

**Ministry of Higher Education**

**Misr Higher Institute for Commerce & Technology**

**Computer Science Department**

**4th Year Students Graduation Project**

**Title**

**Hand Gestures Recognition**

(HGR)

**Prepared By**

**Ahmad Abd-Elghany**

**Under Supervision Of**

**Dr. Mahmoud Abo-Elfotoh**

**And**

**ENG. Adham Shafeek**

June 2014

****

# Abstract

The development of Human Computer Interface (HCI) is continuously pursuing a user-friendly interface and simplification system. There are some innovative breakthroughs recently. The traditional keyboard input has been replaced gradually since many new methods had been invented. Like Gesture recognition devices from sensors to depth cameras.

The body sense technology even gets rid of restriction of input device; make the HCI closer to human's nature action. The most important part of it will be manipulation using hand. Hence, we proposed a system which is accurate in hand gesture recognition that know exactly the gesture then perform some actions based on these gestures.

In this project, we proposed an image processing system using a web camera. Using hand recognition method, we are trying to transfer the gesture to some certain instructions. We mark up the important features of hand:  fingertips, hand contours, providing real-time interaction between gesture and the system.

# Table of Contents

Chapter 1 Introduction

* 1. Overview ……………………………………………………………………….. 1
  2. Motivation ……………………………………………………………………… 2
  3. Vision Systems ...……………………………………………………………..…. 2
  4. Related fields …………………………………………………………………… 3
  5. Recognition Tasks ………………………………………………………………. 7
  6. HCI ……………………………………………………………………………… 10
  7. The Project Use …………………………………………………………………. 11

## Chapter 2 | Planning and Requirements

* 1. **Typical Vision System** ………………………………………………………….. 13
  2. **Vision System Diagram** ………………………………………………………… 15
  3. **Application Diagram** ……………………………………………………………..16
  4. Libraries ……………………………………………………………………….....,17

## **Chapter 3 | System Analysis and Design**

* 1. System Description ………………………………………………………………. 22
  2. System Diagram ………………………………………………………………….. 23
  3. System UI ………………………………………………………………………… 24
  4. UI Description ……………………………………………………………………. 25
  5. Low-Level Design ………………………………………………………………... 26
     1. Skin Detection …………………………………………………………………. 26
     2. Hand Detection ………………………………………………………………… 27
     3. Convex Hull ……………………………………………………………………. 28
  6. 7-Segment Circuit ………………………………………………………………… 29

## Chapter 4 | Implementation

* 1. Introduction ………………………………………………………………………… 31
  2. Problems Faced …………………………………………………………………….. 32
  3. System Functionality ……………………………………………………………….. 32
  4. Flow Chart ………………………………………………………………………….. 33
  5. Functionality …………………………………………………………………………34
     1. Input …………………………………………………………………………….... 34
     2. Skin Detection ……………………………………………………………………. 34
     3. Face Removal …………………………………………………………………….. 37
     4. Canny Edge Detector ……………………………………………………………... 41
     5. Convex Hull ………………………………………………………………………. 45
     6. Image Moments …………………………………………………………………… 46
  6. Output ………………………………………………………………………………… 48
  7. C# Code ………………………………………………………………………………. 49

## Conclusion

1. Conclusion …………………………………………………………………………………………………………………………. 57

CHAPTER ONE  
**INTRODUCTION**

# Chapter 1 | Introduction

# 1 Overview

In recent years, the HCI has become more and more widespread and diversified. Digital devices nowadays are transforming into many different types, smart phones and tablet computer are the current trend. They change the way how people use Computer. Also the appearance of Microsoft Kinect and Wii redefined our knowledge of manipulation interface. The traditional input devices like keyboard and mouse are being replaced.

In the future, computers won’t be just computers. Many daily devices will be integrated with computers, like cars, television, household appliances, etc. Since so many types of machines will come up, we need various HCIs to cope with different type of machines. The hand gesture will be the most popular one. It is the most intuitive way for human to communicate with their body.

With the communication by hand, people will no longer need to use computer devices in a certain location. For example, if a TV is integrated with gesture recognition function, people can manipulate it everywhere in their house. A remote control will be needless. The gap between human and machines will be closer. It increases the freedom of usability. Another example will be the outdoor interactive advertising board. More and more related products will appear and the gesture Interface will be more important.

1. 2 **Motivation**

As previous section described, we believe the hand gesture interface will be a future trend. It breaks the restriction of devices and space. The most important features of hand gesture will be fingertips and palm. Most research in hand gesture recognition try to convert a certain hand gesture into a particular symbol. In those cases, hands can only perform several simple instructions.

1.3 **Vision Systems**

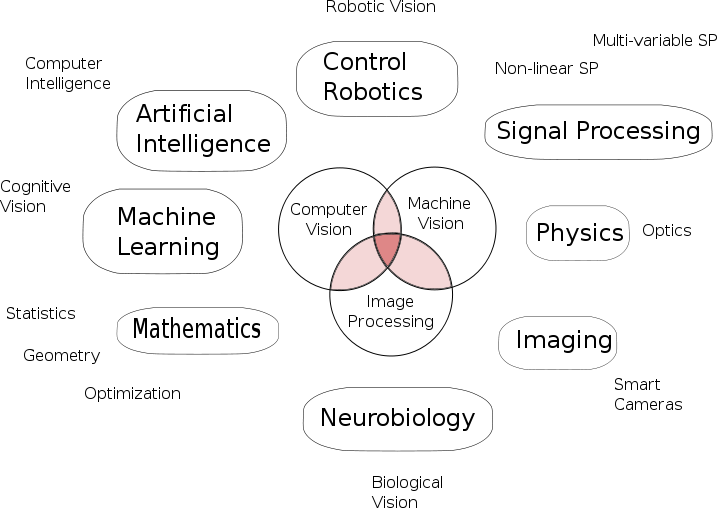
Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory. Computer vision has also been described as the enterprise of automating and integrating a wide range of processes and representations for vision perception.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, learning, indexing, motion estimation, and image restoration.

## 1.4 **Related fields**

Areas of artificial intelligence deal with autonomous planning or deliberation for robotical systems to navigate through an environment. A detailed understanding of these environments is required to navigate through them. Information about the environment could be provided by a computer vision system, acting as a vision sensor and providing high-level information about the environment and the robot.

Artificial intelligence and computer vision share other topics such as pattern recognition and learning techniques. Consequently, computer vision is sometimes seen as a part of the artificial intelligence field or the computer science field in general.

Solid-state physics is another field that is closely related to computer vision. Most computer vision systems rely on [image sensors](http://en.wikipedia.org/wiki/Image_sensors), which detect electromagnetic radiation which is typically in the form of either visible or infra-red light. The sensors are designed using [quantum physics](http://en.wikipedia.org/wiki/Quantum_physics). The process by which light interacts with

surfaces is explained using physics.

Figure 1 Relation between computer vision and various other fields

Physics explains the behavior of [optics](http://en.wikipedia.org/wiki/Optics) which are a core part of most imaging systems. Sophisticated [image sensors](http://en.wikipedia.org/wiki/Image_sensors) even require quantum mechanics to provide a complete understanding of the image formation process. Also, various measurement problems in physics can be addressed using computer vision, for example motion in fluids.

A third field which plays an important role is [neurobiology](http://en.wikipedia.org/wiki/Neurobiology), specifically the study of the biological vision system. Over the last century, there has been an extensive study of eyes, neurons, and the brain structures devoted to processing of visual stimuli in both humans and various animals. This has led to a coarse, yet complicated, description of how "real" vision systems operate in order to solve certain vision related tasks. These results have led to a subfield within computer vision where artificial systems are designed to mimic the processing and behavior of biological systems, at different levels of complexity. Also, some of the learning-based methods developed within computer vision (*e.g.* [neural net](http://en.wikipedia.org/wiki/Neural_network) and [deep learning](http://en.wikipedia.org/wiki/Deep_learning) based image and feature analysis and classification) have their background in biology.

Some strands of computer vision research are closely related to the study of [biological vision](http://en.wikipedia.org/wiki/Biological_vision) – indeed, just as many strands of AI research are closely tied with research into human consciousness, and the use of stored knowledge to interpret, integrate and utilize visual information. The field of biological vision studies and models the physiological processes behind visual perception in humans and other animals. Computer vision, on the other hand, studies and describes the processes implemented in software and hardware behind artificial vision systems. Interdisciplinary exchange between biological and computer vision has proven fruitful for both fields.

Yet another field related to computer vision is [signal processing](http://en.wikipedia.org/wiki/Signal_processing). Many methods for processing of one-variable signals, typically temporal signals, can be extended in a natural way to processing of two-variable signals or multi-variable signals in computer vision. However, because of the specific nature of images there are many methods developed within computer vision which have no counterpart in processing of one-variable signals. Together with the multi-dimensionality of the signal, this defines a subfield in signal processing as a part of computer vision.

Beside the above mentioned views on computer vision, many of the related research topics can also be studied from a purely mathematical point of view. For example, many methods in computer vision are based on [statistics](http://en.wikipedia.org/wiki/Statistics), [optimization](http://en.wikipedia.org/wiki/Optimization_%28mathematics%29) or [geometry](http://en.wikipedia.org/wiki/Geometry). Finally, a significant part of the field is devoted to the implementation aspect of computer vision; how existing methods can be realized in various combinations of software and hardware, or how these methods can be modified in order to gain processing speed without losing too much performance.

The fields most closely related to computer vision are [image processing](http://en.wikipedia.org/wiki/Image_processing), [image analysis](http://en.wikipedia.org/wiki/Image_analysis) and [machine vision](http://en.wikipedia.org/wiki/Machine_vision). There is a significant overlap in the range of techniques and applications that these cover. This implies that the basic techniques that are used and developed in these fields are more or less identical, something which can be interpreted as there is only one field with different names. On the other hand, it appears to be necessary for research groups, scientific journals, conferences and companies to present or market themselves as belonging specifically to one of these fields and, hence, various characterizations which distinguish each of the fields from the others have been presented.

Computer vision is, in some ways, the inverse of [computer graphics](http://en.wikipedia.org/wiki/Computer_graphics). While computer graphics produces image data from 3D models, computer vision often produces 3D models from image data. There is also a trend towards a combination of the two disciplines, *e.g.*, as explored in [augmented reality](http://en.wikipedia.org/wiki/Augmented_reality).

The following characterizations appear relevant but should not be taken as universally accepted:

* [Image processing](http://en.wikipedia.org/wiki/Image_processing) and [image analysis](http://en.wikipedia.org/wiki/Image_analysis) tend to focus on 2D images, how to transform one image to another, *e.g.*, by pixel-wise operations such as contrast enhancement, local operations such as edge extraction or noise removal, or geometrical transformations such as rotating the image. This characterization implies that image processing/analysis neither require assumptions nor produce interpretations about the image content.
* Computer vision includes 3D analysis from 2D images. This analyzes the 3D scene projected onto one or several images, *e.g.*, how to reconstruct structure or other information about the 3D scene from one or several images. Computer vision often relies on more or less complex assumptions about the scene depicted in an image.
* [Machine vision](http://en.wikipedia.org/wiki/Machine_vision) is the process of applying a range of technologies & methods to provide imaging-based automatic inspection, process control and robot guidance in industrial applications. Machine vision tends to focus on applications, mainly in manufacturing, *e.g.*, vision based autonomous robots and systems for vision based inspection or measurement. This implies that image sensor technologies and control theory often are integrated with the processing of image data to control a robot and that real-time processing is emphasized by means of efficient implementations in hardware and software. It also implies that the external conditions such as lighting can be and are often more controlled in machine vision than they are in general computer vision, which can enable the use of different algorithms.
* There is also a field called [imaging](http://en.wikipedia.org/wiki/Imaging_science) which primarily focus on the process of producing images, but sometimes also deals with processing and analysis of images. For example, [medical imaging](http://en.wikipedia.org/wiki/Medical_imaging) includes substantial work on the analysis of image data in medical applications.
* Finally, [pattern recognition](http://en.wikipedia.org/wiki/Pattern_recognition) is a field which uses various methods to extract information from signals in general, mainly based on statistical approaches and [artificial neural networks](http://en.wikipedia.org/wiki/Artificial_neural_networks). A significant part of this field is devoted to applying these methods to image data.

1.5 **Recognition Tasks**

The classical problem in computer vision, image processing, and machine vision is that of determining whether or not the image data contains some specific object, feature, or activity. This task can normally be solved [robustly](http://en.wiktionary.org/wiki/robust) and without effort by a human, but is still not satisfactorily solved in computer vision for the general case – arbitrary objects in arbitrary situations. The existing methods for dealing with this problem can at best solve it only for specific objects, such as simple geometric objects (*e.g.*, polyhedra), human faces, printed or hand-written characters, or vehicles, and in specific situations, typically described in terms of well-defined illumination, background, and [pose](http://en.wikipedia.org/wiki/Pose_%28computer_vision%29) of the object relative to the camera.

Different varieties of the recognition problem are described in the literature:

* [**Object recognition**](http://en.wikipedia.org/wiki/Object_recognition) – one or several pre-specified or learned objects or object classes can be recognized, usually together with their 2D positions in the image or 3D poses in the scene. [Google Goggles](http://en.wikipedia.org/wiki/Google_Goggles) provides a stand-alone program illustration of this function.
* **Identification** – an individual instance of an object is recognized. Examples include identification of a specific person's face or fingerprint, identification of [handwritten digits](http://en.wikipedia.org/wiki/MNIST_database), or identification of a specific vehicle.
* **Detection** – the image data are scanned for a specific condition. Examples include detection of possible abnormal cells or tissues in medical images or detection of a vehicle in an automatic road toll system. Detection based on relatively simple and fast computations is sometimes used for finding smaller regions of interesting image data which can be further analyzed by more computationally demanding techniques to produce a correct interpretation.

Several specialized tasks based on recognition exist, such as:

* [**Content-based image retrieval**](http://en.wikipedia.org/wiki/Content-based_image_retrieval) – finding all images in a larger set of images which have a specific content. The content can be specified in different ways, for example in terms of similarity relative a target image (give me all images similar to image X), or in terms of high-level search criteria given as text input (give me all images which contains many houses, are taken during winter, and have no cars in them).

\* [**Pose estimation**](http://en.wikipedia.org/wiki/Pose_%28computer_vision%29) – estimating the position or orientation of a specific object relative to the camera. An example application for this technique would be assisting a robot arm in retrieving objects from a conveyor belt in an [assembly line](http://en.wikipedia.org/wiki/Assembly_line) situation or picking parts from a bin.

* [**Optical character recognition**](http://en.wikipedia.org/wiki/Optical_character_recognition) (OCR) – identifying [characters](http://en.wikipedia.org/wiki/Character_%28computing%29) in images of printed or handwritten text, usually with a view to encoding the text in a format more amenable to editing or [indexing](http://en.wikipedia.org/wiki/Search_index) (*e.g.* [ASCII](http://en.wikipedia.org/wiki/ASCII)).
* **2D Code reading** Reading of 2D codes such as [data matrix](http://en.wikipedia.org/wiki/Data_Matrix) and [QR](http://en.wikipedia.org/wiki/QR_code) codes.
* [**Facial recognition**](http://en.wikipedia.org/wiki/Facial_recognition_system)
* [**Shape Recognition Technology**](http://en.wikipedia.org/wiki/Pattern_recognition) (SRT) in [people counter](http://en.wikipedia.org/wiki/People_counter) systems differentiating human beings (head and shoulder patterns) from objects

### Motion analysis

Several tasks relate to motion estimation where an image sequence is processed to produce an estimate of the velocity either at each points in the image or in the 3D scene, or even of the camera that produces the images. Examples of such tasks are:

* [**Egomotion**](http://en.wikipedia.org/wiki/Egomotion) – determining the 3D rigid motion (rotation and translation) of the camera from an image sequence produced by the camera.
* [**Tracking**](http://en.wikipedia.org/wiki/Video_tracking) – following the movements of a (usually) smaller set of interest points or objects (*e.g.*, vehicles or humans) in the image sequence.
* [**Optical flow**](http://en.wikipedia.org/wiki/Optical_flow) – to determine, for each point in the image, how that point is moving relative to the image plane, i.e., its apparent motion. This motion is a result both of how the corresponding 3D point is moving in the scene and how the camera is moving relative to the scene.

### Scene reconstruction

Given one or (typically) more images of a scene, or a video, scene reconstruction aims at computing a [3D model](http://en.wikipedia.org/wiki/Computer_model) of the scene. In the simplest case the model can be a set of 3D points. More sophisticated methods produce a complete 3D surface model. The advent of 3D imaging not requiring motion or scanning, and related processing algorithms is enabling rapid advances in this field. Grid-based 3D sensing can be used to acquire 3D images from multiple angles. Algorithms are now available to stitch multiple 3D images together into point clouds and 3D models.

### Image restoration

The aim of image restoration is the removal of noise (sensor noise, motion blur, etc.) from images. The simplest possible approach for noise removal is various types of filters such as low-pass filters or median filters. More sophisticated methods assume a model of how the local image structures look like, a model which distinguishes them from the noise. By first analyzing the image data in terms of the local image structures, such as lines or edges, and then controlling the filtering based on local information from the analysis step, a better level of noise removal is usually obtained compared to the simpler approaches.

An example in this field is the [inpainting](http://en.wikipedia.org/wiki/Inpainting).

1.6 **HCI**

**Human–computer interaction** (**HCI**) involves the study, planning, design and uses of the interaction between people ([users](http://en.wikipedia.org/wiki/User_%28computing%29)) and computers. It is often regarded as the intersection of [computer science](http://en.wikipedia.org/wiki/Computer_science), [behavioral sciences](http://en.wikipedia.org/wiki/Behavioral_sciences), design and [several other fields of study](http://en.wikipedia.org/wiki/Outline_of_human%E2%80%93computer_interaction#Related_fields). The term was popularized by Card, Moran, and Newell in their seminal 1983 book, *The Psychology of Human-Computer Interaction*, although the authors first used the term in 1980, and the first known use was in 1975. The term connotes that, unlike other tools with only limited uses (such as a hammer, useful for driving nails, but not much else), a computer has many affordances for use and this takes place in an open-ended dialog between the user and the computer.

Because human–computer interaction studies a human and a machine in conjunction, it draws from supporting knowledge on both the machine and the human side. On the machine side, techniques in [computer graphics](http://en.wikipedia.org/wiki/Computer_graphics), [operating systems](http://en.wikipedia.org/wiki/Operating_systems), [programming languages](http://en.wikipedia.org/wiki/Programming_language), and development environments are relevant. On the human side, [communication theory](http://en.wikipedia.org/wiki/Communication_theory), graphic and [industrial design](http://en.wikipedia.org/wiki/Industrial_design) disciplines, [linguistics](http://en.wikipedia.org/wiki/Linguistics), [social sciences](http://en.wikipedia.org/wiki/Social_sciences), [cognitive psychology](http://en.wikipedia.org/wiki/Cognitive_psychology), [social psychology](http://en.wikipedia.org/wiki/Social_psychology), and [human factors](http://en.wikipedia.org/wiki/Human_factors) such as [computer user satisfaction](http://en.wikipedia.org/wiki/Computer_user_satisfaction) are relevant. Engineering and design methods are also relevant. Due to the multidisciplinary nature of HCI, people with different backgrounds contribute to its success. HCI is also sometimes referred to as human–machine interaction (HMI), man–machine interaction (MMI) or computer–human interaction (CHI).

Poorly designed [human-machine interfaces](http://en.wikipedia.org/wiki/Human-machine_interface) can lead to many unexpected problems. A classic example of this is the [Three Mile Island accident](http://en.wikipedia.org/wiki/Three_Mile_Island_accident), a nuclear meltdown accident, where investigations concluded that the design of the human–machine interface was at least partially responsible for the disaster. Similarly, accidents in aviation have resulted from manufacturers' decisions to use non-standard [flight instrument](http://en.wikipedia.org/wiki/Flight_instruments#Layout) or throttle quadrant layouts: even though the new designs were proposed to be superior in regards to basic human–machine interaction, pilots had already ingrained the "standard" layout and thus the conceptually good idea actually had undesirable results.

1.7 **The Project Use:**

* **Computer Vision**
* **HCI**
* **Hardware Controller**

# CHAPTER TWO

# PLANNING

# AND

# REQUIREMENTS

# Chapter 2 | Planning and Requirements

# 2.1 Typical Vision System

Typically, a machine vision system is PC-based, using a group of devices to receive, analyze and interpret the image of a real scene. The system makes judgments on the image using predefined criteria set by the user. This information can be used to automate go/no-go inspection decisions, assembly verification, part location and machine guidance, gaging/dimensional measurements, feedback control loops and a host of other tasks.

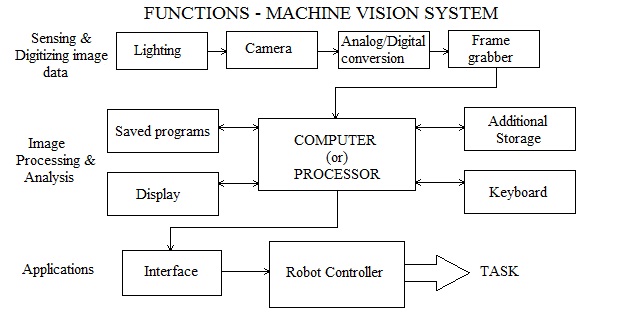
It is a common misperception that machine vision systems provide generic optical detection and processing capabilities. While every system includes essential functions, most customers require some level of customization in development and should be cautious of vendors claiming to have "one-size-fits-all" solutions. Systems perform best in their own tightly controlled, highly specialized environment.

Figure 2 Vision System

Application requirements vary drastically by industry, but a number of components are common to every machine vision system. Technology is evolving rapidly in all these areas, creating new opportunities on the manufacturing floor.

The following are common components:

http://www.qualitydigest.com/oct97/assets/images/Qdbullet.gif*Cameras* -- CCD cameras are becoming smaller, lighter and less expensive. Images are sharper and more accurate, and the new dual output cameras produce images twice as fast as previous models. A new generation of CCD color cameras adds another dimension to machine vision by enabling systems to better detect and discriminate between objects, remove backgrounds and perform spectral analysis.

http://www.qualitydigest.com/oct97/assets/images/Qdbullet.gif*PCs* -- With the advent of the PCI bus, the PC has had a major impact on the use of machine vision in manufacturing applications. Personal computers up to then couldn't gather data at a rate fast enough to keep up with machine vision's heavy      I/O requirements, including data transfer rates of 20 MB/second or greater. The VME bus, a specialized architecture for data acquisition and process control, with bus speeds of 40 MB/second, became a development standard instead. However, today's PCs can handle machine vision's demands, with 132 MB/second PCI bus transfer speeds and >100 MHz Pentium microprocessors. PCs are now routinely embedded into equipment on the factory floor. The distributed intelligence made possible by PC technology has contributed immeasurably to the pace and effectiveness of factory automation.

http://www.qualitydigest.com/oct97/assets/images/Qdbullet.gif*Software* -- Graphical user interfaces and libraries of high-level software modules operating in standard environments such as Windows have eased the development process and made machine vision a user-friendly tool. Leading-edge software suppliers have begun to provide object-oriented application development tools that will speed application development even more.

*2.2* ***Vision System Diagram***

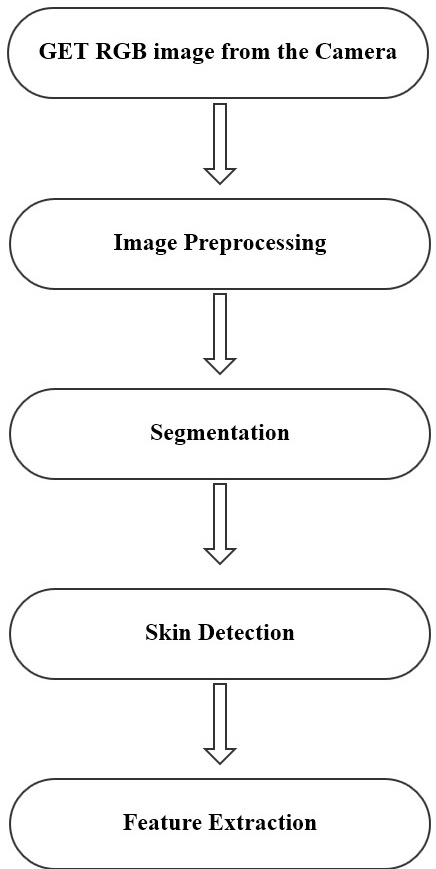


Figure 3 Vision System Diagram

2.4 **Libraries**

1 : **OpenCV** :-

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

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding [7 million](http://sourceforge.net/projects/opencvlibrary/files/stats/timeline?dates=2001-09-20+to+2013-09-26). The library is used extensively in companies, research groups and by governmental bodies.

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

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

2 : **EmguCV** :-

[Emgu](http://www.emgu.com/wiki/index.php/Main_Page) opens the OpenCV (Open Source Computer Vision Library) library of programming functions mainly aimed at real time computer vision to .NET developers. OpenCV was originally developed by Intel and now supported by Willow Garage.

Current versions for x86 , x64 and ARM(only for iOS and Android) architectures are available for download at their SourceForge website,

A download and installation guide is available [here](http://www.emgu.com/wiki/index.php/Download_And_Installation). This is particularly useful for newcomers as ensuring the correct references are used and that the associated OpenCV .dll files are copied to the output directory can be difficult.

Emgu CV uses a dual-license business model for its software development library and offers licenses for two distinct purposes: open source, and commercial development.

**Cross Platform**

Unlike other wrappers such as OpenCVDotNet, SharperCV or Code Project which use unsafe code, Emgu CV is written entirely in C#. The benefit is that it can be compiled in Mono and therefore is able to run on any platform Mono supports, including Linux, Solaris and Mac OS X. A lot of efforts have been spent creating a pure C# implementation since the headers have to be ported, compared with managed C++ implementation where header files can simply be included. But it is well worth it if you see Emgu CV running on Fedora 10! Plus it always gives you the comfort knowing that your code is cross-platform.

Emgu CV can be used from several different languages, including C#, VB.NET, C++ and IronPython. On this wiki, we provide examples for all those languages, which are available from the Examples section on Tutorial page. Our discussion forum is also available if you have any questions related to your favorite programming language.

**Other Advantages**

* Image class with Generic Color and Depth
* Automatic garbage collection
* XML Serializable Image
* XML Documentation and intellisense support
* The choice to either use the Image class or direct invoke functions from OpenCV
* Generic operations on image pixels.

**Architecture Overview**

Emgu CV has two layers of wrapper as shown below

* The basic layer (layer 1) contains function, structure and enumeration mappings which directly reflect those in OpenCV.
* The second layer (layer 2) contains classes that mix in advantanges from the .NET world.

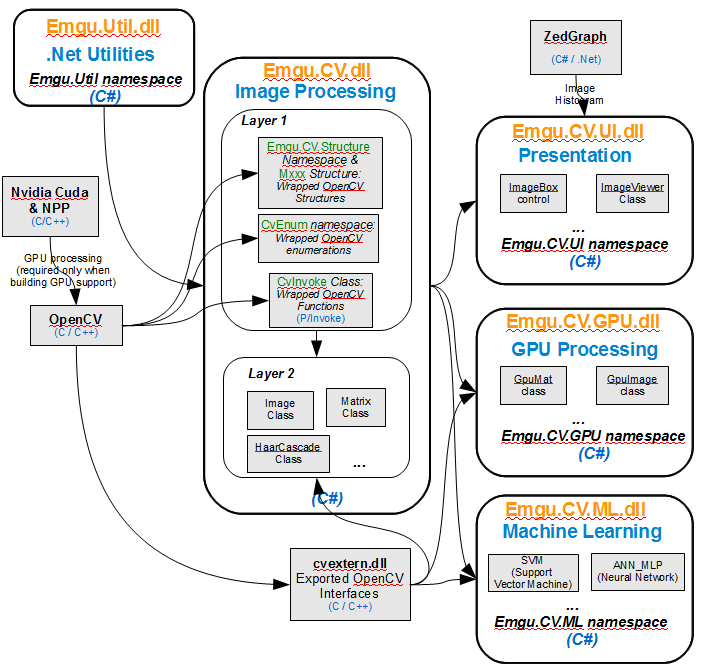


Figure 5 EmguCV Architecture

# CHAPTER THREE

# SYSTEM ANALYSIS

# and

# DESIGN

# Chapter 3 | System Analysis and Design

3.1 **System Description**

Our system and its computation are based on the contour of hand. In our system, we assume that the input contour is the contour of hand. We are not performing any hand or arm recognition.

The original input data will be the RGB image captured by a web camera. We perform some preliminary processing to generate a binary image which provides enough information of hand contour.

Then we detect the face in the original image and remove the face.

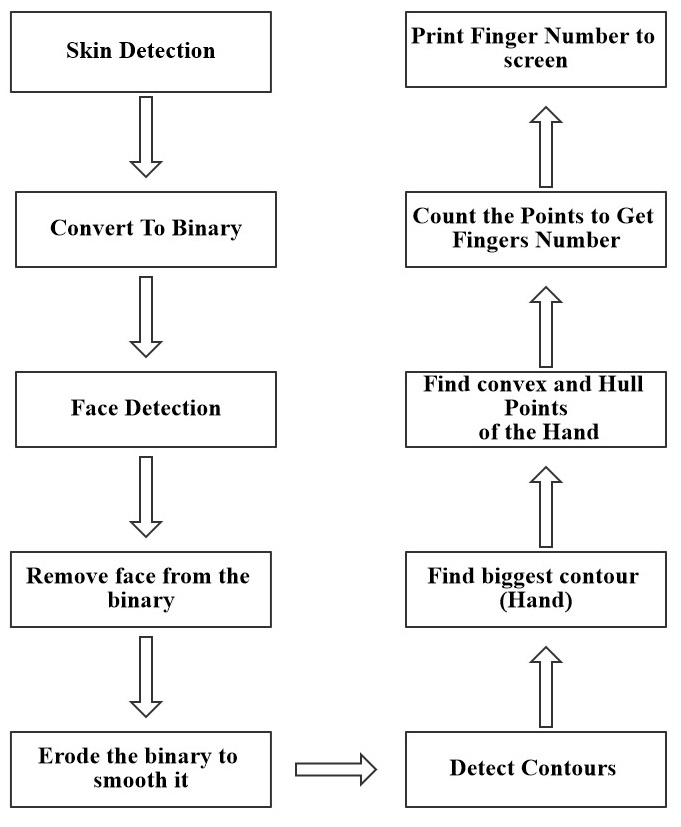
The face then is removed from the binary as well.

The next step is to erode it to get a perfect binary image. Then we detect contours and look for the biggest contour. We assume that the highest contour we get is none other than the hand.

Now we take the square out from the image containing the hand. Then we find hull and convex points of the hand. These points help us counting the finger.

3.2 **System Diagram**

Figure 6 System Diagram



2.3 **Application Diagram**

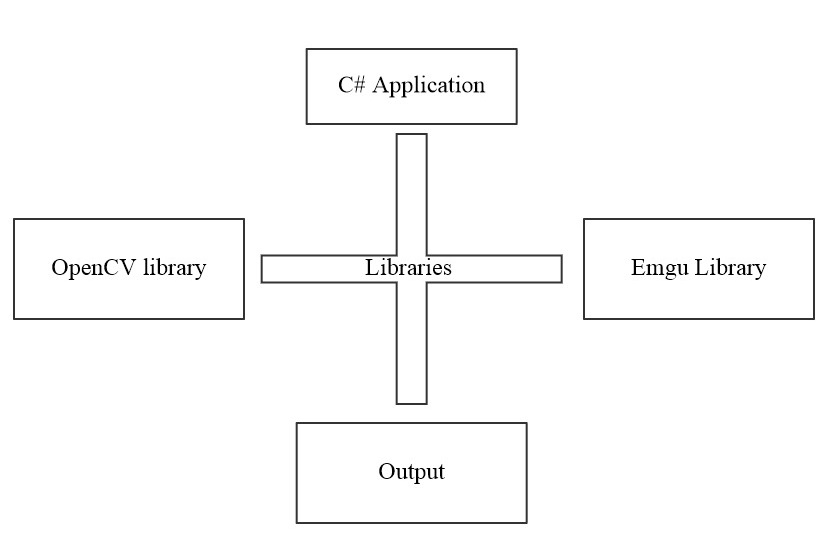


Figure 4 Application Diagram

3.3 **System UI**

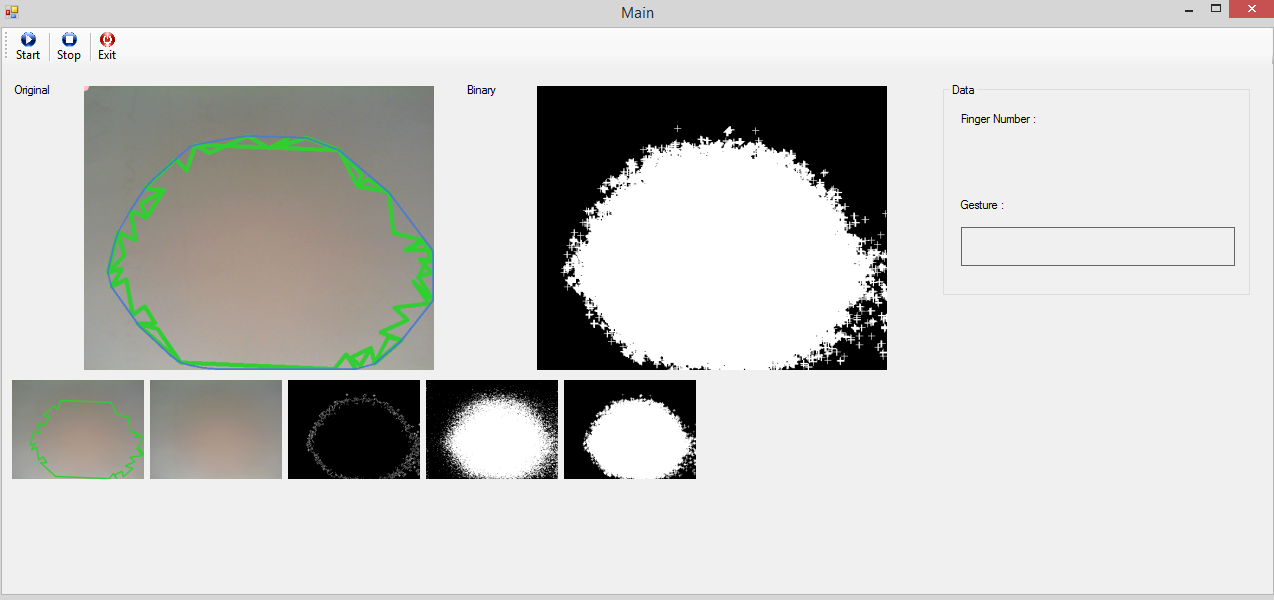


Figure 7 System UI

*3.4* ***UI Description***

The User Interface of the system is simple, first we have three main Buttons and they are self-described :

1. Start
2. Stop
3. Exit

Then we have a panel contains the bitmap images that responsible for the different camera streams :

1. The original Image after performing the different tasks.
2. The binary stream of the image.
3. Different 6 streams showing every individual task that had performed on the original image.

Then we have a container for displaying the data that we want to print out to the user :

1. Label that displaying the finger number in the image.
2. And label for displaying the text depends on the finger number

*3.5* ***Low-Level Design***

This level is the level where we explode or break down the process into smaller ones.

We will examine the skin detection, face detection and the convex hull processes because these are the processes where the automation and most of the project work will be take place.

*3.5.A* ***Skin Detection***

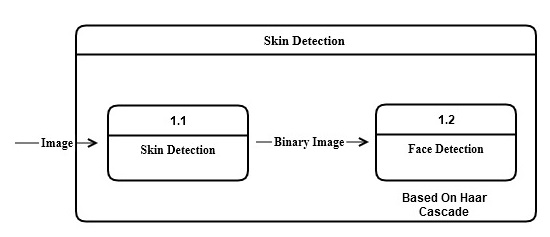


Figure 8 (Low Level Diagram – level 1)

After that we remove the face from both, the original image and the binary, this phase is important as it give is the power to calculate the hand contour accurately.

*3.5.B* ***Hand Detection***

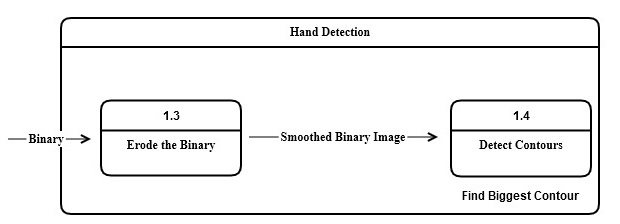


Figure 9 (Low Level Diagram – level 2)

In this cycle, after blurring the image to smooth it for the behalf of having a good shaped binary image. We detect contours, and start looking for the biggest contour assuming that the highest one is none other the hand.

After that we take a square out from the image containing the hand, note that this is useless step but we made it for the sake of clarity.

*3.5.C* ***Convex Hull***

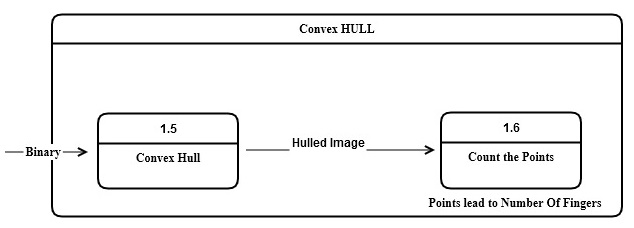


Figure 10 (Low Level Diagram – level 3)

At this phase, we get to the end of the levels, we now find the convex hull points of the hand based on the contour we found earlier.

After finding them, we count the points for the sake of finding fingers numbers, then we done with the Low Level phases.

After, finding the finger number we can now send any message the screen or to any USB cable to switch the light on for instance, the number of tasks that we can perform based on this system is countless.

*3.6* ***7-Segment Circuit***

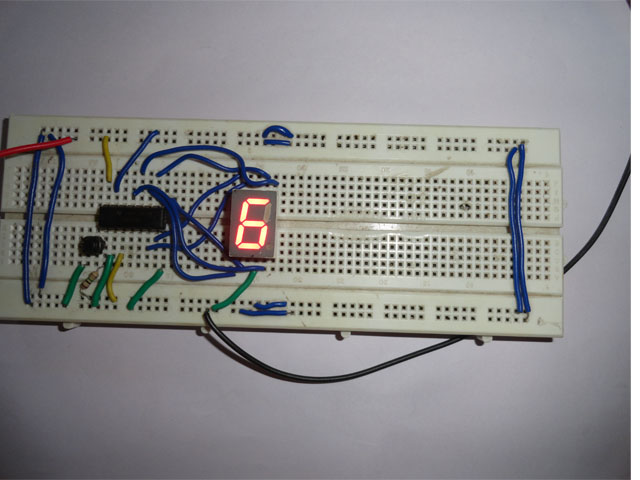


Figure 11 (7-Segment circuit)

We used a 7-Segement Circuit to output the finger number from the system,

From this perspective we can develop any circuit and attached it to the system, and then we could use it to help the handicapped people to interact with the outer world, and the computer itself.

# CHAPTER FOUR

# IMPLEMENTATION

# Chapter 4 | Implementation

*4.1* ***Introduction***

In this project different techniques have being studied like color based detection, face detection using HAAR cascade, feature extraction and Convex Hull.

For detection, Color based technique was implemented, which depends on the

Detection of the human skin color with all its different variations in the image. Firstly, we detected the skin using Color based Technique, in this technique the image converted into a new color space, YCrCb.

The new image is scanned pixel by pixel and if each pixel satisfies the

Conditions, which is the range of the human skin, its value is set to white else it is set to black.

Then we detect the face using HAAR cascade method, after we recognize the face we remove it from the image by implementing an oval shape over it.

Then the resultant image is converted into black white color space in which pixels have values of 0 and 1 only (black and white). The black white image is sent to the Canny Edges function which is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. For that we detect the contour of the hand.

Finally we send the resulting image to the Convex Hull function, which detect the contour and paint points over it, with these points we can calculate the fingers number accurately, and then we can print the results on the screen and to the stream.

*4.2* ***Problems Faced***

After applying this technique there was a problem that there was noise that

Tremendously affected the segmentation process so a noise removal filter was implemented to successfully eliminate this problem. We’ve used Erode and Dilate filters to smooth the edges and get clean binary image.

*4.3* ***System Functionality***

******

Figure 12 System Functions

*4.4* ***Flow Chart***

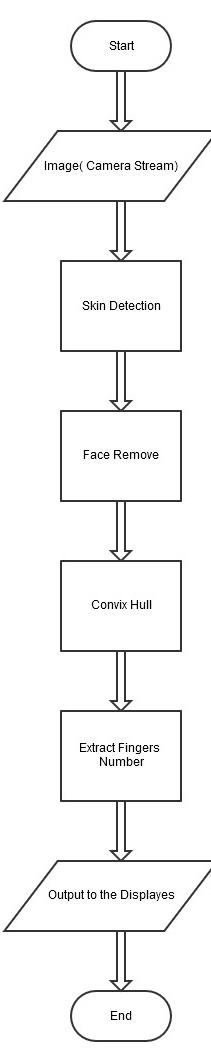


Figure 13 Flow Chart of the system

*4.5* ***Functionality***

*4.5.1* ***Input***

Firstly, we started to capture the stream from an ordinary webcam. We Used the Capture variable from Emgu library to handle that, Fortunate enough the Emgu has made this step so easy, it’s automatically loop over the machine’s attached devices then stored them with indexes, so simple enough we just type it like this :

Capture capture = new Capture(0); //create a camera capture

This is basically the input of our system.

*4.5.2* ***Skin Detection***

**YCbCr** Is one of two primary color spaces used to represent digital component video (the other is RGB). The difference between YCbCr and RGB is that YCbCr represents color as brightness and two color difference signals, while RGB represents color as red, green and blue. In YCbCr, the Y is the brightness (luma), Cb is blue minus luma (B-Y) and Cr is red minus luma (R-Y).

We started by taking the original image and converted it to YCbCr using the help of Emgu, we wrapped the code into a method that takes the arguments of skin values that fall between :

(Ycc)min, (Ycc)max)

After that, we’ve implemented the Erode and Dilate to smooth the edges and to get clean binary image.

Based on OpenCV, the definition of Erode and Dilate is :

**Dilation**

* This operations consists of convoluting an image Awith some kernel (B), which can have any shape or size, usually a square or circle.
* The kernel B has a defined anchor point, usually being the center of the kernel.
* As the kernel Bis scanned over the image, we compute the maximal pixel value overlapped by Band replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to “grow” (therefore the name dilation). Take as an example the image above.

Applying dilation we can get:

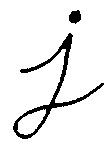


Figure 14 Dilation filter

The background (bright) dilates around the black regions of the letter.

**Erosion**

* This operation is the sister of dilation. What this does is to compute a local minimum over the area of the kernel.
* As the kernel Bis scanned over the image, we compute the minimal pixel value overlapped by Band replace the image pixel under the anchor point with that minimal value.
* Analogously to the example for dilation, we can apply the erosion operator to the original image (shown above). You can see in the result below that the bright areas of the image (the background, apparently), get thinner, whereas the dark zones (the “writing”( gets bigger.

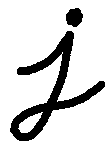


Figure 15 Erosion filter

***Code Snippet***:

CvInvoke.cvErode(skin, skin, Rect6, 1);

CvInvoke.cvDilate(skin, skin, Rect12, 1);

*4.5.3* ***Face Removal***

First, we detect the face using Haar Cascade, based on OpenCV Documentation:

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. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier.

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



Figure 16

Now all possible sizes and locations of each kernel is used to calculate plenty of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the integral images. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.

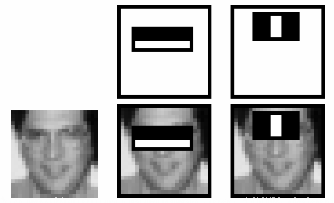


Figure 17

For this, 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. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can’t classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features.

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don’t process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don’t consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

Authors’ detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in first five stages. (Two features in the above image is actually obtained as the best two features from Adaboost). According to authors, on an average, 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works.

OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in opencv/data/haarcascades/ folder.

So we’ve used the pre-trained classifier of the face region. We wrapped the code into small function that take the original image and loops over faces based on the classifier, then draw a black Ellipse over the faces then return with the output which is the same image but with a black ellipse over the faces.



*4.5.4* ***Canny Edge Detector***

In this step, we detect the edges using Canny Edge Detector method to send the output after that to the Convex Hull function.

So basically The **Canny edge detector** is an [edge detection](http://en.wikipedia.org/wiki/Edge_detection) operator that uses a multi-stage [algorithm](http://en.wikipedia.org/wiki/Algorithm) to detect a wide range of edges in images. It was developed by [John F. Canny](http://en.wikipedia.org/wiki/John_F._Canny) in 1986. Canny also produced a *computational theory of edge detection* explaining why the technique works.

An edge in an image may point in a variety of directions, so the Canny algorithm uses four filters to detect horizontal, vertical and diagonal edges in the blurred image. The [edge detection operator](http://en.wikipedia.org/wiki/Edge_detection) ([Roberts](http://en.wikipedia.org/wiki/Roberts_Cross), [Prewitt](http://en.wikipedia.org/wiki/Prewitt), [Sobel](http://en.wikipedia.org/wiki/Sobel_operator) for example) returns a value for the first derivative in the horizontal direction (Gx) and the vertical direction (Gy). From this the edge gradient and direction can be determined:

\mathbf{G} = \sqrt{ {\mathbf{G}_x}^2 + {\mathbf{G}_y}^2 }

\mathbf{\Theta} = \operatorname{atan2}\left(\mathbf{G}_y, \mathbf{G}_x\right),

Where G can be computed using the [hypot](http://en.wikipedia.org/wiki/Hypot) function and [atan2](http://en.wikipedia.org/wiki/Atan2) is the arctangent function with two arguments. The edge direction angle is rounded to one of four angles representing vertical, horizontal and the two diagonals (0, 45, 90 and 135 degrees for example). Non-maximum suppression is an [edge thinning](http://en.wikipedia.org/wiki/Edge_detection#Edge_thinning) technique.

Given estimates of the image gradients, a search is carried out to determine if the gradient magnitude assumes a local maximum in the gradient direction. In some implementations, the algorithm categorizes the continuous gradient directions into a small set of discrete directions, and then moves a 3x3 filter over the output of the previous step (that is, the edge strength and gradient directions). At every pixel, it suppresses the edge strength of the center pixel (by setting its value to 0) if its magnitude is not greater than the magnitude of the two neighbors in the gradient direction. For example,

* if the rounded gradient angle is zero degrees (i.e. the edge is in the north–south direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the **east and west** directions,
* if the rounded gradient angle is 90 degrees (i.e. the edge is in the east–west direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the **north and south** directions,
* if the rounded gradient angle is 135 degrees (i.e. the edge is in the northeast–southwest direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the **north west and south east** directions,
* if the rounded gradient angle is 45 degrees (i.e. the edge is in the north west–south east direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the **north east and south west** directions.

In more accurate implementations, linear interpolation is used between the two neighboring pixels that straddle the gradient direction. For example, if the gradient angle is between 45 degrees and 90 degrees interpolation between gradients at the **north** and **north east** pixels will give one interpolated value, and interpolation between the **south** and **south west** pixels will give the other (using the conventions of last paragraph). The gradient magnitude at the central pixel must be greater than both of these for it to marked as an edge.

Note that the sign of the direction is irrelevant, i.e. north–south is the same as south–north and so on. Tracing edges through the image and hysteresis thresholding.

Large intensity gradients are more likely to correspond to edges than small intensity gradients. It is in most cases impossible to specify a threshold at which a given intensity gradient switches from corresponding to an edge into not doing so. Therefore Canny uses thresholding with [hysteresis](http://en.wikipedia.org/wiki/Hysteresis).

Thresholding with hysteresis requires two thresholds – high and low. Making the assumption that important edges should be along continuous curves in the image allows us to follow a faint section of a given line and to discard a few noisy pixels that do not constitute a line but have produced large gradients. Therefore we begin by applying a high threshold. This marks out the edges we can be fairly sure are genuine. Starting from these, using the directional information derived earlier, edges can be traced through the image. While tracing an edge, we apply the lower threshold, allowing us to trace faint sections of edges as long as we find a starting point.

Once this process is complete we have a binary image where each pixel is marked as either an edge pixel or a non-edge pixel. From complementary output from the edge tracing step, the binary edge map obtained in this way can also be treated as a set of edge curves, which after further processing can be represented as polygons in the image domain.

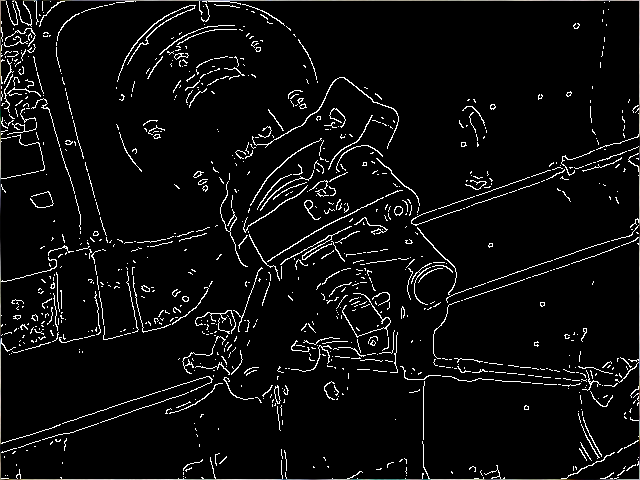
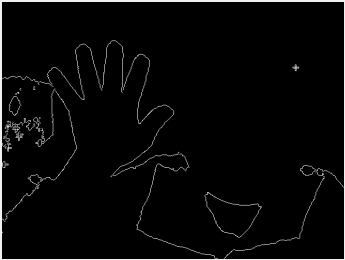


Figure 18 Canny edge detector applied to steam engine image

**Code Snippet:**

CvInvoke.cvCanny(Mask, CannyImage, 30, 60, 3);



Canny Edge detector applied to the captured stream

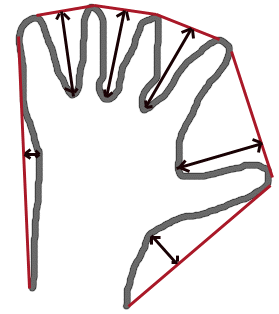
*4.5.5* ***Convex Hull***

In [mathematics](http://en.wikipedia.org/wiki/Mathematics), the **convex hull** or **convex envelope** of a set *X* of points in the [Euclidean plane](http://en.wikipedia.org/wiki/Euclidean_plane) or [Euclidean space](http://en.wikipedia.org/wiki/Euclidean_space) is the smallest [convex set](http://en.wikipedia.org/wiki/Convex_set) that contains *X*. For instance, when *X* is a bounded subset of the plane, the convex hull may be visualized as the shape formed by a rubber band stretched around *X*.

Formally, the convex hull may be defined as the intersection of all convex sets containing *X* or as the set of all [convex combinations](http://en.wikipedia.org/wiki/Convex_combination) of points in *X*. With the latter definition, convex hulls may be extended from Euclidean spaces to arbitrary [real vector spaces](http://en.wikipedia.org/wiki/Real_vector_space); they may also be generalized further, to [oriented matroids](http://en.wikipedia.org/wiki/Oriented_matroid).

The [algorithmic](http://en.wikipedia.org/wiki/Algorithm) problem of finding the convex hull of a finite set of points in the plane or other low-dimensional Euclidean spaces is one of the fundamental problems of [computational geometry](http://en.wikipedia.org/wiki/Computational_geometry).

In computational geometry, a number of algorithms are known for computing the convex hull for a finite set of points and for other geometric objects.

Computing the convex hull means constructing an unambiguous, efficient [representation](http://en.wikipedia.org/wiki/Data_structure) of the required convex shape. The complexity of the corresponding algorithms is usually estimated in terms of ***n***, the number of input points, and ***h***, the number of points on the convex hull.

For points in two and three dimensions, [output-sensitive algorithms](http://en.wikipedia.org/wiki/Output-sensitive_algorithm) are known that compute the convex hull in time O(*n* log *h*). For dimensions *d* higher than 3, the time for computing the convex hull is O(n^{\lfloor d/2\rfloor}), matching the worst-case output complexity of the problem.

Figure 20

*4.5.6* ***Image Moments***

In [image processing](http://en.wikipedia.org/wiki/Image_processing), [computer vision](http://en.wikipedia.org/wiki/Computer_vision) and related fields, an **image moment** is a certain particular weighted average ([moment](http://en.wikipedia.org/wiki/Moment_%28mathematics%29)) of the image pixels' intensities, or a function of such moments, usually chosen to have some attractive property or interpretation.

Image moments are useful to describe objects after segmentation. [Simple properties of the image](http://en.wikipedia.org/wiki/Image_moment#Examples) which are found *via* image moments include area (or total intensity), its [centroid](http://en.wikipedia.org/wiki/Centroid), and [information about its orientation](http://en.wikipedia.org/wiki/Image_moment#Examples_2).

For a 2D continuous function *f*(*x*,*y*) the [moment](http://en.wikipedia.org/wiki/Moment_%28mathematics%29) (sometimes called "raw moment") of order (*p* + *q*) is defined as

 M_{pq}=\int\limits_{-\infty}^{\infty} \int\limits_{-\infty}^{\infty} x^py^qf(x,y) \,dx\, dy

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

M_{ij} = \sum_x \sum_y x^i y^j I(x,y)\,\!

In some cases, this may be calculated by considering the image as a [probability density function](http://en.wikipedia.org/wiki/Probability_density_function), *i.e.*, by dividing the above by

\sum_x \sum_y I(x,y) \,\!

A uniqueness theorem (Hu [1962]) states that if *f*(*x*,*y*) is piecewise continuous and has nonzero values only in a finite part of the *xy* plane, moments of all orders exist, and the moment sequence (*Mpq*) is uniquely determined by *f*(*x*,*y*). Conversely, (*Mpq*) uniquely determines *f*(*x*,*y*). In practice, the image is summarized with functions of a few lower order moments.

### Examples

Simple image properties derived *via* raw moments include:

* Area (for binary images) or sum of grey level (for greytone images): *M*00
* Centroid: { *x*, *y* } = {*M*10/*M*00, *M*01/*M*00 }

**Code Snippet:**

MCvMoments moment = new MCvMoments();\

Hull = Big.GetConvexHull(Emgu.CV.CvEnum.ORIENTATION.CV\_CLOCKWISE);

Defects = Big.GetConvexityDefacts(storage, Emgu.CV.CvEnum.ORIENTATION.CV\_CLOCKWISE);

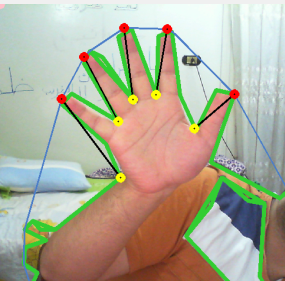


Figure 21 Convex Hull

*4.6* ***Output***

Latest phase of the system is to output the finger number to the display,

In this case we displayed it over the monitor, with the helpful of the finger number technique we have the ability to make some certain decisions based on finger number, for example if the finger count is 1, we may want to send a signal to any circuit attached to the computer and then switch the light on.

The tasks is numerous, in our system we attached a circuit to display the finger count in a 7-segment display.

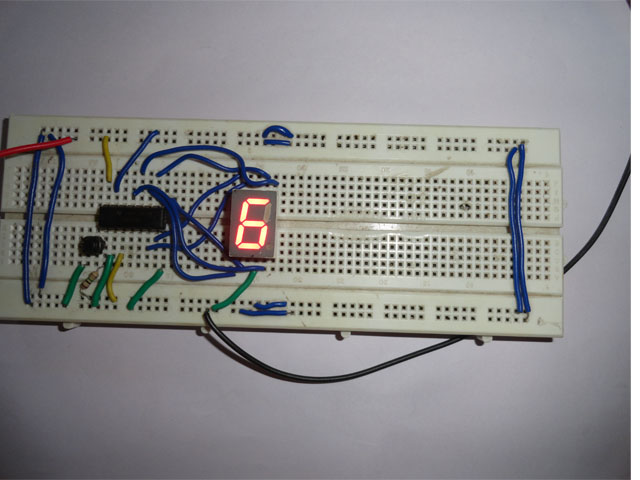


Figure 22 Displaying the output

*4.7* ***C# Code***

public partial class Main : Form

{

/\* For handeling the click event \*/

Capture capture; //Webcam Variable

Timer ImageTimer; //Timer to click pictures

Image<Bgr, Byte> MyImage;

Image<Gray, Byte> Mask;

Image<Bgr, Byte> MyImageRegion;

Image<Gray, Byte> MyMaskRegion;

private HaarCascade haar; //To detect the face using Haar Method

Seq<Point> Hull;

Seq<MCvConvexityDefect> Defects;

Seq<Point> filteredHull;

int fingerNum; //Global Number of fingers.

MCvConvexityDefect[] DefectArray;

public Main()

{

InitializeComponent();//Intializes the form UI.

capture = new Capture(0); //create a camera capture

capture.SetCaptureProperty(Emgu.CV.CvEnum.CAP\_PROP.CV\_CAP\_PROP\_FRAME\_WIDTH, 1280);

capture.SetCaptureProperty(Emgu.CV.CvEnum.CAP\_PROP.CV\_CAP\_PROP\_FRAME\_HEIGHT, 720);

haar = new HaarCascade("C:\\Emgu\\emgucv-windows-universal-cuda 2.9.0.1922\\opencv\\data\\haarcascades\\haarcascade\_frontalface\_default.xml");

ArrayList Images = new ArrayList();

}

public void ImageTimer\_Tick(object sender, EventArgs e)

{

//MyImage = I;

MyImage= capture.QueryFrame();

if (MyImage != null)

{

imageBox1.Image = MyImage;

//Remove the face out of the image

Image<Bgr, Byte> MyNoFaceImage = RemoveFace(MyImage);

imageBox2.Image = MyNoFaceImage;

//Take the skin out of the without face image

Mask = SkinDetect(MyNoFaceImage, new Ycc(0, 131, 80), new Ycc(255, 185, 135), 1);//Grayscale Image

imageBox4.Image = Mask;

//Extracting the canny edges

Image<Gray, Byte> CannyImage = new Image<Gray, Byte>(Mask.Size);

CvInvoke.cvCanny(Mask, CannyImage, 30, 60, 3);

imageBox5.Image = CannyImage;

//Getting the mask

MyMaskRegion = GetMask();

imageBox6.Image = MyMaskRegion;

imageBox7.Image = MyImageRegion;

if (MyImageRegion != null)

{

MyImageRegion.Resize(100, 100, INTER.CV\_INTER\_CUBIC);

}

ExtractFingerNumber(MyMaskRegion);

imageBox1.SizeMode = PictureBoxSizeMode.Zoom;

imageBox2.SizeMode = PictureBoxSizeMode.Zoom;

imageBox3.SizeMode = PictureBoxSizeMode.Zoom;

imageBox4.SizeMode = PictureBoxSizeMode.Zoom;

imageBox5.SizeMode = PictureBoxSizeMode.Zoom;

imageBox6.SizeMode = PictureBoxSizeMode.Zoom;

imageBox7.SizeMode = PictureBoxSizeMode.Zoom;

imageBox8.SizeMode = PictureBoxSizeMode.Zoom;

}

}

private Image<Bgr, Byte> RemoveFace(Image<Bgr, Byte> Input)

{

Image<Bgr, Byte> Output = Input.Copy();

Image<Gray, Byte> GrayImage = Input.Convert<Gray, Byte>();

var faces = GrayImage.DetectHaarCascade(

haar, 1.4, 4,

HAAR\_DETECTION\_TYPE.DO\_CANNY\_PRUNING,

new Size(Input.Width / 8, Input.Height / 8))[0];

foreach (var face in faces)

{

Rectangle R = face.rect;

Ellipse E = new Ellipse(new PointF(R.Location.X + R.Width / 2, R.Location.Y + R.Width / 2), new SizeF(R.Width, (int)(R.Height \* 1.2)), 90);

Output.Draw(E, new Bgr(Color.Black), -1);

}

return Output;

}

private Image<Gray, Byte> SkinDetect(Image<Bgr, Byte> Input, IColor min, IColor max, int a)

{

Image<Gray, byte> skin = new Image<Gray, byte>(Input.Width, Input.Height);

if (a == 1)

{

Image<Ycc, Byte> YCrCbInput = Input.Convert<Ycc, Byte>();

skin = YCrCbInput.InRange((Ycc)min, (Ycc)max);

}

else

{

Image<Hsv, Byte> HsvInput = Input.Convert<Hsv, Byte>();

skin = HsvInput.InRange((Hsv)min, (Hsv)max);

}

Image<Gray, Byte> skin2 = skin.Convert<Gray, Byte>();

imageBox3.Image = skin2;

StructuringElementEx Rect12 = new StructuringElementEx(12, 12, 6, 6, Emgu.CV.CvEnum.CV\_ELEMENT\_SHAPE.CV\_SHAPE\_CROSS);

StructuringElementEx Rect6 = new StructuringElementEx(6, 6, 3, 3, Emgu.CV.CvEnum.CV\_ELEMENT\_SHAPE.CV\_SHAPE\_CROSS);

CvInvoke.cvErode(skin, skin, Rect6, 1);

CvInvoke.cvDilate(skin, skin, Rect12, 1);

return skin;

}

private Image<Gray, Byte> GetMask()

{

using (MemStorage storage = new MemStorage()) //allocate storage for contour approximation

{

Contour<Point> contours = Mask.FindContours(Emgu.CV.CvEnum.CHAIN\_APPROX\_METHOD.CV\_CHAIN\_APPROX\_SIMPLE, Emgu.CV.CvEnum.RETR\_TYPE.CV\_RETR\_LIST, storage);

Contour<Point> Big = null;

Double BigArea = 0;

while (contours != null)

{

if (contours.Area > BigArea)

{

BigArea = contours.Area;

Big = contours;

}

contours = contours.HNext;

}

if (Big != null)

{

Contour<Point> currentContour = Big.ApproxPoly(Big.Perimeter \* 0.0035, storage);

MyImage.Draw(currentContour, new Bgr(Color.LimeGreen), 5);

var Ract = currentContour.BoundingRectangle;

int Diff = Ract.Height - Ract.Width;

if (Diff < 0)

{

Ract.Height += Math.Abs(Diff);

Ract.Y -= Math.Abs(Diff) / 2;

if (Ract.Y < 0) { Ract.Y = 0; }

if (Ract.Y + Ract.Height > MyImage.Height) { Ract.Y = MyImage.Height - Ract.Height; }

}

else

{

Ract.Width += Math.Abs(Diff);

Ract.X -= Math.Abs(Diff) / 2;

if (Ract.X < 0) { Ract.X = 0; }

if (Ract.X + Ract.Width > MyImage.Width) { Ract.X = MyImage.Width - Ract.Width; }

}

Ract.Inflate(new Size(40, 40));

MyImageRegion = MyImage.Copy();

Image<Gray, Byte> MYROI = Mask.Copy();

MyImageRegion.ROI = Ract;

MYROI.ROI = Ract;

return MYROI;

}

}

return null;

}

private void ExtractFingerNumber(Image<Gray, Byte> MyMask)

{

using (MemStorage storage = new MemStorage()) //allocate storage for contour approximation

{

Contour<Point> contours = null;

try

{

contours = MyMask.FindContours(Emgu.CV.CvEnum.CHAIN\_APPROX\_METHOD.CV\_CHAIN\_APPROX\_SIMPLE, Emgu.CV.CvEnum.RETR\_TYPE.CV\_RETR\_LIST, storage);

}

catch (Exception ex){

}

Contour<Point> Big = null;

Double BigArea = 0;

Contour<Point> Small = null;

Double SmallArea = 100000;

while (contours != null)

{

if (contours.Area > BigArea)

{

BigArea = contours.Area;

Big = contours;

}

if ((contours.Area < SmallArea)&&(contours.Area>40))

{

SmallArea = contours.Area;

Small = contours;

}

contours = contours.HNext;

}

CircleF This = new CircleF() ;

if (Big != null)

{

Contour<Point> currentContour = Big.ApproxPoly(Big.Perimeter \* 0.0025, storage);

MyImageRegion.Draw(currentContour, new Bgr(Color.LimeGreen), 5);

MCvMoments moment = new MCvMoments(); // a new MCvMoments object

moment = Big.GetMoments(); // Moments of biggestContour

CvInvoke.cvMoments(Big, ref moment, 0);

double m\_00 = CvInvoke.cvGetSpatialMoment(ref moment, 0, 0);

double m\_10 = CvInvoke.cvGetSpatialMoment(ref moment, 1, 0);

double m\_01 = CvInvoke.cvGetSpatialMoment(ref moment, 0, 1);

int current\_X = Convert.ToInt32(m\_10 / m\_00) / 10; // X location of centre of contour

int current\_Y = Convert.ToInt32(m\_01 / m\_00) / 10; // Y location of center of contour

Hull = Big.GetConvexHull(Emgu.CV.CvEnum.ORIENTATION.CV\_CLOCKWISE);

Defects = Big.GetConvexityDefacts(storage, Emgu.CV.CvEnum.ORIENTATION.CV\_CLOCKWISE);

DefectArray = Defects.ToArray();

MCvBox2D box = Big.GetMinAreaRect();

MyImageRegion.DrawPolyline(Hull.ToArray(), true, new Bgr(200, 125, 75), 2);

//MyImage.DrawPolyline(Defects.ToArray(), true, new Bgr(255, 255, 0), 2);

List<CircleF> SmallPoints = new List<CircleF>();

List<CircleF> BigPoints=new List<CircleF>();

List<LineSegment2D> FingerLines=new List<LineSegment2D>();

Double angled=0;

fingerNum = 0;

for (int i = 0; i < DefectArray.Length; i++)

{

PointF startPoint = new PointF((float)DefectArray[i].StartPoint.X, (float)DefectArray[i].StartPoint.Y);

PointF depthPoint = new PointF((float)DefectArray[i].DepthPoint.X, (float)DefectArray[i].DepthPoint.Y);

PointF endPoint = new PointF((float)DefectArray[i].EndPoint.X, (float)DefectArray[i].EndPoint.Y);

LineSegment2D startDepthLine = new LineSegment2D(DefectArray[i].StartPoint, DefectArray[i].DepthPoint);

LineSegment2D depthEndLine = new LineSegment2D(DefectArray[i].DepthPoint, DefectArray[i].EndPoint);

CircleF startCircle = new CircleF(startPoint, 5f);

CircleF depthCircle = new CircleF(depthPoint, 5f);

CircleF endCircle = new CircleF(endPoint, 5f);

if ((startCircle.Center.Y < box.center.Y || depthCircle.Center.Y < box.center.Y) && (startCircle.Center.Y < depthCircle.Center.Y) && (Math.Sqrt(Math.Pow(startCircle.Center.X - depthCircle.Center.X, 2) + Math.Pow(startCircle.Center.Y - depthCircle.Center.Y, 2)) > box.size.Height / 7))

{

fingerNum++;

MyImageRegion.Draw(startDepthLine, new Bgr(Color.Black), 2);

SmallPoints.Add(depthCircle);

BigPoints.Add(startCircle);

FingerLines.Add(startDepthLine);

}

}

foreach (CircleF a in SmallPoints)

{

MyImageRegion.Draw(a, new Bgr(Color.Yellow), 5);

}

foreach (CircleF a in BigPoints)

{

MyImageRegion.Draw(a, new Bgr(Color.Red), 5);

}

MyImageRegion.Draw(This, new Bgr(Color.Pink), 15);

label2.Text="";

if (fingerNum == 5)

{

label2.Text = "خمسة " ;

}

if (fingerNum == 0)

{

if (box.size.Height / box.size.Width > 1.5)

{

label2.Text = "صفر " ;

}

else

{

label2.Text = "صفر";

}

}

if (fingerNum == 3)

{

if (BigArea/SmallArea<20)

{

label2.Text = "ثلاثة";

}

label2.Text = "ثلاثة";

}

if (fingerNum == 2)

{

if((Math.Abs(FingerLines[0].GetExteriorAngleDegree(FingerLines[1]))<100)&&(Math.Abs(FingerLines[0].GetExteriorAngleDegree(FingerLines[1]))>70))

{

label2.Text = "Gun ";

}

if ((Math.Abs(FingerLines[0].GetExteriorAngleDegree(FingerLines[1])) < 50) && (Math.Abs(FingerLines[0].GetExteriorAngleDegree(FingerLines[1])) > 20))

{

label2.Text = "اثنان";

}

}

if (fingerNum ==1 )

{

label2.Text = "واحد";

label2.ForeColor = Color.Red;

}

if (fingerNum == 4)

{

label2.Text = "أربعة";

}

label1.Text = fingerNum.ToString();

}

}

}

private void toolStripButton1\_Click(object sender, EventArgs e)

{

ImageTimer = new Timer();

ImageTimer.Interval = 1000 / 24;

ImageTimer.Tick += new EventHandler(ImageTimer\_Tick);

ImageTimer.Start();

}

private void toolStripButton2\_Click(object sender, EventArgs e)

{

ImageTimer.Stop();

}

private void toolStripButton3\_Click(object sender, EventArgs e)

{

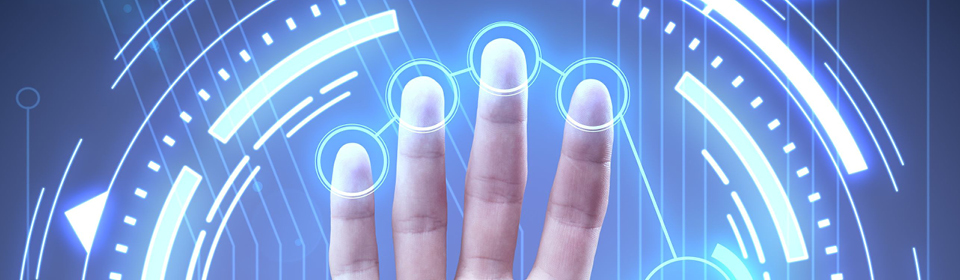
this.Close();

}

}

# Conclusion

***Conclusion***



We’ve developed an accurate hand gestures detection system based on hand Convex Hull. We started by detecting the hand skin color to track the hand and being able to extract the gestures easily. After that we removed the face for the sake of detecting the counters without any problems.

Then, we converted the image into binary, we faced a problem in the binary image; the edges has a sharp pixels, we filtered it using Erode and Dilate filters.

After that, we detected the edges using Canny Edge detector, then applied the Convex Hull to paint some points over the hand, then we summed it to get the fingers count. At the end, we succeeded in detecting the fingers accurately and being able to print the finger numbers to the monitor.

***Future work***

The body sense technology even gets rid of restriction of input device; make the HCI closer to human's nature action. The most important part of it will be manipulation using hand.

The studies in this field is growing continuously as its still new technology, and we believe that it will control the whole future interacting with machines.

# References

***[1] OpenCV Documentation.***

***[2] EmguCV documentation.***

***[3] Mastering OpenCV (Book).***

***[4] Digital Image processing" second Edition by Rafael Gonzalez & Richard Woods"***

***[5] Li, S.Z., Jain, A.K. (2004). Handbook of face recognition. Springer Science and Business Media, Inc. New York, USA.***

***[6] Real-Time Arabic Sign Language (ArSL) Recognition***

***[7] Real-Time Palm Tracking and Hand Gesture Estimation Based on Fore-Arm Contour***

***[8] Real-Time Tracking of Multiple Skin-Colored Objects with a Possibly Moving Camera” Antonis A. Argyros and Manolis I.A. Lourakis”***

***[9] Real –Time Hand Tracking and Gesture Recognition for Human-Computer Interaction “Cristina Manresa, Javier Varona, Ramon Mas and Francisco J. Per***