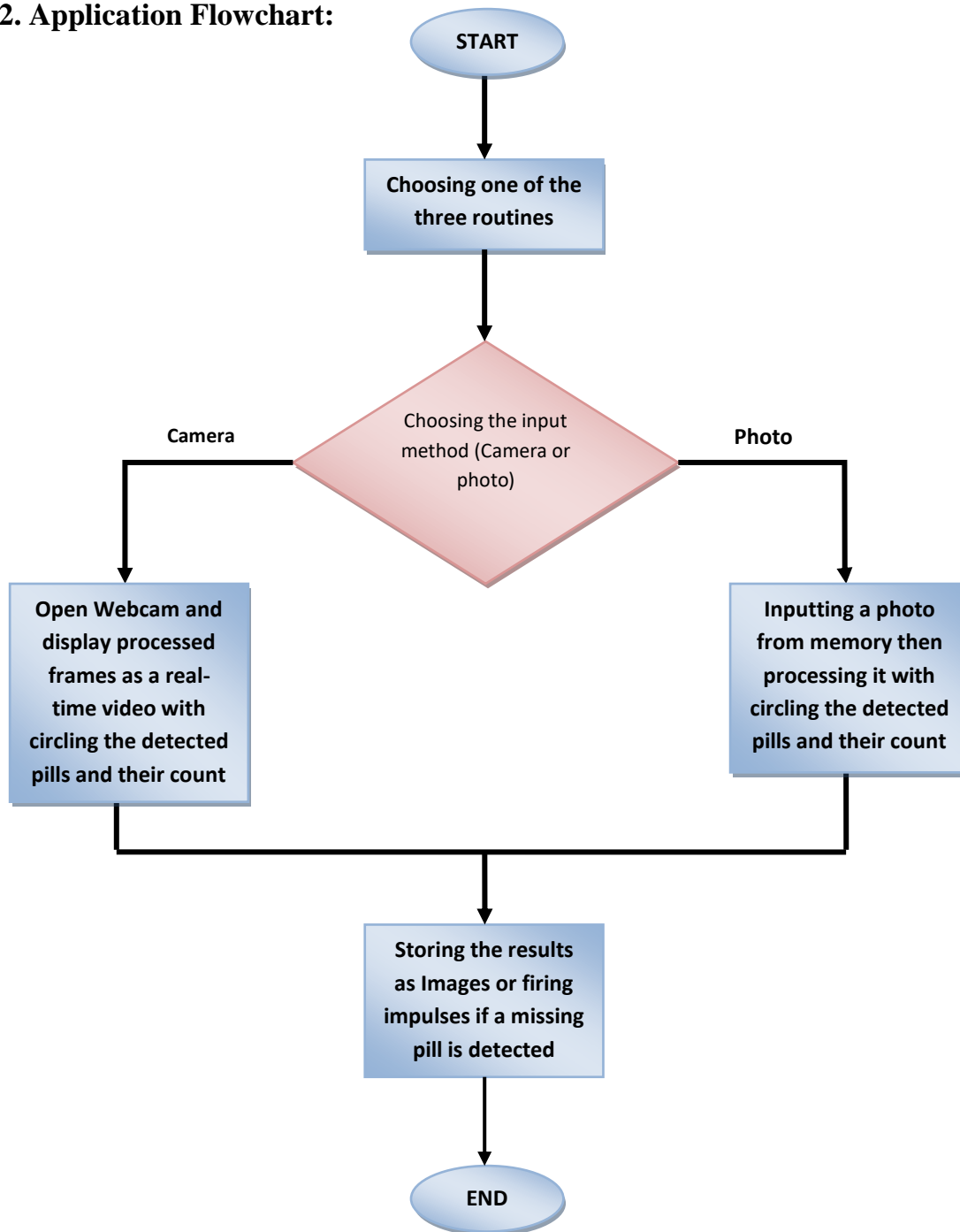


Fig 4.8: Execution Flowchart

IV.4.3. Application Screenshots:

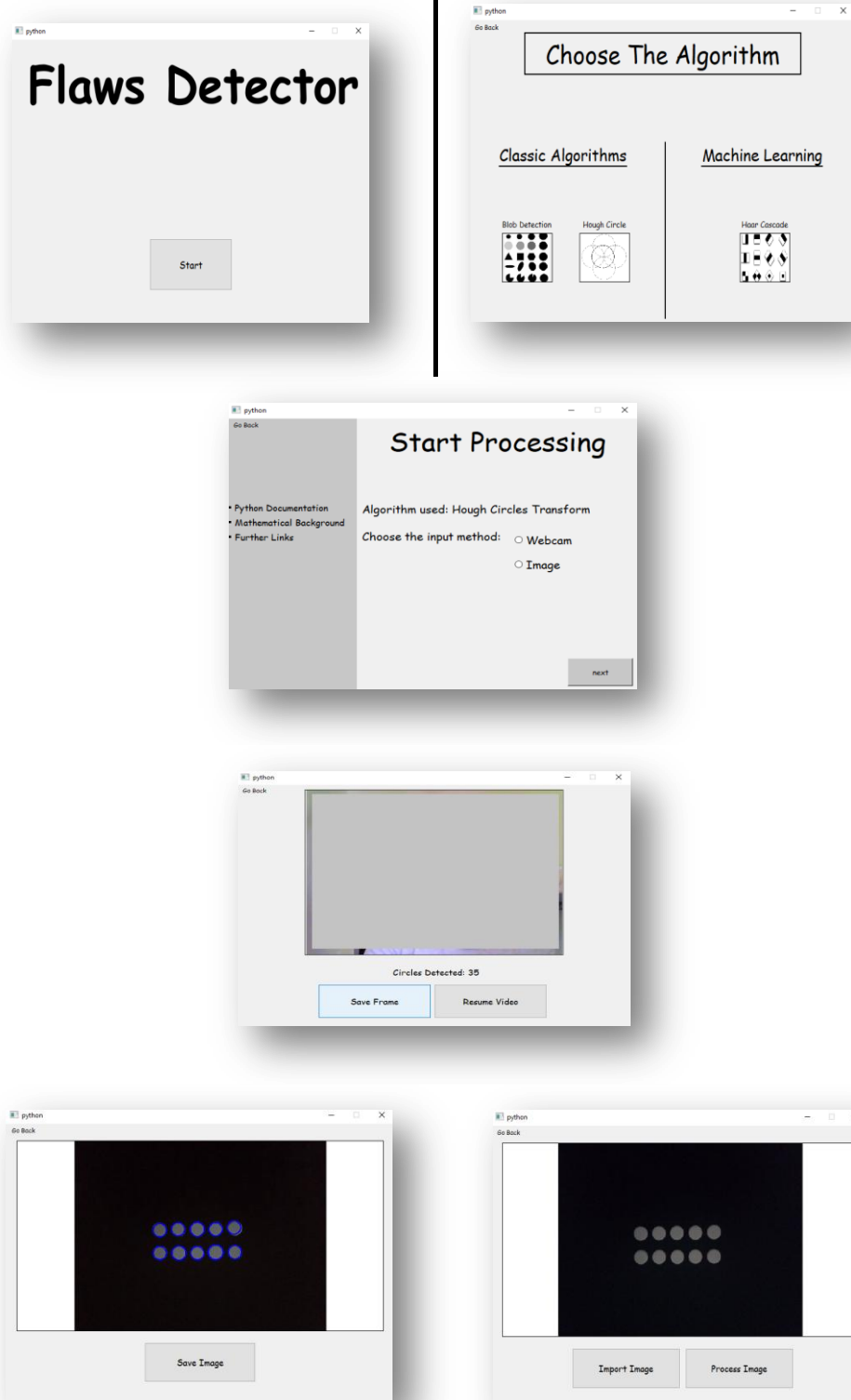


Fig 4.9: GUI Application Screenshots

IV.5. Further Realization:

As our desktop application is executing, if a missing pill is detected, we can conduct signals to a servomotor to eliminate the flawed tablet off the manufacturing cycle.

Perquisites:

The application is to be running on a Raspberry Pi, which has a servomotor connected on one of its pins.

IV.5.1. Raspberry Pi:

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside its target market for uses such as robotics. It does not include peripherals (such as keyboards and mice) and cases. However, some accessories have been included in several official and unofficial bundles[14].

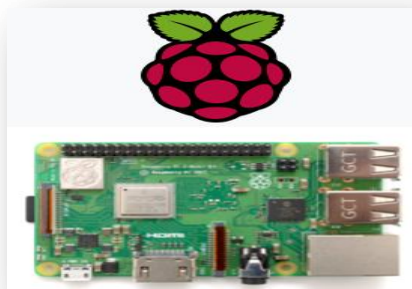


Fig 4.10: Raspberry Pi

IV.5.2. Servomotors with Raspberry Pi:



Fig 4.11: Servomotor

Servo control is achieved by sending a servo a PWM (pulse-width modulation) signal, a series of repeating pulses of variable width where either the width of the pulse (most common modern hobby servos) or the duty cycle of a pulse train (less common today) determines the position to

be achieved by the servo. The PWM signal might come from a radio control receiver to the servo or from common microcontrollers such as the Raspberry Pi[15].

Flowchart:

The overall flowchart will be modified as follows:

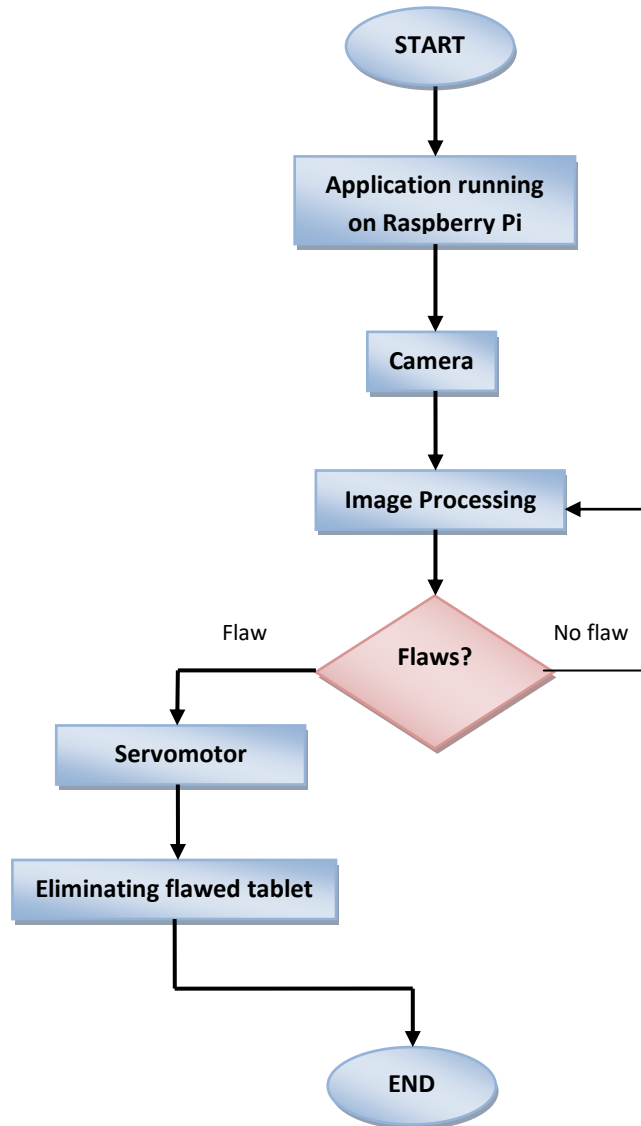


Fig 4.12: Further Realization Flowchart

IV.6. Summary:

As a sum up for the experimental part, changing any of the parameters included in each classic algorithm reduced the detection efficiency. However machine learning stands out due to its independent processing.

Conclusion

This project combined the fields of image processing, machine learning, and robotics in order to provide a solution (or more of a correction) for detecting flaws in medical pill manufacturing using computer vision algorithms in industrial applications.

We have been through a general introduction about image processing, some of the most used algorithms such as Canny edge and Hough Circle Transform, machine learning, and finally an implementation of resulting the count of pills in an image.

In our implementation, we have developed three experiments (cascade of algorithms) to detect and count the number of pills compressed in a medical tablet, which were interfaced using a GUI desktop application that runs on multiplatform devices.

After every experiment, we conclude that classic algorithms are lighter but stay less accurate due to their dependence on many parameters, where machine learning provided more efficiency overall.

During this project, we learned image processing and its efficient applications, with some of its powerful algorithms used in computer vision. We also learned how to use programming to implement these algorithms, plus developing a user-friendly cross-platform GUI desktop application.

To improve our work, the algorithms used can be developed in order to be more exact and less sensitive to external factors. We also can detect more shapes and add other features such as sorting by colors or textures.

Reference

- [1] <https://www.coursera.org/lecture/digital/electromagnetic-spectrum-qjIY0>
- [2] Mestel, Rosie (2002-03-25). "The Colorful History of Pills Can Fill Many a Tablet". latimes. Archived from the original on 2015-09-19.
- [3] <https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm>
- [4] https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html
- [5] https://docs.opencv.org/master/da/d53/tutorial_py_houghcircles.html
- [6] <https://www.learnopencv.com/blob-detection-using-opencv-python-c/>
- [7] <https://searchenterpriseai.techtarget.com/definition/machine-learning-ML>
- [8] https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html
- [9] <https://amin-ahmadi.com/cascade-trainer-gui/>
- [10] <https://pythonprogramming.net/>
- [11] <https://opencv.org/>
- [12] <https://scikit-learn.org/>
- [13] <https://riverbankcomputing.com/software/pyqt/intro>
- [14] <https://www.raspberrypi.org/>
- [15] <https://tutorials-raspberrypi.com/raspberry-pi-servo-motor-control/>