1. INTRODUCTION

1.1 Introduction to Gesture Recognition:

For a long time research on human-computer interaction (HCI) has been restricted to techniques based on the use of monitor, keyboard and mouse. Recently this paradigm has changed. Techniques such as vision, sound, speech recognition, projective displays and location aware devices allow for a much richer, multi-model interaction between man and machine. Gesture based interactive system, basically is an interaction system which is aimed at simulation of computer functions using certain gestures of fingers. The traditional equipment-based interaction methods require the commands to be issued to the computer using either the keyboard or mouse. But due to the new technologies and researches, there is an increased demand for interaction systems involving interaction with the display surface from a distance. A lot of work has been done on the development of hand tracking, finger tracking as interaction devices.

In the proposed system we are performing gesture recognition to perform Human Computer Interaction. We applied colored tapes on fingers and used gesture detection and color tracking system to perform various windows operations. Gesture recognition System is able to track the number of fingers using colored object tracking system. Instead of using a pre-defined color as color of the tape on fingers, there is a flexibility that we chose the required color. The HSV values of the color can be set dynamically according to the need. However, we have set a default values to Red color. Colored tapes are tied to finger tips. When a gesture is posed in front of the web-cam the gesture is identified as gesture number 1 or 2 or 3 etc. The number of fingers shown in the gesture are counted and thereby specific gesture is identified.

The conversion of coloured image to grey scale image is performed using HSV colour model HSV (Hue, Saturation and Value): It express Hue with dominant colour (such as red, green, purple and yellow) of an area. Saturation measures the colourfulness of an area in proportion to its brightness. The “intensity”, “lightness”, or “Values” is related to the colour luminance. This model discriminates luminance from chrominance. This is a more intuitive method for describing colours, and because the intensity is independent of the colour information this is very useful model for computer vision. This

model gives poor result where the brightness is very low. Erosion and dilation are performed on binary images.

To improvise the counting system we are performing erosion and dilation on detected objects. Dilation and erosion are two fundamental morphological operations. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the element used to process the image. It is typically applied to binary image, but there are versions that work on grey scale image. This system also provides a method to set the level of erosion and dilations. After converting the captured gesture into a grey scale image, erosion and dilation are performed. The pixels which are random forms into a bulk of single object. The number of objects are counted and thereby the gesture is identified. Based on the number of fingers shown in the gesture an action is performed. For example, if gesture 1 is identified then accordingly 'paint' program is opened.

The aim of this project is to implement a Hand Gesture Recognition System and to evaluate its strengths and weaknesses. Here, the model we chose for our system is described and then the theoretical background of the various functions used are outlined. Next our implementation will be described and finally the results obtained are discussed.

1.2 Need of the Project:

Gesture recognition enables humans to communicate with the machine and interact naturally without any mechanical devices. Using the concept of gesture recognition, it is possible to point a finger at the computer screen so that the cursor will move accordingly. This could potentially make conventional input devices such as mouse, keyboards and even touch-screens redundant.

This technology can be used in –

i. Robotics.

ii. Gaming.

iii. Systems which could understand human behavior based on their way of interaction.

iv. Enable disabled (handicapped) people to interact with computers with much ease.

The high-end applications of Gesture Recognition stand symbolic to the sophistication that the world is witnessing. This technology is the basic for applications such as

i. Switching Channels, Without a TV Remote

ii. Automated Homes.

iii. Controlling applications such as music players, photo gallery using gestures.

iv. Driving to Safety using Gestures.

v. Performing Mouse operations using Gestures.

vi. Medical applications where surgeons can operate on medical equipment without moving from their place using hand gestures.

1.3 Survey of the project:

Gesture recognition can be seen as a way for humans to interact with computers directly without any input devices. Gesture recognition can be performed using skin color detection methods or object detection methods. The skin color detection techniques are useful to track the palm position and pose by identifying the skin color and eliminating the background. The background from the frame is eliminated and only palm is cropped. Each palm position is eroded and dilated so that the appropriate gesture is identified. This model uses histogram generations and comparisons. Whereas palm detection using colored bands makes use of object tracking mechanisms. The number of fingers are identified and accordingly specific action is performed. The erosion and dilation operations are performed on threshold image and number of blobs are identified. According the requirement action is performed.

Gesture recognition is an ideal example of multidisciplinary research. There are different tools for gesture recognition, based on the approaches ranging from statistical modelling, computer vision and pattern recognition, image processing, connectionist systems, etc. Most of the problems have been addressed based on statistical modelling, such as PCA, HMMs ,Kalman filtering, more advanced particle filtering and condensation algorithms. FSM has been effectively employed in modelling human gestures. Computer vision and pattern recognition techniques involving feature extraction, object detection, clustering, and classification, have been successfully used for many gesture recognition systems. Image-processing techniques such as analysis and detection of shape, texture, colour, motion, optical flow, image enhancement, segmentation, and contour modelling, have also

been found to be effective. Connectionist approaches, involving multilayer perceptron (MLP), Time Delay Neural Network (TDNN), and radial basis function network (RBFN), have been utilized in gesture recognition as well.

While static gesture (pose) recognition can typically be accomplished by template matching, standard pattern recognition, and neural networks, the dynamic gesture recognition problem involves the use of techniques such as time-compressing templates, dynamic time warping, HMMs, and TDNN.

2. LITERATURE REVIEW

2.1 Computer Vision:

Computer vision is a field which provides a detailed study of the image .This also includes capturing and analysis of the particular image obtained from the real world. In general computer vision provides a data from the real world which produces symbolic information. Computer vision is about extracting information from the images.

2.2 Gesture Recognition:

Gesture Recognition is a topic in computer science and language technology with the goal of interpreting human gestures via mathematical algorithms. Gestures can originate from any bodily motion or state but commonly originate from the face or hand. Current focuses in the field include emotion recognition from the face and hand gesture recognition. Many approaches have been made using cameras and computer vision algorithms to interpret sign language. However, the identification and recognition of posture, gait, proxemics, and human behaviors is also the subject of gesture recognition techniques. Gesture recognition can be seen as a way for computers to begin to understand human body language, thus building a richer bridge between machines and humans than primitive text user interfaces or even GUIs (graphical user interfaces), which still limit the majority of input to keyboard and mouse.

2.2.1 Coloured Markers on finger tips:

These coloured markers are placed at the tip of the user fingers. This helps the webcam to identify the movement of hand and the gesture recognition. In our system a colour is first selected by setting its HSV value. Markers of that colour are tied to the finger tips. Whenever a movement is performed by fingers, the coloured object is identified. Thereby on closing and opening of fingers, the number of objects of the colour set prior are identified. The Gesture is recognised based on the number of coloured objects counted.

2.2.2 Other Techniques used for Gesture Recognition:

For any system the first step is to collect the data necessary to accomplish a specific task. For hand posture and gesture recognition system different technologies are used for acquiring input data.

Present technologies for recognizing gestures can be divided into vision based, instrumented (data) glove, and coloured marker approaches.

A. Vision Based approaches: In vision based methods the system requires only camera(s) to capture the image required for the natural interaction between human and computers and no extra devices are needed. Although these approaches are simple but a lot of gesture challenges are raised such as the complex background, lighting variation, and other skin colour objects with the hand object, besides system requirements such as velocity, recognition time, robustness, and computational efficiency.

B. Instrumented Glove approaches: Instrumented data glove approaches use sensor devices for capturing hand position, and motion. These approaches can easily provide exact coordinates of palm and finger's location and orientation, and hand configurations however these approaches require the user to be connected with the computer physically which obstacle the ease of interaction between users and computers, besides the price of these devices are quite expensive, it is inefficient for working in virtual reality.

C. Coloured Markers approaches: Marked gloves or coloured markers are gloves that worn by the human hand with some colours to direct the process of tracking the hand and locating the palm and fingers, which provide the ability to extract geometric features necessary to form hand shape. The colour glove shape might consist of small regions with different colours or as applied in where three different colours are used to represent the fingers and palms, where a wool glove was used. The amenity of this technology is its simplicity in use, and cost low price comparing with instrumented data glove. However this technology still limits the naturalness level for human computer interaction to interact with the computer. recognition

(a) Data-Glove based.

Figure: 2.2.2-(a)

(b) Vision based.

Figure 2.2.2-(b)

(c) Colored markers

Figure 2.2.2-(c)

2.2.3 **Existing Work:**

Alon *et al*. introduced unified framework for gesture recognition and spatiotemporal gesture segmentation applied to American Sign Language (ASL). To recognize manual gestures in video, it is required to do both spatial and temporal gesture segmentations. Spatial gesture segmentation is the problem of determining where the gesturing hand is located in each video frame. Temporal gesture segmentation is the problem of determining when gesture starts and ends.

Various grey-level segmentation techniques**,** such as use of single threshold value, adaptive thresholding, P-tile method, edge pixel method, iterative method and use of fuzzy set are available for object segmentation. Stergiopoulou and Papamarkos [9] proposed hand segmentation by colour segmentation using YCbCr color map. Approximation of the hand morphology was accomplished by SGONG (self-growing and self-organized neural gas) Network.

Algorithms which were able to identify finger, palm centre, hand centre, hand slope as well as finger features for recognition purpose were developed. Probability based classification method was used for gesture classification. Skin colour detection and complex background are major challenges in hand gesture recognition. Binary linked objects are groups of pixels that share the same label due to their connectivity in a binary image. Burande *et al.* [10] implemented Blobs analysis technique for skin colour detection under complex background. Kalman filtering, HMM and Graph matching algorithm were used for gesture recognition. Howe *et al.* [11] had introduced fusion of skin colour and motion segmentation. Face and hand of signer were successfully detected by using skin colour segmentation. False detection of skin region in the uncontrolled background also occurs due to light variation, so motion segmentation was used to find the difference between the moving foreground object and the stationery background.

3. PRESENT WORK

3.1 PROBLEM STATEMENT:

With the goal of interpreting human hand gestures via mathematical algorithms, many approaches have been made to recognize sign language. Our aim is to design a system that can efficiently recognize the gesture posed by hand and perform corresponding action. The system consists of modules: skin filtering, palm cropping, colored object detection, gesture identification and event handling. Skin filtering is performed to eliminate the background noises and highlight the portion of palm, representing the gesture. Hand tracking and segmentation algorithm (HTS) is used to efficiently handle the challenges of vision based system such as skin color detection, complex background removal and variable lighting condition. The objective of this work is to open a windows application using hand gestures as medium of communication.

3.1.1 Existing System:

Many novel algorithms were developed for gesture recognition using Skin Colored base Segmentation and Hidden Markov Models. But the implementation of these algorithms become difficult in variable lightening conditions. Also identifying the gesture dynamically becomes difficult because the skin color ranges vary from person to person. These algorithms were found to work for skin colour detection but they were sensitive for complex background

3.1.2 Proposed System:

The proposed work makes use of coloured markers attached on tips of fingers. The movement of hand can be easily recognised using this coloured object's movement. The colour used for markers can be chosen dynamically. The problem faced because of variable lightening conditions to detect skin colour is reduced to a great extent. Also the erosion and dilation functions whose parameters can be set dynamically help in making precise recognition. Using the coloured markers on finger tips the number of fingers in the input are identified. Accordingly gesture is identified and corresponding action is performed.

3.2 REQUIREMENTS SPECIFICATION:

3.2.1 Hardware Requirements

- HardDisk:100 GB(min)
- Processor: Pentium and higher
- RAM: 512MB.(min)
- Web Camera

3.2.2 Software Requirements

- Operating System : Windows XP and above Versions
- Programming Language : Visual CPP
- IDE : Microsoft Visual Studio 2010
- Library: OpenCV.

3.3 PRCOCESS MODEL:

3.3.1 Description:

The work setup consists of a web-camera which is set on top of computer or available along with the computer. Any colored tapes (red or green are used here) are tied to the tips of the five fingers. User can set the required color by setting its HSV values in the interface provided. For example to set detect red colored tapes we have to set its minimum and maximum HSV values as following 163-179, 126-217 and 68-127.

Also we can set the erosion and dilation values as required in the interface provided. For our convenience we are setting erosion and dilation parameters as 0 and 17 respectively. The image sequence is captured first. Since video is collection of frames, functions is applied to each frame. To count the number of blobs the image should be in threshold image format. The HSV image is converted to threshold image. According to the erosion and dilation parameters set by

the interface some pixels in the boundaries of the image are added or deleted. This makes the blobs clearer, eliminating the noise. The number of contours in the image are counted. This gives the count of number of fingers raised. If the number of fingers is 1 then corresponding Gesture 1 is identified and action is performed.

3.3.2 Algorithm:

Input: Captured Video from web-cam
Output: Performing action based on identification of gesture.

Algorithm GestureRecognize

Step1: Capture the video sequence from web-cam.
Step2: Convert the image frames into HSV format.
Step3: Identify the specified colored objects in the frame.
Step4: Obtain the thresholded image of HSV image.
Step5: Perform Erosion.
Step6: Perform Dilation.
Step7: Convert the frame into contours.
Step8: Count the number of Contours.
Step9: Identify Gesture (finger tips counting).
Step10: Perform corresponding action.
Step11: Stop

3.3.3 Block Diagram:

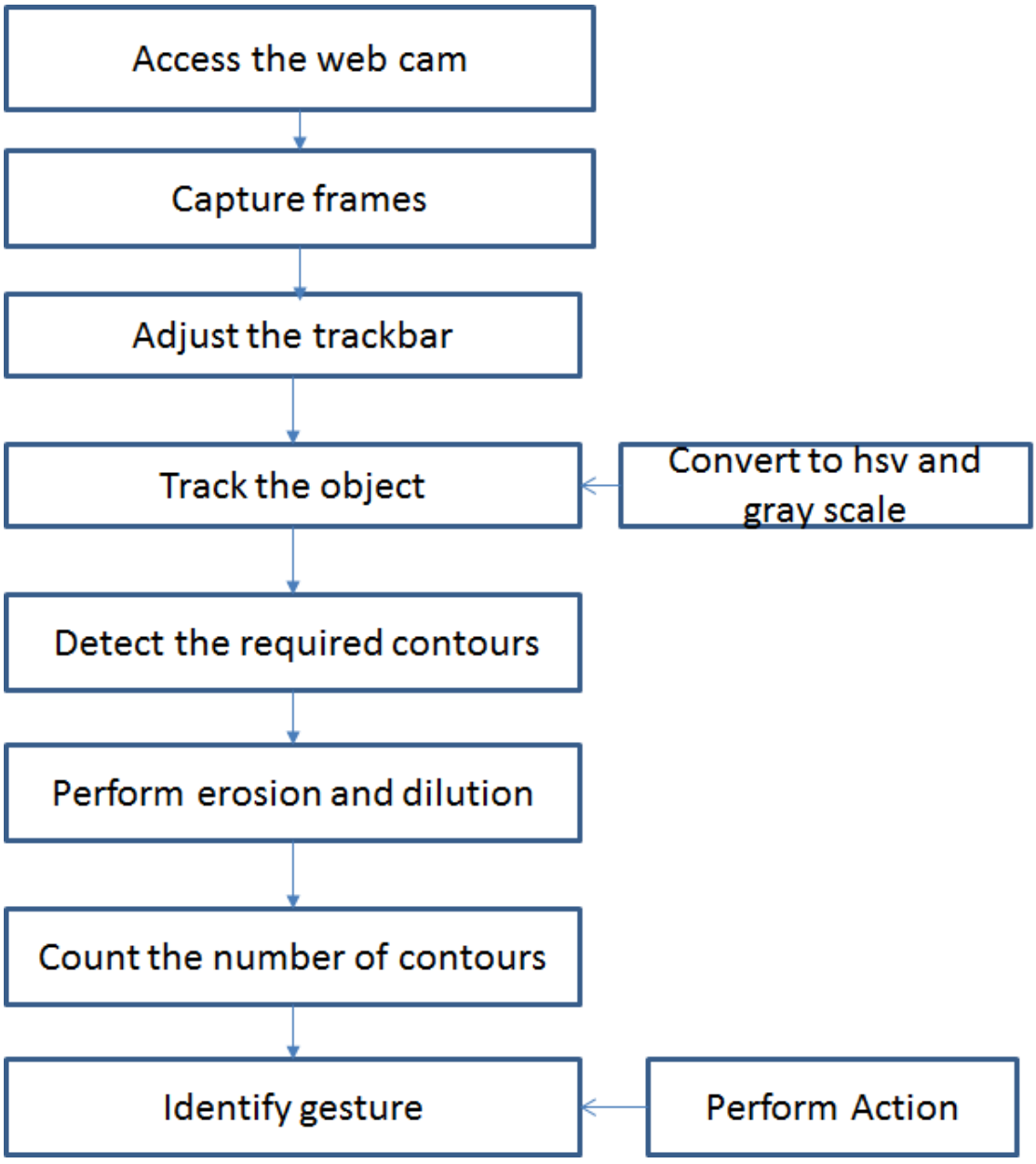


Figure 3.3.3.1

3.4 MODULES:

Module 1: Accessing Webcam

- Setting up the environment.
- Configuring opencv with Visual Studio.
- Accessing webcam from cpp program.

Module 2: Identifying the colored objects

- Storing the sequence of images.
- Setting up HSV values of color to be detected.
- Detecting the specific color.
- Converting HSV-Image to Thresholded form.

Module 3: Performing Erode and Dilution

- Set values of erode and dilution.
- Perform erode and dilution operation.

Module 4: Identifying the Gestures

- Dividing image into contours.
- Count the number of contours.
- Recognize the gestures.
- Perform related action.

3.5 MODULE WISE IMPLEMENTATION:

3.5.1 Accessing the web-cam:

Image can be stored in Mat form or Iplimage * format. Some functions to count contours are not applicable on iplmage* data structure. Hence, whenever necessary, Iplimage* can be converted to Mat format.

This is the output window of how a web-cam can be accessed to recognize a video stream.

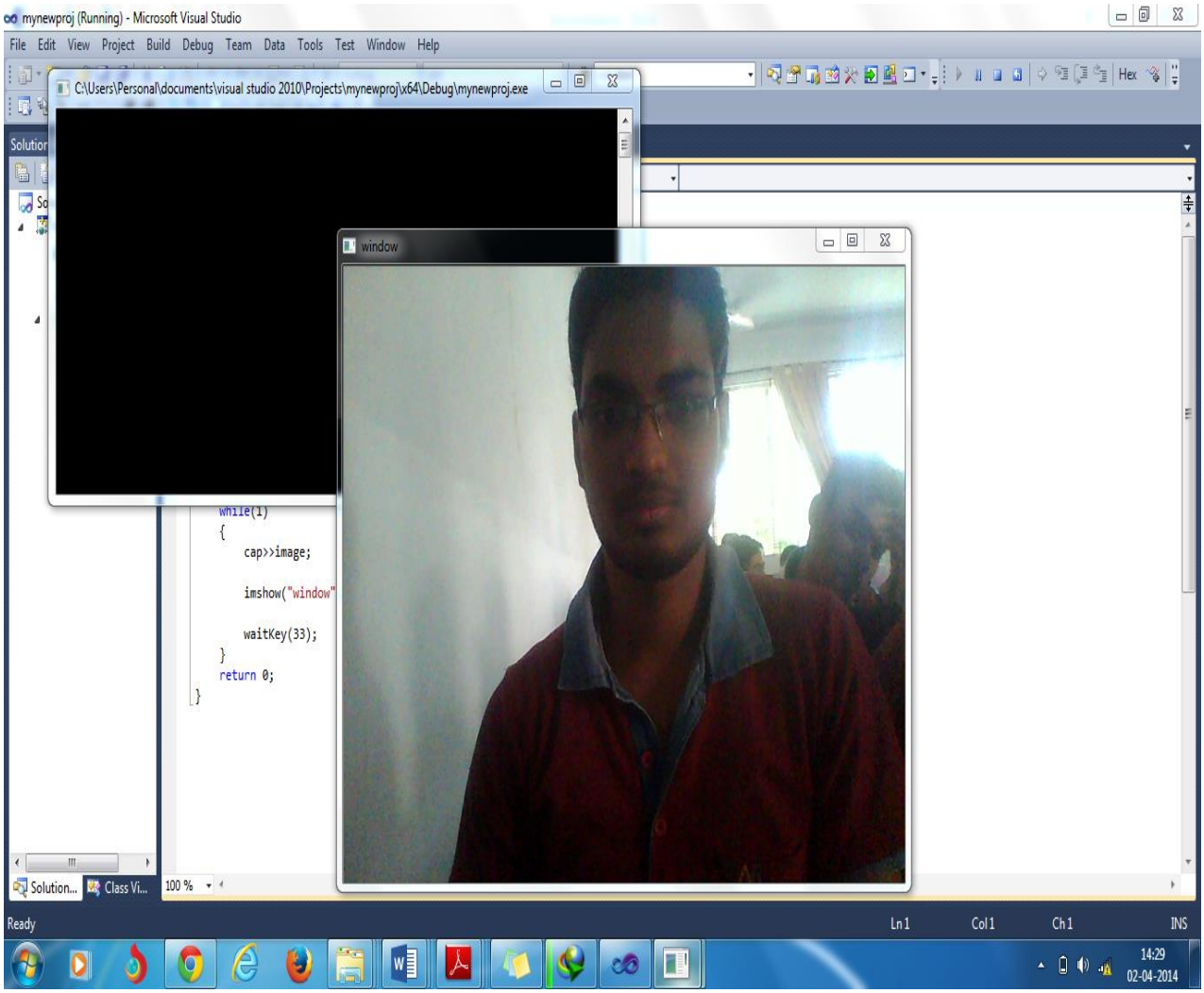


Figure 3.5.1.1

3.5.2 Converting an image from one format to another:

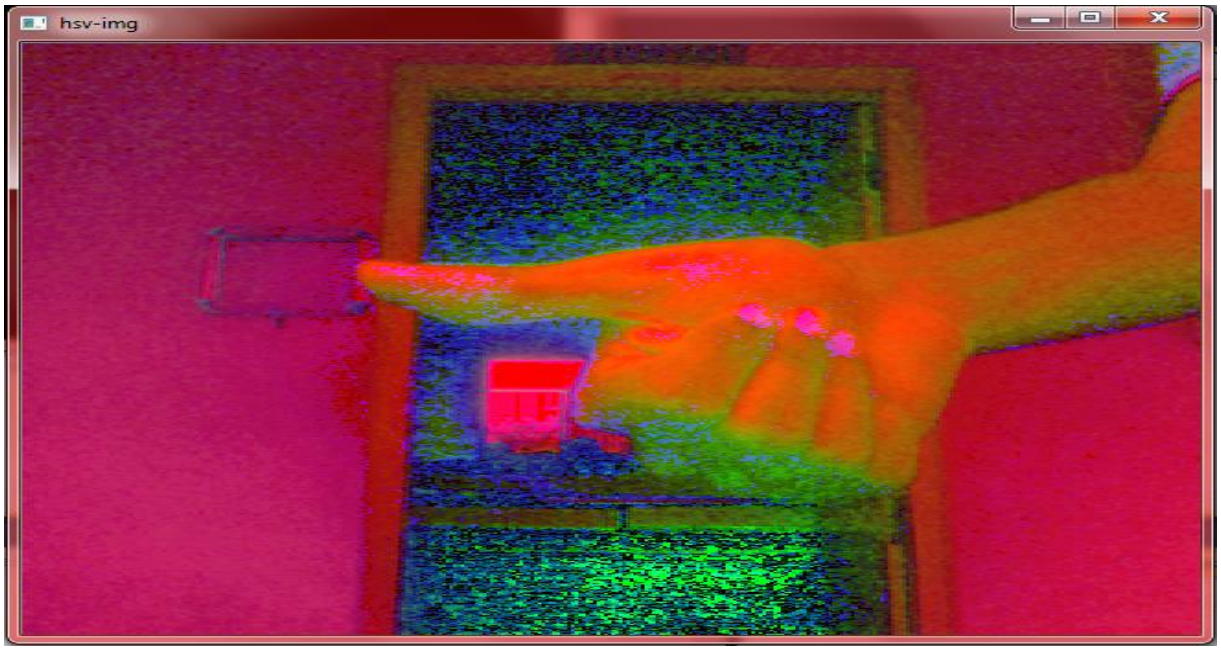


Figure 3.5.2.1

Here, Skin colored objects are identified. This is conversion of normal image to HSV image.

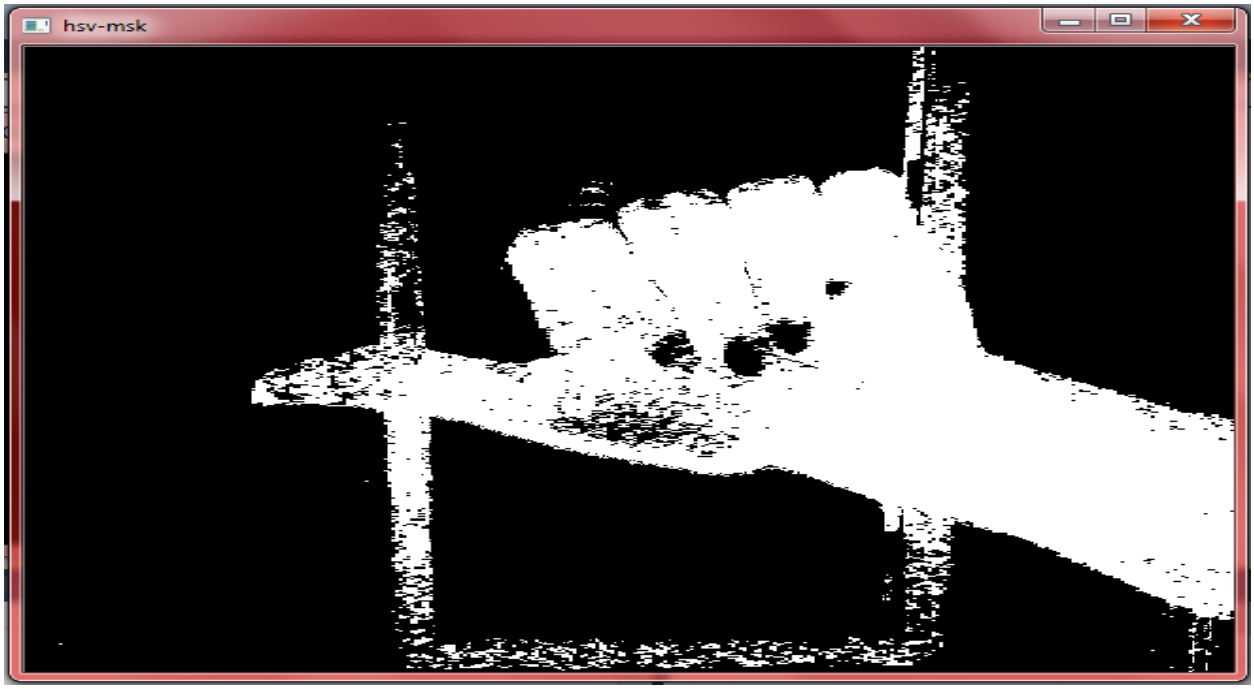


Figure 3.5.2.2

Required color is converted from HSV to Grey Colored

3.5.3 Perform Erosion and perform Dilation:

After performing erosion and dilation the image output is as follows.

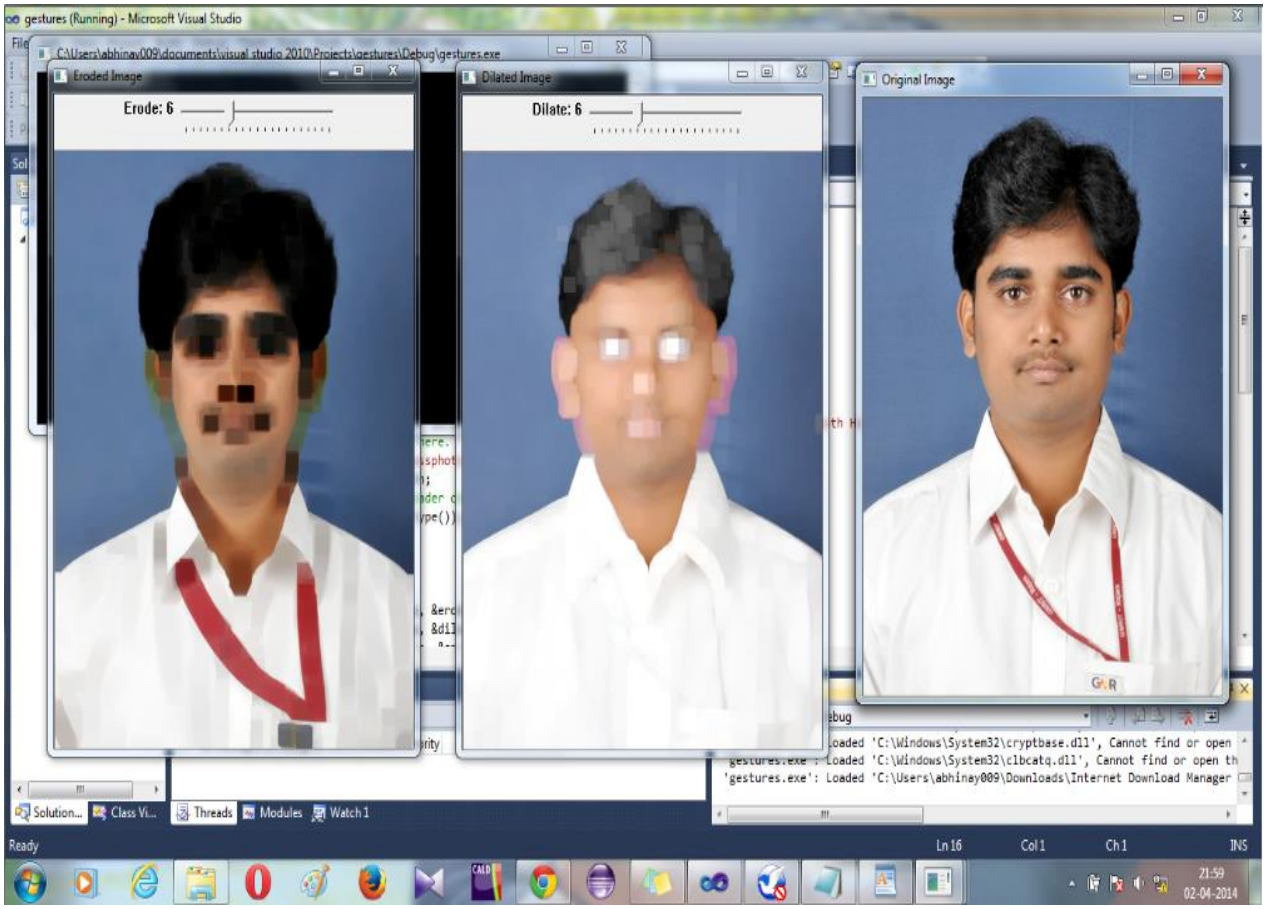


Figure 3.5.3.1

3.6 FEASIBILITY STUDY:

The feasibility of the project is analyzed in the phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential. Three key considerations involved in the feasibility analysis are:

3.6.1 Economical Feasibility: The purpose of assessing the Economical Feasibility is to identify the financial benefits and costs associated with the development of the project. Economical Feasibility is definitely feasible from economic point of view because the software and hardware requirements and the minimum number of operating personnel required for the operation of the project.

3.6.2 Technical Feasibility: This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. The purpose of assessing Technical Feasibility is to gain an understanding of the designer's ability to construct the proposed system. This analysis should include an assessment of hardware, software and operation environments to be used as well as system size and complexity.

3.6.3 Operational Feasibility: The purpose of Operational feasibility is to access the degree to which the proposed system solves the difficulties of the existing system. Our assessment of the operational feasibility includes an analysis of how the proposed system helped the developers to build an efficient application. The current system faces a problem of which framework is to be used for better performance of an application. This problem can be solved by the proposed system through measuring the performance of frameworks.

3.7 THEORITICAL DESCRIPTION:

A) EROSION AND DILATION:

- The most basic morphological operations are two: Erosion and Dilation. They have a wide array of uses, i.e. :
    - Removing noise
    - Isolation of individual elements and joining disparate elements in an image.
    - Finding of intensity bumps or holes in an image

- We will explain dilation and erosion briefly, using the following image as an example:

Figure 3.7 (i)

DILATION:

- This operations consists of convoluting an image A with 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 centre of the kernel.
- As the kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and 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 3.7 (ii)

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 B is scanned over the image, we compute the minimal pixel value overlapped by B and 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 3.7(iii)

cvErode:

Syntax: void cvErode(IplImage* src, IplImage* dst, IplConvKernel* B, int iterations);

Src: Source image.

Dst: Destination image.

B Structuring element used for erosion. If NULL, a 3x3 rectangular structuring element is used

Iterations: Number of times erosion is applied.

Discussion:

The function cvErode erodes the source image. The function takes the pointer to the structuring element, consisting of "zeros" and "minus ones"; the minus ones determine neighbors of each pixel from which the minimum is taken and put to the corresponding destination pixel. The function supports the in-place mode when the source and destination pointers are the same.

Erosion can be applied several times (iterations parameter). Erosion on a color image means independent transformation of all channels.

cvDilate:

Dilates image by using arbitrary structuring element.

Syntax: void cvDilate(IplImage* pSrc, IplImage* pDst, IplConvKernel* B, int Iterations);

pSrc Source image.

Dst: Destination image.

B Structuring element used for dilation. If NULL, a 3x3 rectangular structuring element is used.

Iterations: Number of times dilation is applied.

Discussion:

The function cvDilate performs dilation of the source image. It takes pointer to the structuring element that consists of "zeros" and "minus ones"; the minus ones determine neighbors of each pixel from which the maximum is taken and put to the corresponding destination pixel. Function supports in-place mode. Dilation can be applied several times (iterations parameter). Dilation of colour image means independent transformation of all channels.

The following image before and after performing erosion, dilation operations..

Figure 3.7 (iv)

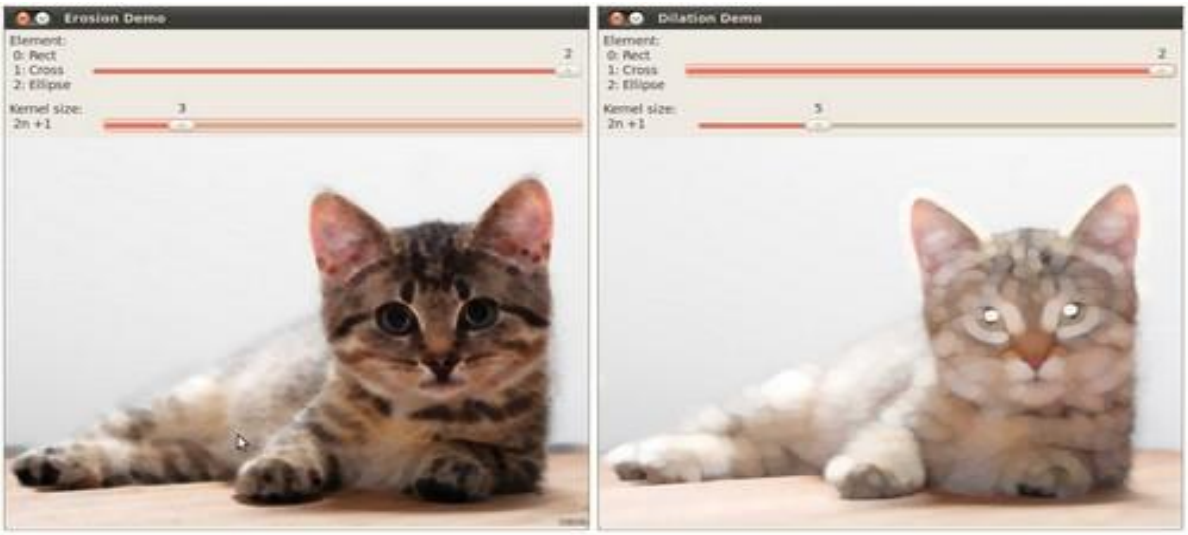


Figure 3.7 (v)

B) THRESHOLDING IMAGES:

Thresholding functions are used mainly for two purposes: masking out some pixels that do not belong to a certain range, for example, to extract blobs of certain brightness or colour from the image: converting grayscale image to bi-level or black-and-white image. Usually, the resultant image is used as a mask or as a source for extracting higher-level topological information, e.g., contours, skeletons, lines etc. Generally, threshold is a determined function *t(x,y)* on the image: the predicate function *f(x,y,p(x,y))* is typically represented as $g(x,y) < p(x,y) < h(x,y)$, where *g* and *h* are some functions of pixel value and in most cases they are simply constants. There are two basic types of thresholding operations. The first type uses a predicate function, independent from location, that is, *g(x,y)* and *h(x,y)*are constants over the image. However, for concrete image some optimal, in a sense, values for the constants can be calculated using image histograms or other statistical criteria.

The second type of the functions chooses *g(x,y)* and *h(x,y)*depending on the pixel neighbourhood in order to extract regions of varying brightness and contrast. The functions, described in this chapter, implement both these approaches. They support single-channel images with depth IPL_DEPTH_8U, IPL_DEPTH_8S or IPL_DEPTH_32F and can work in-place.

- **cvAdaptiveThreshold**

Provides adaptive thresholding binary image.

Sytax: void cvAdaptiveThreshold(IplImage* src, IplImage* dst, double max, CvAdaptiveThreshMethod method, CvThreshType type, double* parameters);

src : Source image.

dst : Destination image.

max : Max parameter, used with the types CV_THRESH_BINARY and CV_THRESH_BINARY_INV only.

Method : Method for the adaptive threshold definition; now CV_STDDEF_ADAPTIVE_THRESH only.

The function cvAdaptiveThreshold calculates the adaptive threshold for every input image pixel and segments image.

- **cvThreshold**

Thresholds binary image.

Syntax: void cvThreshold(IplImage* src, IplImage* dst, float thresh, float maxvalue, CvThreshType type);

Src : Source image.

Dst : Destination image; can be the same as the parameter src.

Thresh : Threshold parameter.

Maxvalue : Maximum value; parameter, used with threshold types CV_THRESH_BINARY, CV_THRESH_BINARY_INV, and CV_THRESH_TRUNC.

C) FINDING COUNTOURS:

Syntax: int cvFindContours(IplImage* img, CvMemStorage* storage, CvSeq** firstContour, int headerSize=sizeof(CvContour), CvContourRetrievalMode mode=CV_RETR_LIST,CvChainApproxMethod method=CV_CHAIN_APPROX_SIMPLE);

img :Single channel image of IPL_DEPTH_8U type. Non-zero pixels are treated as 1-pixels. The function modifies the content of the input parameter.

Storage: Contour storage location.

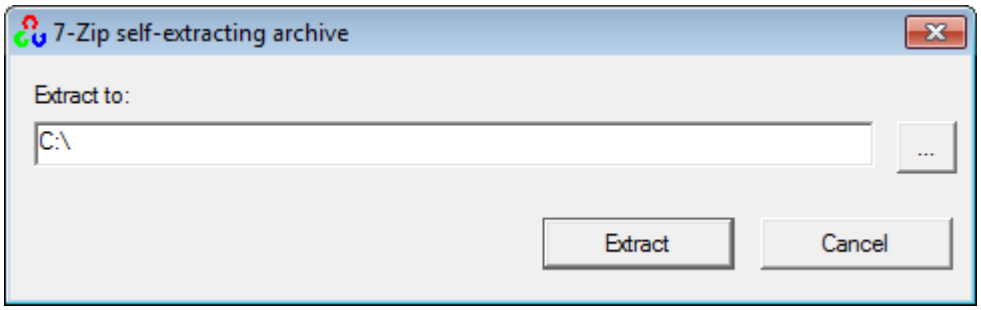
firstContour : Output parameter. Pointer to the first contour on the highest level.

headerSize : Size of the sequence header; must be equal to or greater than sizeof(CvChain) when the method CV_CHAIN_CODE is used, and equal to or greater than sizeof(CvContour) otherwise.
mode: Retrieval mode.
• CV_RETR_EXTERNAL retrieves only the extreme outer contours (list);
• CV_RETR_LIST retrieves all the contours (list);
• CV_RETR_CCOMP retrieves the two-level hierarchy (list of connected components);
• CV_RETR_TREE retrieves the complete hierarchy (tree) method
• CV_CHAIN_CODE outputs contours in the Freeman chain code.

D) Configuring openCV with Visual Studio

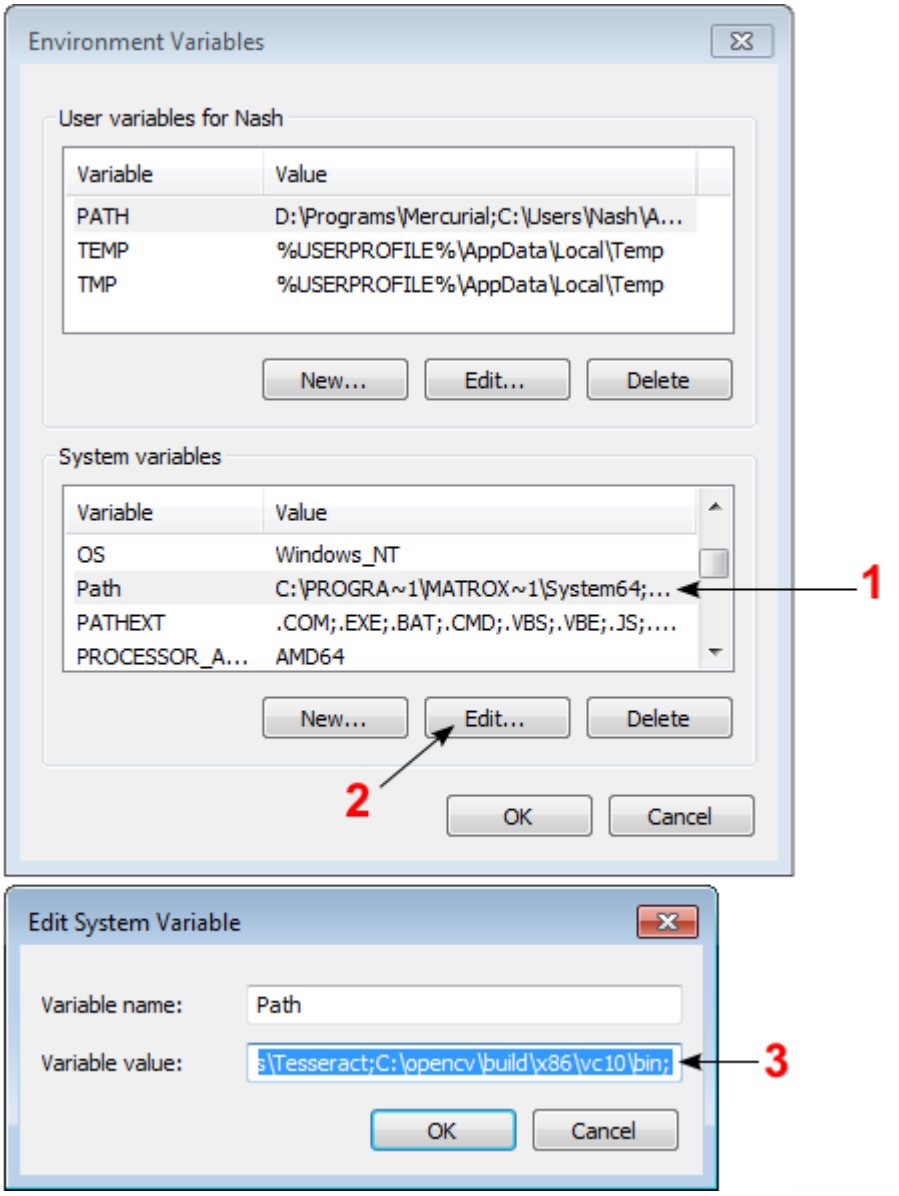
i) Installing OpenCV 2.4.3

Get OpenCV 2.4.8. It is a self-extracting so just double click to start the installation. Install it in a directory, say *C:*.



Wait until all files get extracted. It will create a new directory *C:\opencv* which contains OpenCV header files, libraries, code samples, etc.

Now we need to add the directory *C:\opencv\build\x86\vc10\bin* to system PATH. This directory contains OpenCV DLLs required for running the code.
Open **Control Panel → System → Advanced system settings → Advanced** Tab **→ Environment variables...**



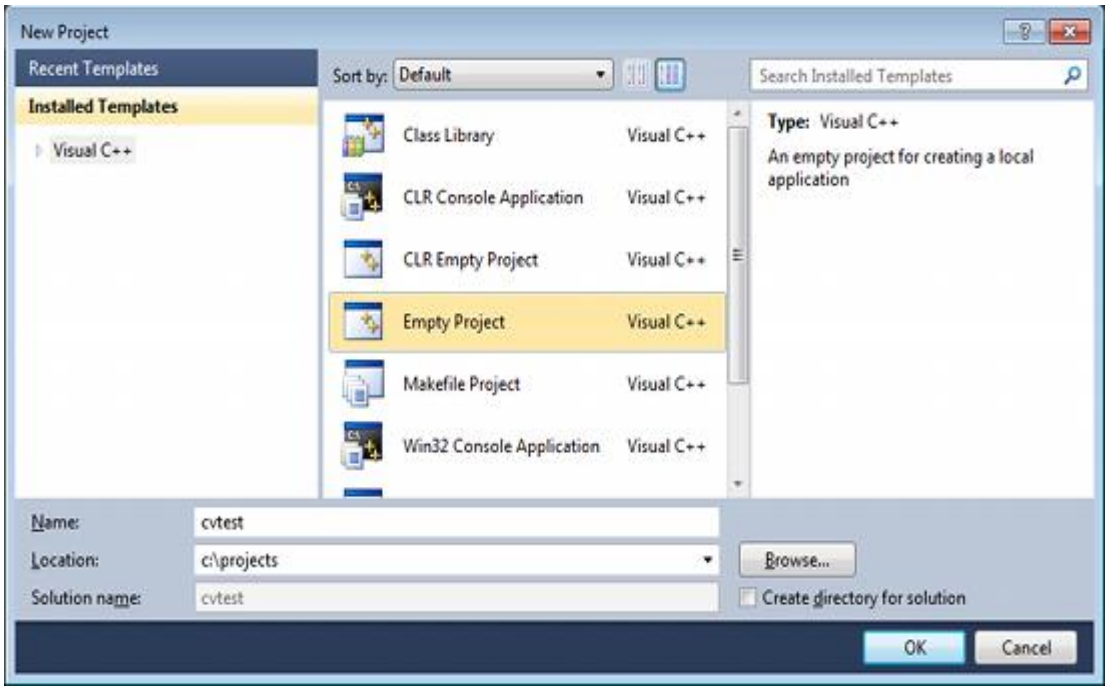
On the System Variables section, select **Path** (1), **Edit** (2), and type *C:\opencv\build\x86\vc10\bin;* (3), then click **Ok**.

On some computers, you may need to restart your computer for the system to recognize the environment path variables.

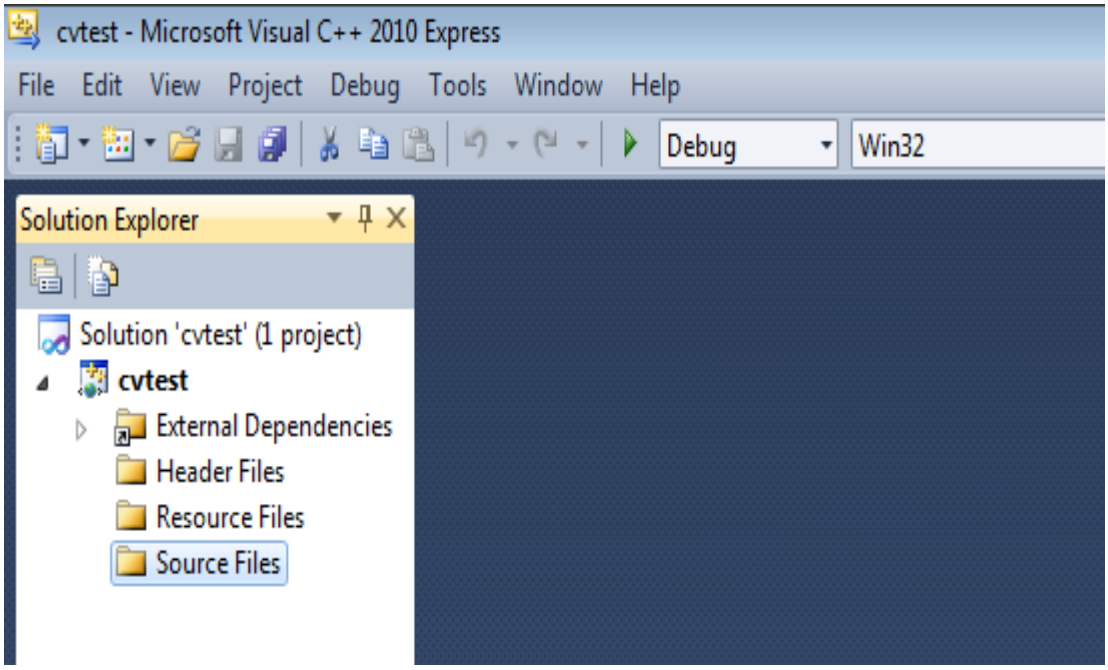
This will complete the OpenCV 2.4.8 installation.

ii) Create a new project and set up Visual C++

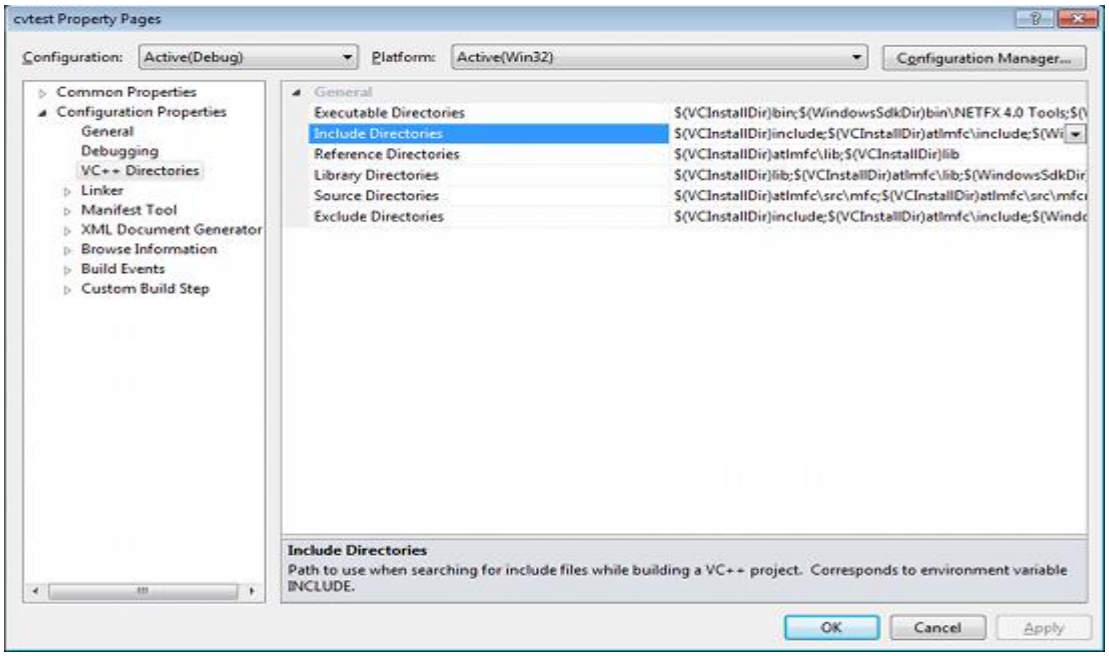
Open Visual C++ and select **File → New → Project... → Visual C++ → Empty Project**. Give a name for the project (e.g: cvtest) and set the project location.



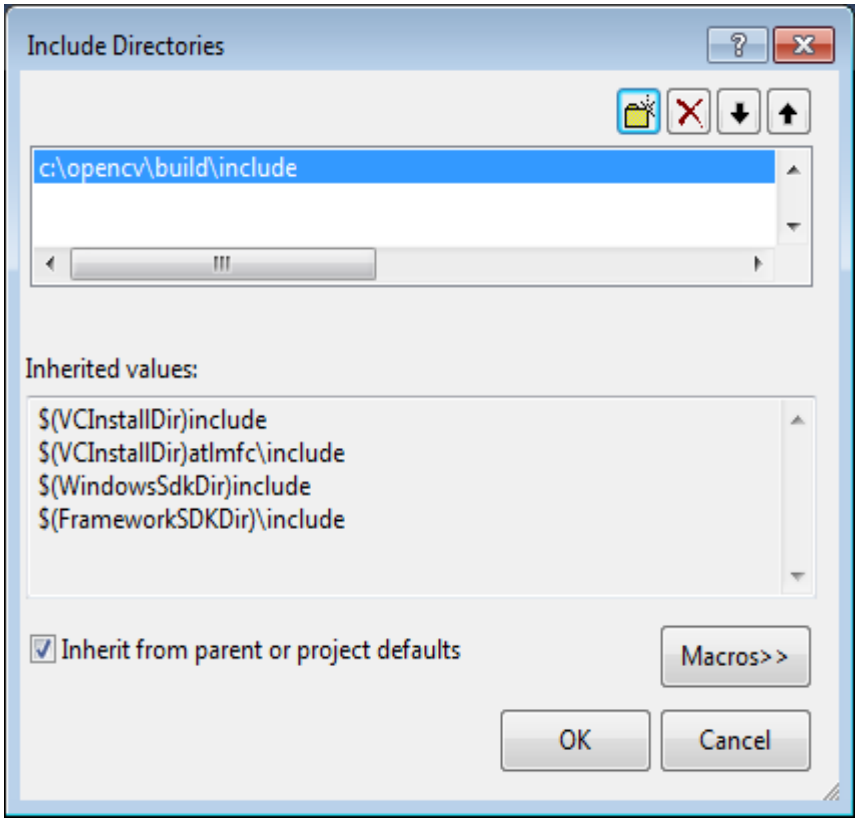
Click **Ok**. Visual C++ will create an empty project.



Make sure that "Debug" is selected in the solution configuration combo box. Right-click cvtest and select **Properties → VC++ Directories**.

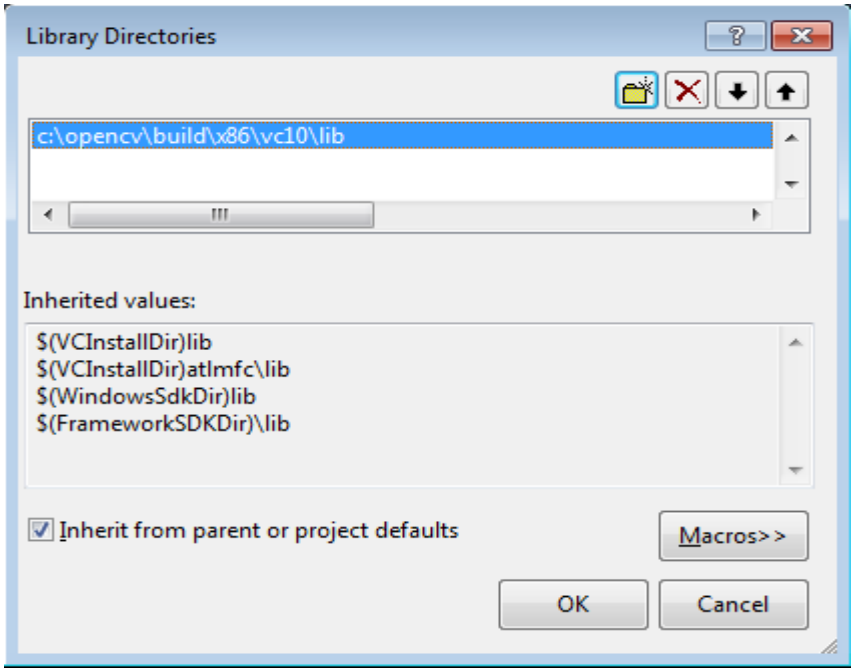


Select **Include Directories** to add a new entry and type *C:\opencv\build\include*.



Click **Ok** to close the dialog.

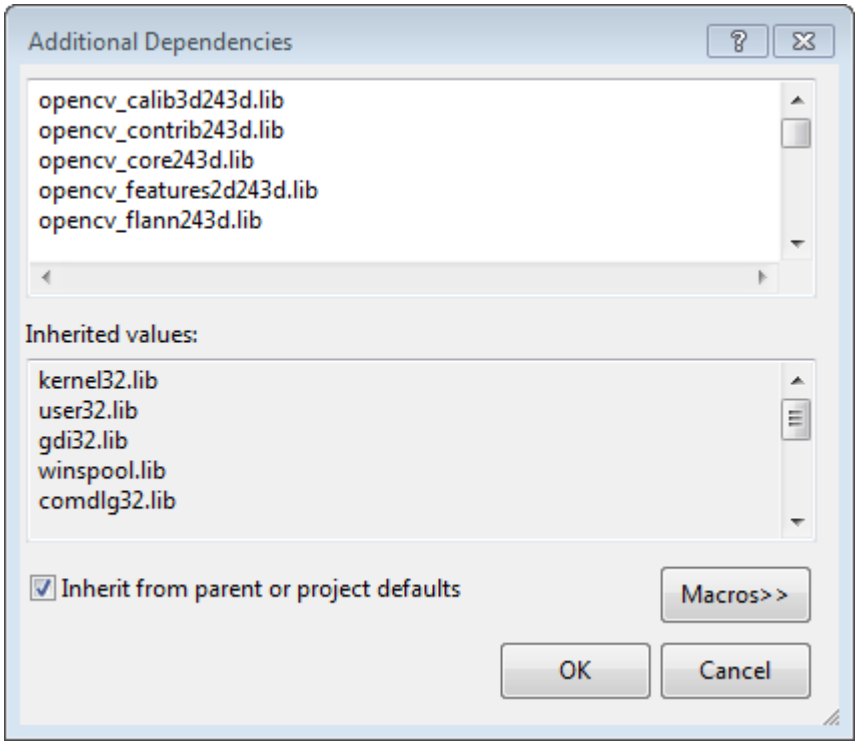
Back to the Property dialog, select **Library Directories** to add a new entry and type *C:\opencv\build\x86\vc10\lib*.



Click **Ok** to close the dialog.

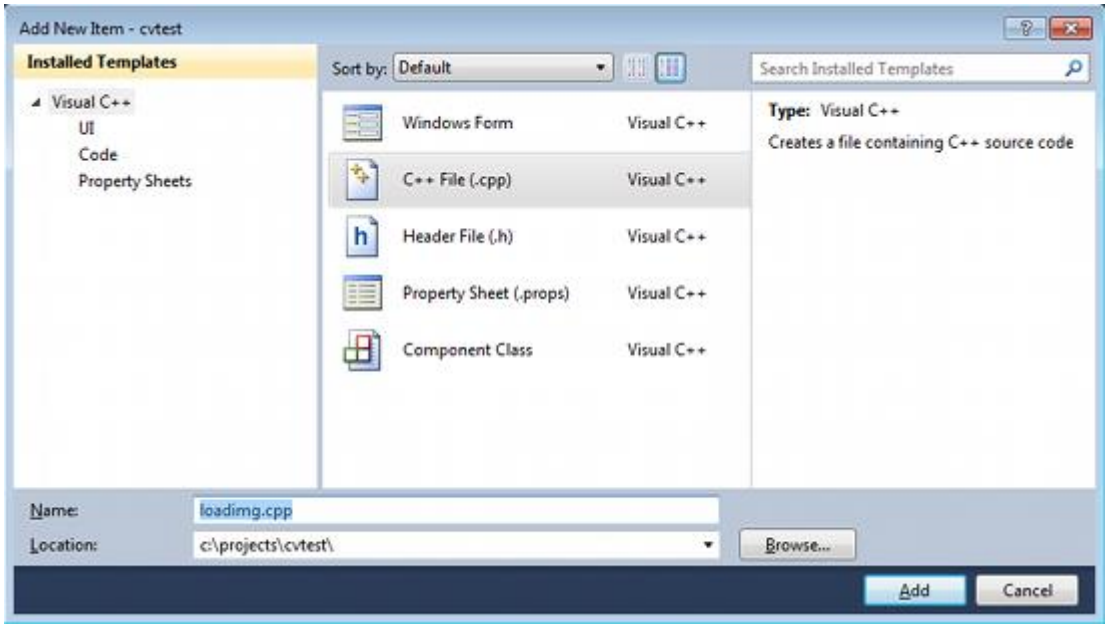
Back to the property dialog, select **Linker → Input → Additional Dependencies** to add new entries. On the popup dialog, type the files below:

opencv_calib3d248d.lib
opencv_contrib248d.lib
opencv_core248d.lib
opencv_features2d248d.lib
opencv_flann248d.lib
opencv_gpu248d.lib
opencv_highgui248d.lib
opencv_imgproc248d.lib
opencv_legacy248d.lib
opencv_ml248d.lib
opencv_nonfree248d.lib
opencv_objdetect248d.lib
opencv_photo248d.lib
opencv_stitching248d.lib
opencv_ts248d.lib
opencv_video248d.lib
opencv_videostab248d.lib



Click **Ok** to close the dialog. Click **Ok** on the project properties dialog to save all settings.

We are done setting up Visual C++. Right click your project and select **Add → New Item... → Visual C++ → C++ File**.



And write the code in the visual CPP file

3.8 TECHNICAL DESCRIPTION

3.7.1 Using OPENCV Library

Image functions:

- cvCreateImageHeader

Allocates, initializes, and returns structure
IplImage.

Syntax*:* IplImage* cvCreateImageHeader(CvSize *size*, int *depth*, int *channels*);

Parameters:

Size: Image width and height.
Depth: Image depth.
Channels: Number of channels.

- cvCreateImage

Creates header and allocates data.

Syntax: IplImage* cvCreateImage(CvSize size, int depth, int channels);
Size: Image width and height.
Depth: Image depth.
Channels: Number of channels.

- cvReleaseImageHeader

Releases header.

Syntax: void cvReleaseImageHeader(IplImage** image);
Image: Double pointer to the deallocated header.

- cvReleaseImage

Releases header and image data.

Syntax: void cvReleaseImage(IplImage** image)

Image: Double pointer to the header of the deallocated image.

- cvCreateImageData

Allocates image data.

Syntax: void cvCreateImageData(IplImage* image);

Image: Image header.

- cvReleaseImageData

Releases image data.

Syntax: void cvReleaseImageData(IplImage* image);

Image: Image header.

- cvSetImageData

Sets pointer to data and step parameters to given values.

Syntax: void cvSetImageData(IplImage* image, void* data, int step);

Image: Image header.

Data: User data.

Step: Distance between the raster lines in bytes.

- cvSetImageCOI

Sets channel of interest to given value.

Syntax: void cvSetImageCOI(IplImage* image, int coi);

image: Image header.

coi: Channel of interest.

- cvSetImageROI

Sets image ROI to given rectangle.

Syntax: void cvSetImageROI(IplImage* image, CvRect rect);

Image: Image header.

Rect: ROI rectangle.

- cvGetImageRawData

Fills output variables with image parameters.

Syntax: void cvGetImageRawData(const IplImage* image, uchar** data, int* step,CvSize* roiSize);

image: Image header.

data: Pointer to the top-left corner of ROI.

step: Full width of the raster line, equals to image->widthStep.

roiSize: ROI width and height.

- cvInitImageHeader

Initializes image header structure without memory allocation.

Syntax: void cvInitImageHeader(IplImage* image, CvSize size, int depth, int channels, int origin, int align, int clear);

image: Image header.

Size: Image width and height.

Depth: Image depth.

Channels: Number of channels.

Origin: IPL_ORIGIN_TL or IPL_ORIGIN_BL.

Align: Alignment for the raster lines.

Clear: If the parameter value equals 1, the header is cleared before initialization.

- cvCopyImage

Copies entire image to another without considering ROI.

Syntax: void cvCopyImage(IplImage* src, IplImage* dst);

src Source image.

dst Destination image.

Memory Storage:

- cvCreateMemStorage

Creates memory storage.

Syntax: CvMemStorage* cvCreateMemStorage(int blockSize=0);
blockSize : Size of the memory blocks in the storage; bytes.

- cvCreateChildMemStorage

Creates child memory storage.

Syntax: CvMemStorage* cvCreateChildMemStorage(CvMemStorage* parent);
parent : Parent memory storage.

- cvCreateMemStorage

Creates memory storage.

Syntax: CvMemStorage* cvCreateMemStorage(int blockSize=0);
blockSize : Size of the memory blocks in the storage; bytes.

- cvCreateMemStorage

Creates memory storage.

Syntax: CvMemStorage* cvCreateMemStorage(int blockSize=0);
blockSize Size of the memory blocks in the storage; bytes.

- cvCreateChildMemStorage

Creates child memory storage.

Syntax: CvMemStorage* cvCreateChildMemStorage(CvMemStorage* parent);
parent : Parent memory storage.

- cvReleaseMemStorage

Releases memory storage.

Syntax: void cvCreateChildMemStorage(CvMemStorage** storage);
storage : Pointer to the released storage.

- cvClearMemStorage

Clears memory storage

Syntax: void cvClearMemStorage(CvMemStorage* storage);
storage : Memory storage

- cvSaveMemStoragePos

Saves memory storage position.

Syntax:void cvSaveMemStoragePos(CvMemStorage* storage, CvMemStoragePos* pos);
storage : Memory storage.
pos : Currently retrieved position of the in-memory storage top.

- cvRestoreMemStoragePos

Restores memory storage position.

Syntax: void cvRestoreMemStoragePos(CvMemStorage* storage, CvMemStoragePos* pos);
storage : Memory storage.
pos : New storage top position.

3.8.2 Microsoft Visual Studio:

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows superfamily of operating systems, as well as web sites, web applications and web services. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows

Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer , class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source-control systems (like Subversion and Visual SourceSafe) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual Studio supports different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010). Support for other languages such as M, Python, and Ruby among others is available via language services installed separately. It also supports XML/XSLT, HTML/XHTML, JavaScript and CSS. Individual language-specific versions of Visual Studio also exist which provide more limited language services to the user: Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

3.9 SYSTEM DESIGN:

3.9.1 Introduction to Object Oriented Design

UML (UNIFIED MODELING LANGUAGE) Graph allows the declarative specification and drawing of UML class and sequence diagrams. The specification is done in text diagrams that are then transformed into the appropriate graphical representations. There is no rule specifying that models should appear in a graphical form.

A model is a simplification of reality, so a model for a software artifact could really be an outline of that artifact; think of a class definition without code in the method bodies. However,

we usually prefer to examine many of our models in a graphical representation: UML employs ten different diagrams for visualizing different perspectives of a system.Designers typically create their model diagrams using a drawing editor.

However, all drawing editors require the tedious placing and manipulation of drawing shapes on the canvas. The effort and the motor coordination skills required for this activity are mostly irrelevant to the end result: unlike architectural or mechanical engineering models the appearance of a software system's model diagram is only marginally related to the quality of the represented software design. Computer power and automatic graph drawing algorithms have now sufficiently advanced so as to allow the automatic placement of graph nodes on the canvas and the near optimal routing of the respective edges. We can therefore design models using a declarative textual representation and subsequently view, publish, and share them in graphical form.

UML Graph's support for declaratively specifying class and sequence diagrams is part of an ongoing effort aiming to support all ten types of UML diagrams. Creating models in a declarative, textual notation offers a number of advantages.

- First of all, the model composition mechanism matches well both a programmer's high level skills, the textual abstract formalization of concrete concepts, and the associated low-level skills, the manipulation of text using an editor and other text-based tools.
- The declarative notation, by being closer to the program's representation, forces the designer to distinguish between the model and the respective implementation, between the essential system characteristics and the trivial adornments. It is more difficult for designers to get away, as they often do now, with drawing for a model a nice picture of the implementation they have in mind.
- The declarative representation is also highly malleable; the existing visual structure does not hinder drastic changes, nor is effort wasted on the tidy arrangement of graph nodes a psychological barrier against massive design refactoring.
- Declarative models are also highly automatable: they can be easily generated from even higher-level descriptions by trivial scripts and tools operating on design process inputs such as database schemas, existing code, or structured requirements documents.
- Text macro processors can be used for configuration management, while revision control and team integration activities can utilize the same proven tools and processes that are currently used for managing source code. Thus with a tool like CVS or RCS one can keep

track of design revisions, create and merge branches, and monitor model changes, while a system like CVS can allow work to be split into teams.

- Finally, the declarative approach can readily utilize existing text processing tools for tasks that a drawing editor system may not provide.
- Consider how your favorite model editor handles the following tasks and how you could handle them using a simple Perl script or a text-processing pipeline applied to the declarative model specification:
- Identify all classes containing a given field (as prelude to an aspect-oriented cross-cut).
- Count the total number of private fields in a given design
- Order methods appearing in multiple classes by their degree of commonality
- Identify differences between two designs.

3.9.2 Use Case Analysis

Use case:

- A use case contains all the events that can occur between an actor and a set of scenarios that explains the interactions as a sequence of happenings.
- Use cases are low technology in nature, highly descriptive in form and written in human understandable languages. However it should be meaningful and describe the process of the system in consideration.

Actor

Actors are not part of the system. They represent anyone or anything that interacts with the system.

- An actor may only input information from the system.
- An actor may only receive information from the system.
- Input and receive information to and from the system.

Use-case diagrams are used during analysis to capture the system requirements and to understand how the system should work. During the design phase, we can use use-case diagrams to specify the behavior of the system as implemented. These diagrams present a high-level view of how the system is used as viewed from an outsider's (actor's) perspective.

Use case diagram:

Explanation:

Here user and GestureRecognitionSystem (GRS) are two actors. Where user can have the functions of setting the track bar values and performing gesture in front of the web-camera. The GRS system tracks the input received through web-cam and performs operations like Identify colored object, erode&dilate and identify the gesture based on algorithm. Accordingly output action is performed.

Figure 3.9.2(i)

3.9.3 Interaction Diagrams

An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them.

Sequence Diagram

A sequence diagram is an interaction diagram that emphasizes the time ordering of messages. Graphically, a sequence diagram is a table that shows objects along the X-axis and messages, ordered in increasing time, along Y-axis. Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction.

Explanation

Sequence diagram shows the flow of steps from user to GRS. The flow of input gesture from user to GRS via a webcam is shown. The sequence of actions performed by GRS are shown in order.

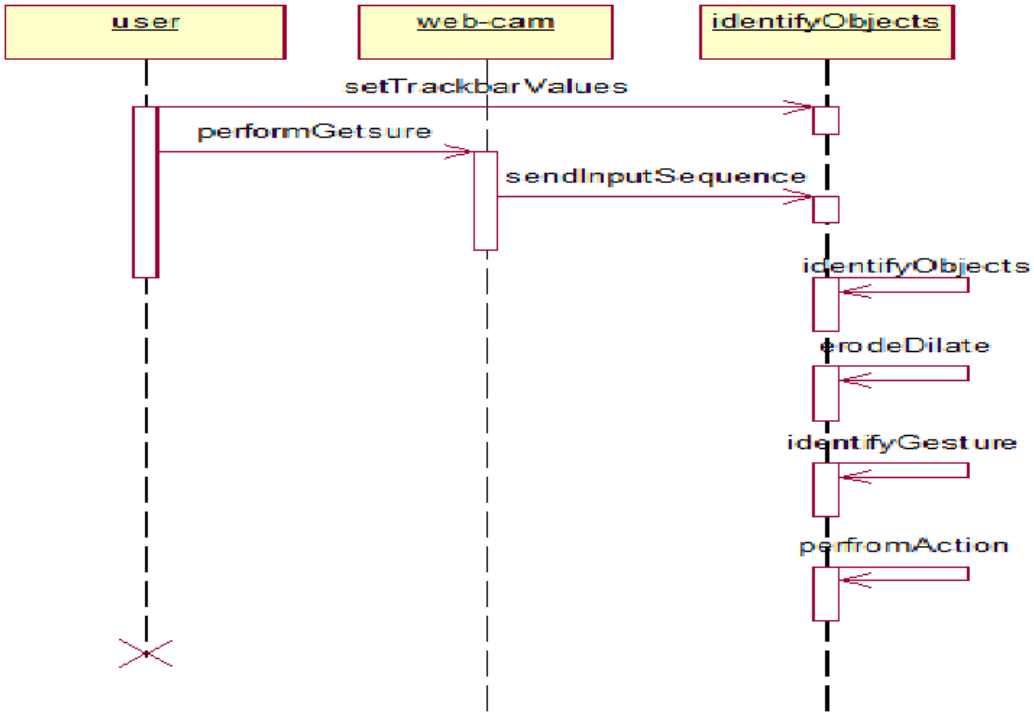


Figure:3.9.3(ii)

3.8.5 Class Diagrams

Class diagrams are the most common diagrams in modeling object oriented systems. A class diagram shows set of classes, interfaces, collaborations and relationships. Class diagrams are used to model static view of the system. They are used for specialization, visualizing and documenting structural models for constructing executable systems to forward and reverse engineering.

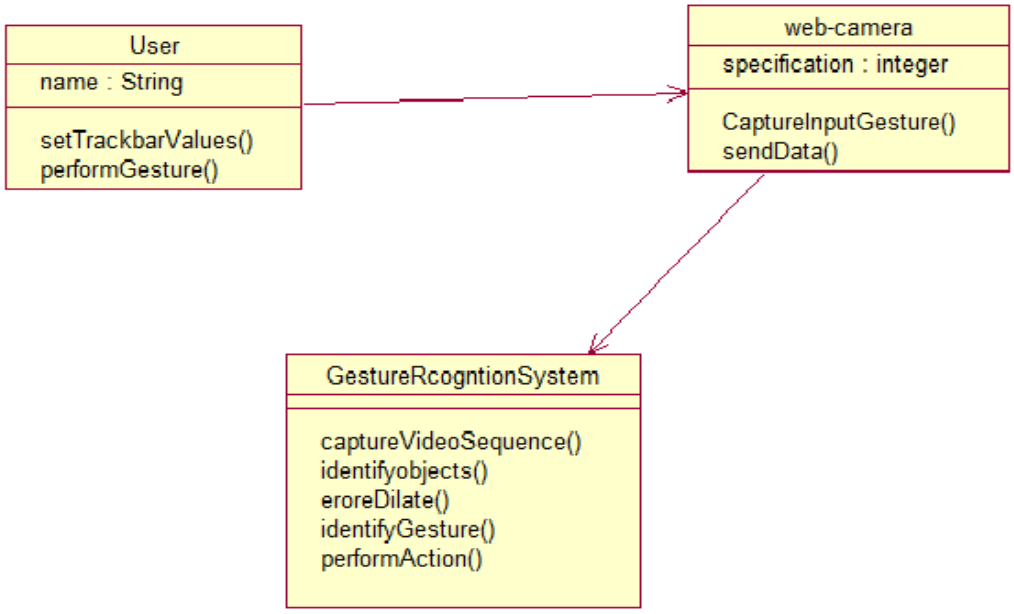


Figure 3.9.2(iii)

4. RESULTS AND DISCUSSIONS

4.1 OVERVIEW:

Gesture recognition can be seen as a way for computers to begin to understand human body language. With the goal of interpreting human hand gestures via mathematical algorithms, many approaches have been made to recognize sign language. Our aim is to design a system that can efficiently recognize the gesture posed by hand and perform corresponding action.

4.2 PROS:

- Dynamic gesture recognition with facility to choose desired color.
- Erosion and dilation operations are performed for precise outputs.
- Colored markers give precise outputs in variable lightening conditions.

4.3 USER FRIEENDLINESS:

When we see the proposed system design, we can clearly say that it is very much user friendly and the user who is using it may not necessary to know the knowledge over the backend tools for its usage.

4.4 OUTPUT SCREENS:

Every red colored object is counted as a single contour. The eroded and dilated image of a colored tape after detection is shown in the figure

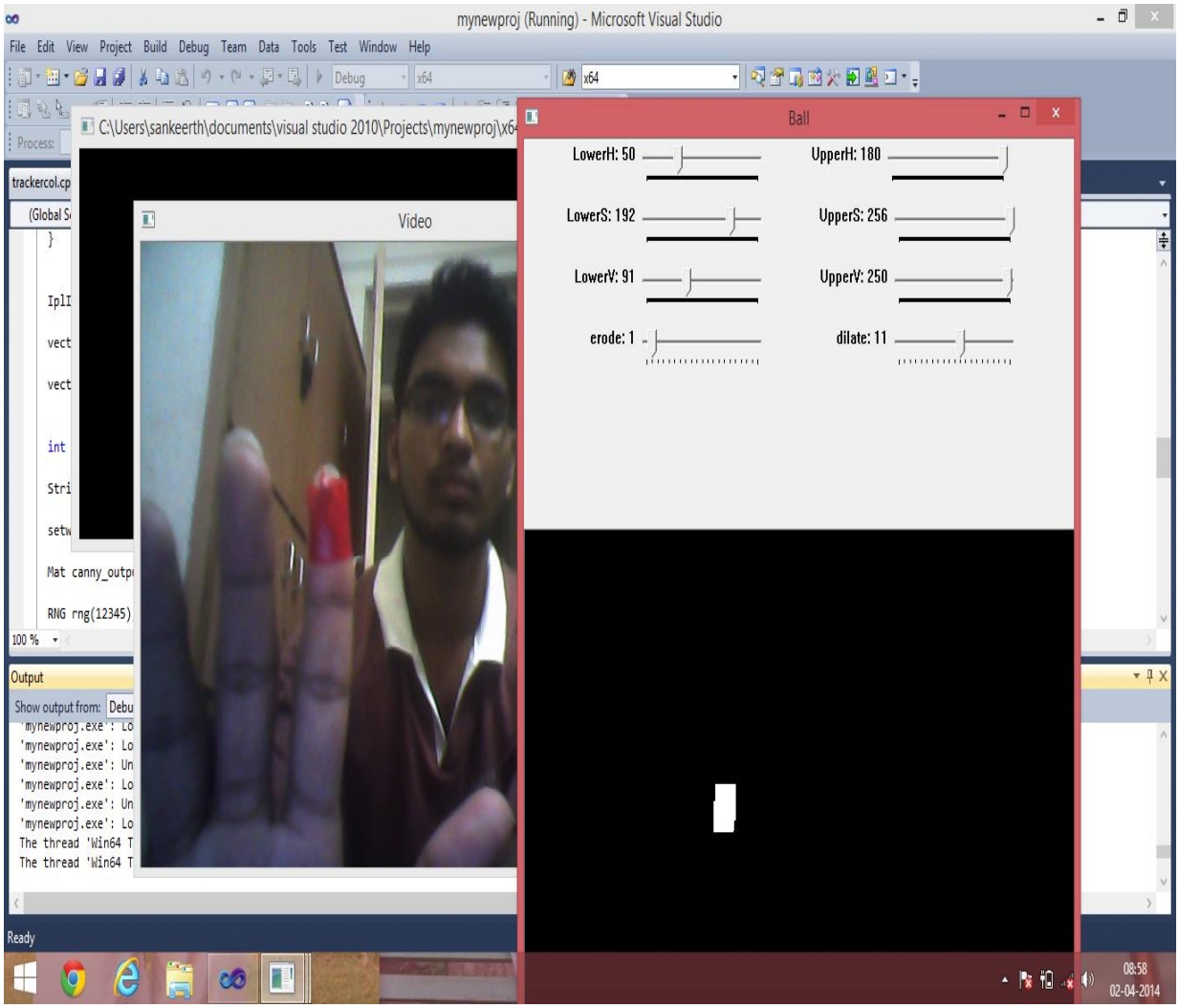


Figure 4.4(i)

GESTURE-1: When user shows one finger with colored strip, the gesture is identified as gesture number 1 and accordingly notepad program is opened. User can set the track bars according to the HSV values of the color he chooses.

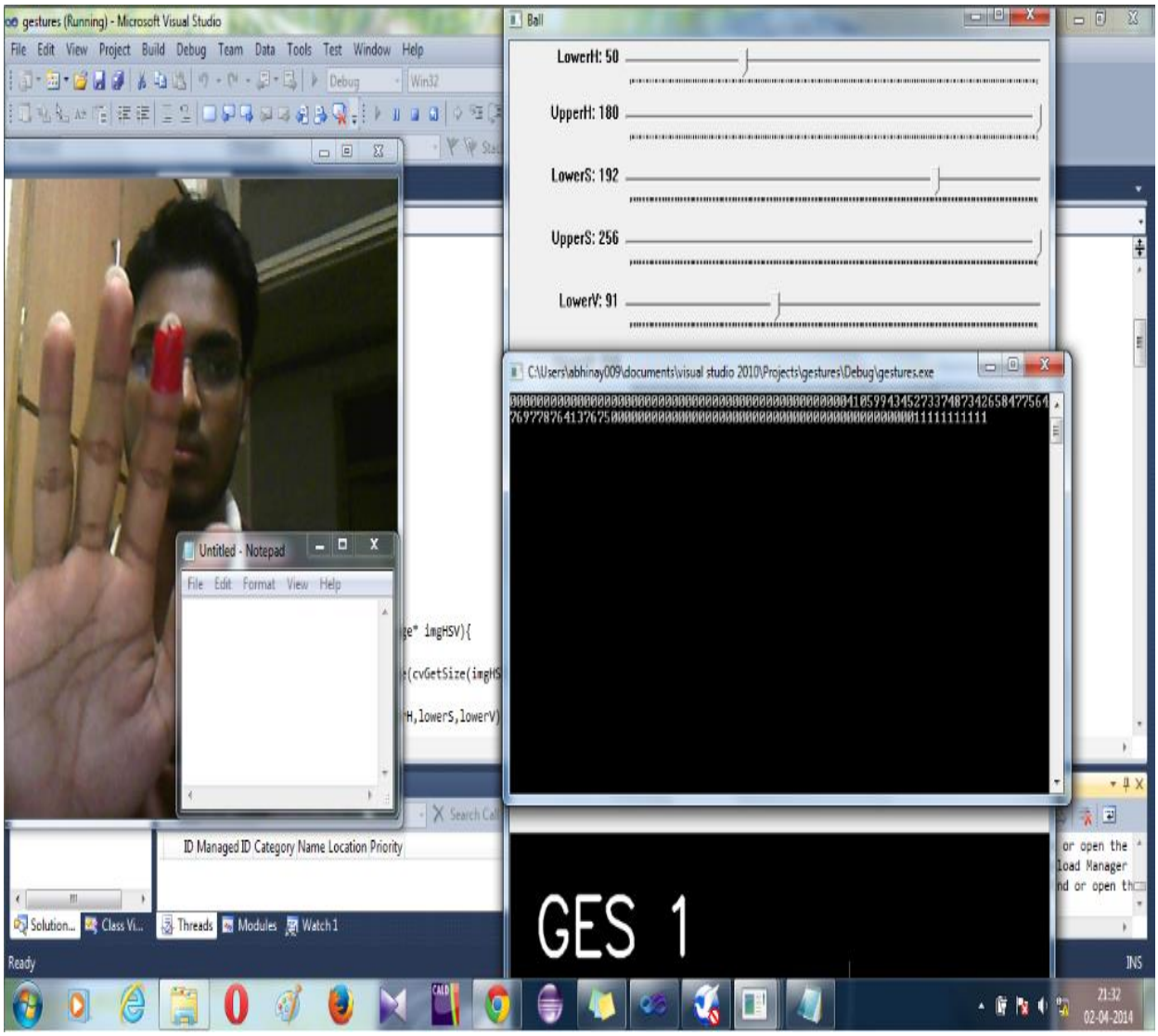


Figure 4.4(ii)

GESTURE 2: When user shows 2 fingers with colored strips, the gesture is identified as gesture number 2 and accordingly a calculator program is opened.

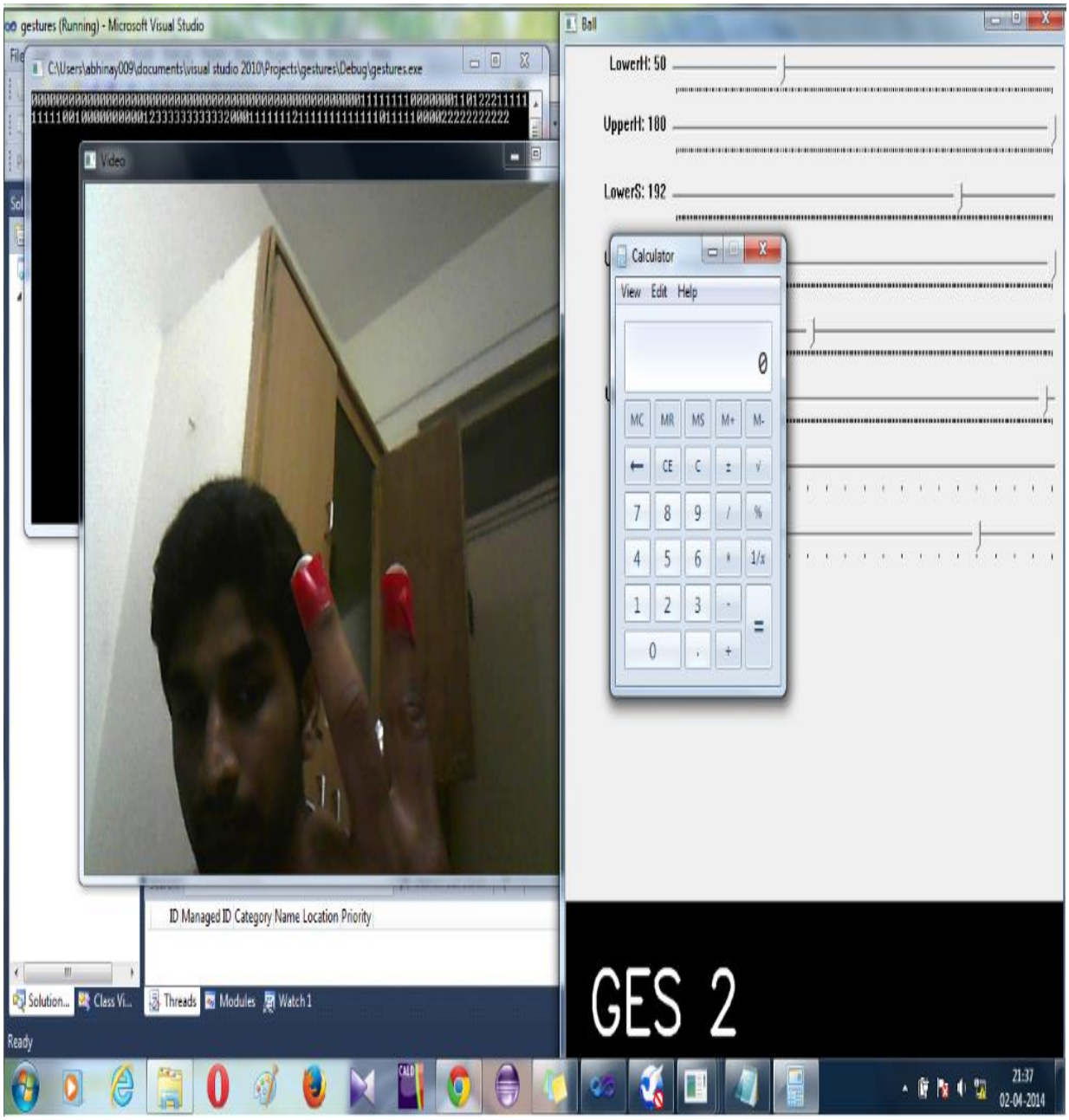


Figure 4.4(iii)

GESTURE 3: When user shows three fingers with colored strips, the gesture is identified as gesture number 3 and accordingly mspaint program is opened.

Figure 4.4(iv)

GESTURE 4: When user shows four fingers with colored strip, the gesture is identified as gesture number 4 and accordingly windows media player program is opened.

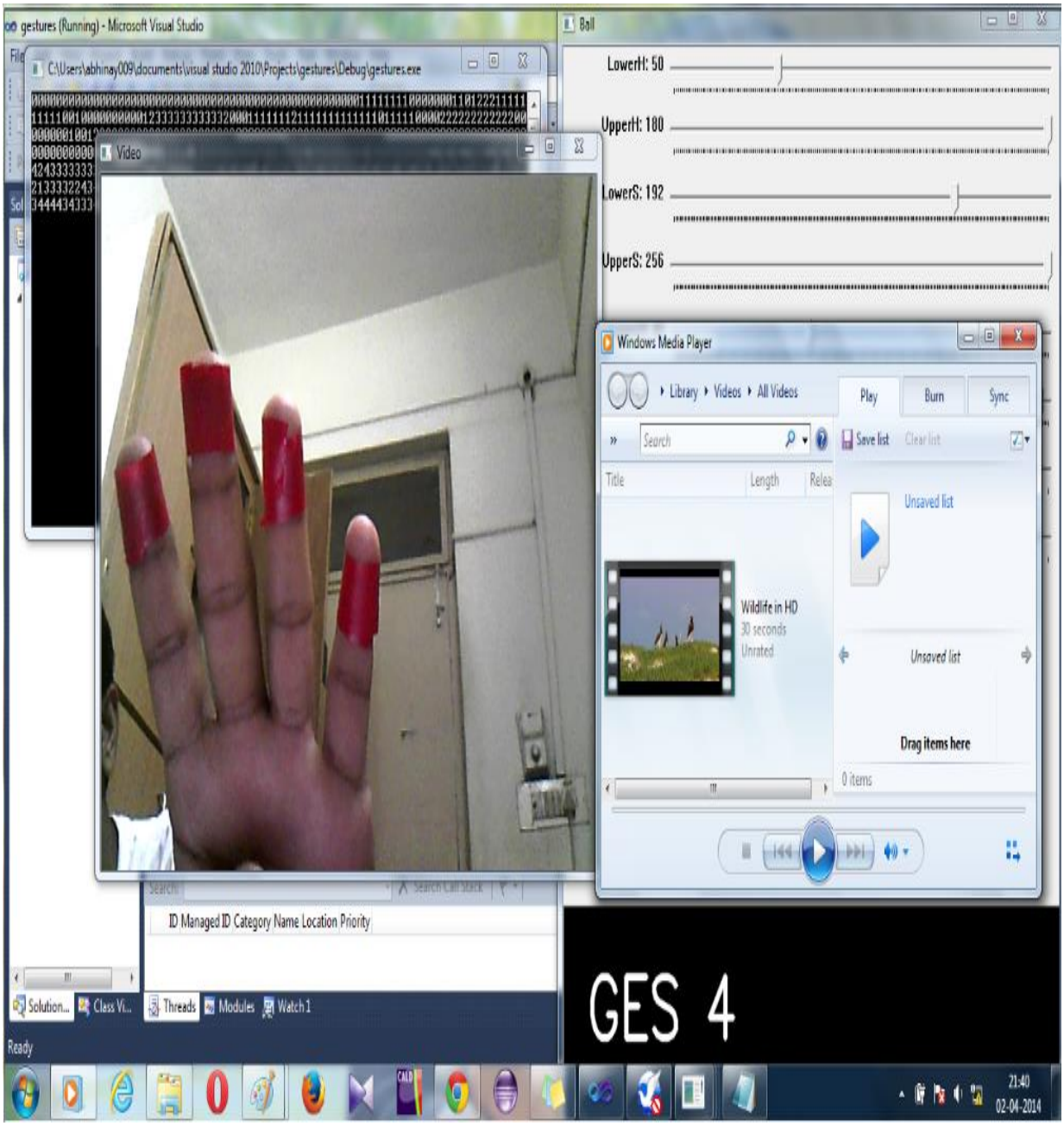


Figure 4.4(v)

GESTURE 5: When user shows five finger with colored strips, the gesture is identified as gesture number 5 and accordingly a wordpad program is opened.

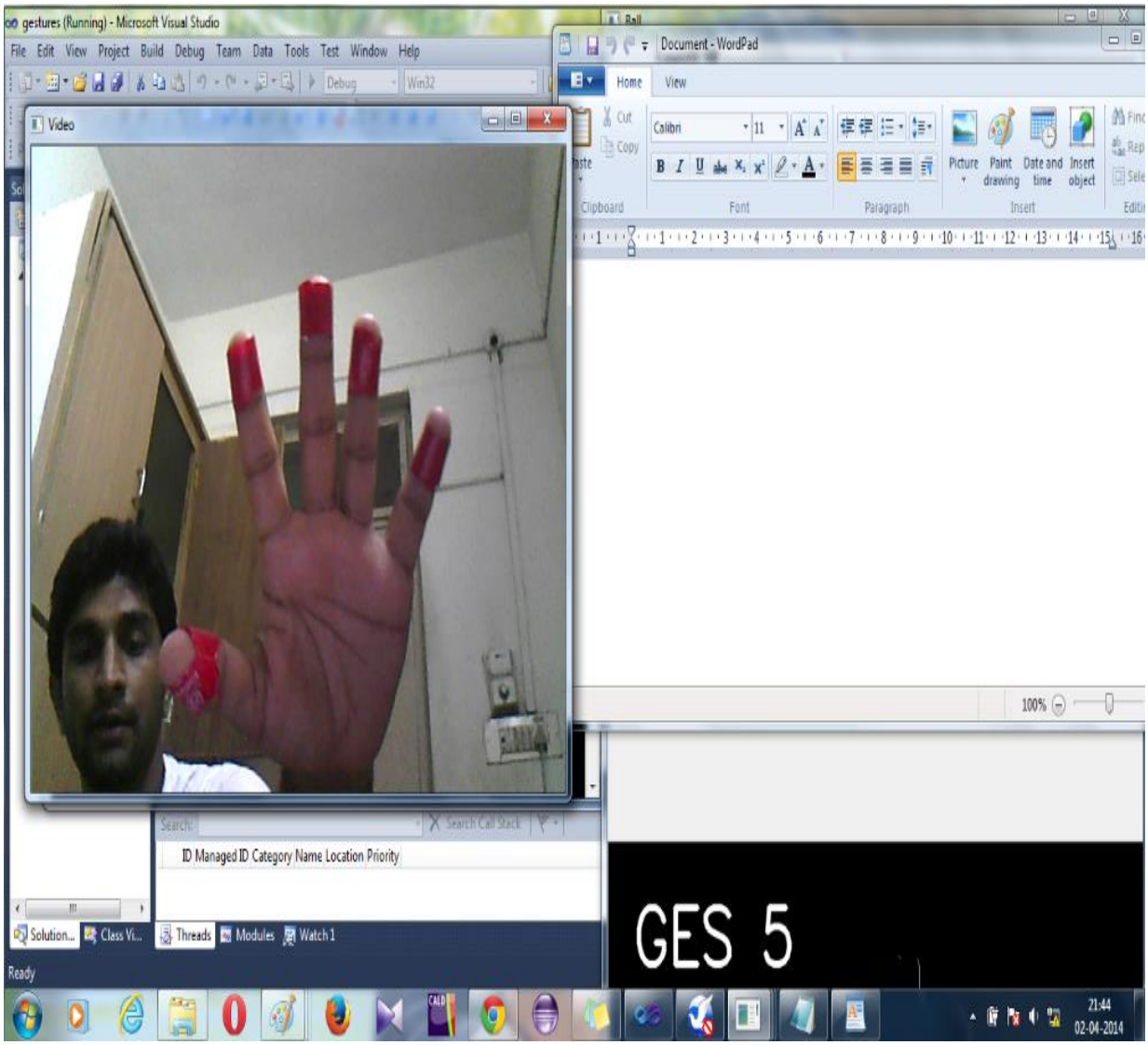


Figure 4.4(vi)

4.5 TESTING:

When no object of the specified color is found, User is prompted with an error message of no gesture.

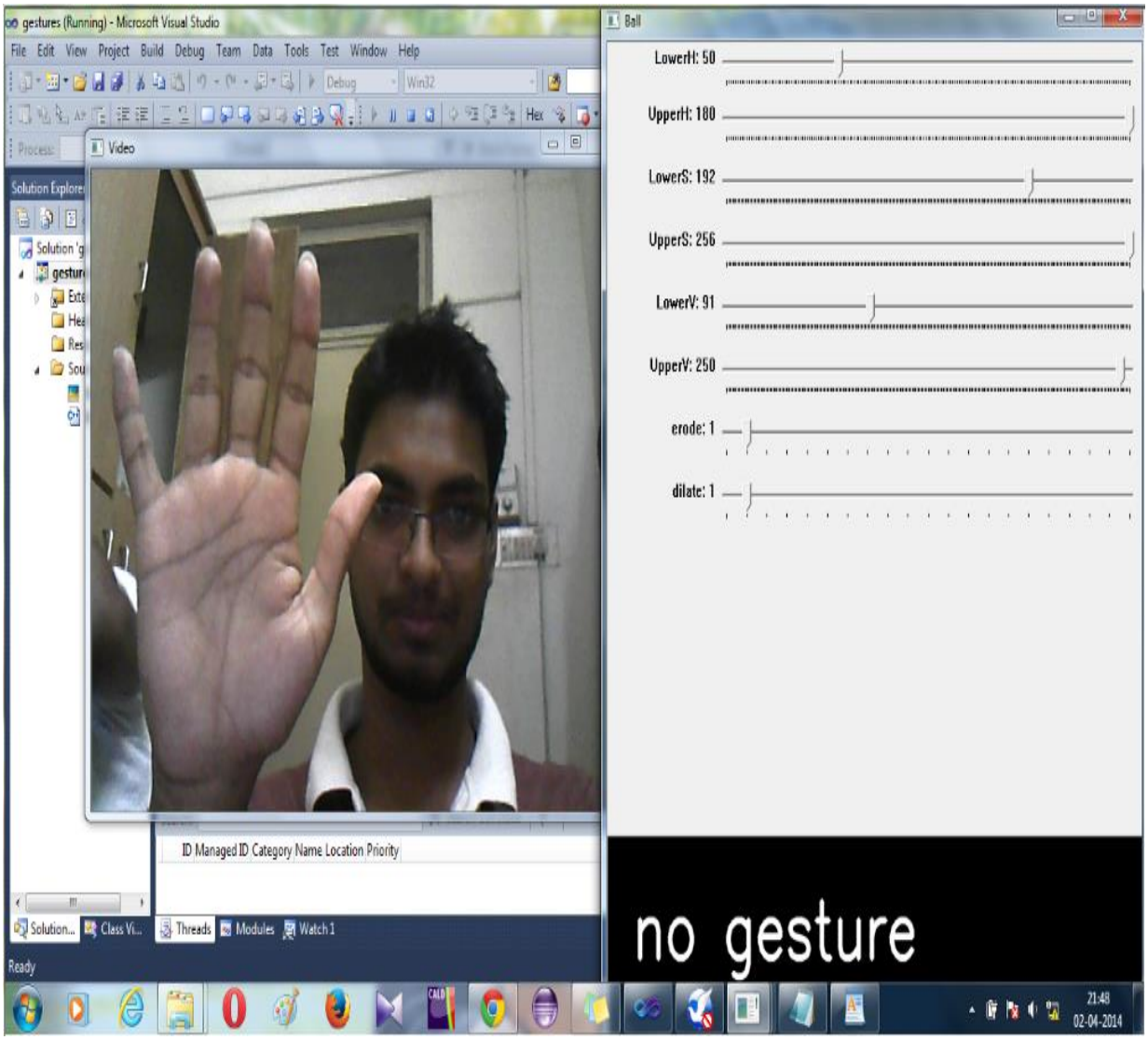


Figure 4.5(i)

If the input video frame consists of too many colored objects which are in the range of specified HSV values, then user is prompted with a message of too many gestures.

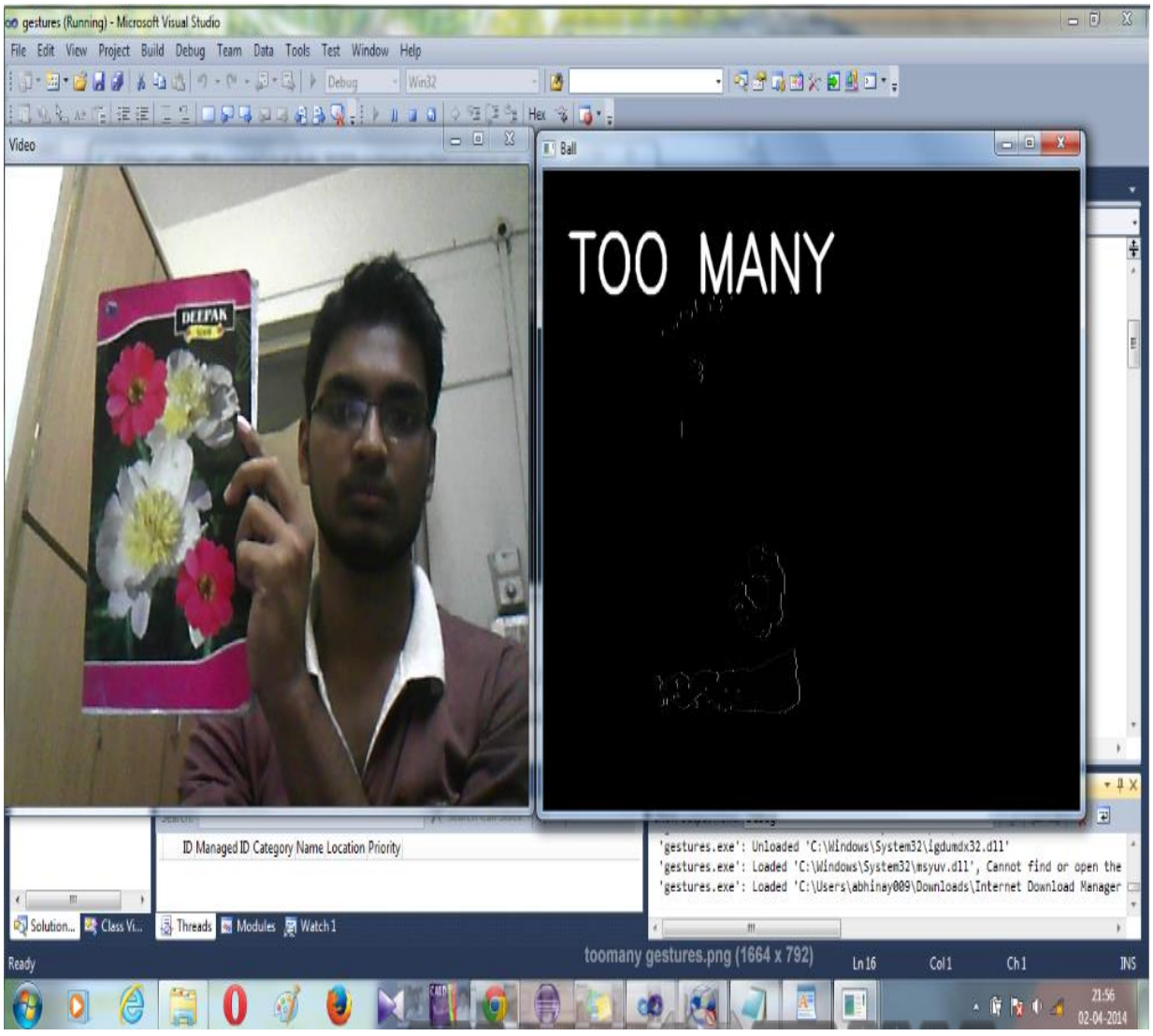


Figure 4.5(ii)

5. CONCLUSIONS AND FUTTURE SCOPE:

This is the preliminary phase of gesture recognition that we have implemented. This system is capable of identifying various gestures using colored markers detection. The advantage of this system is that the color of the markers can be chosen dynamically using the track bars. We have used the method of dividing the threshold image into contours to count the number of opened fingers. For precise recognitions we have performed erosion and dilation operation on image frames. By counting the number of contours, the gesture is identified. By relating the gesture as input we are performing related actions like, opening a notepad or calculator or internet explorer etc.

The future work of this project includes simulation of mouse operations using gestures. Also performing skin color detection based hand gesture recognition.

This technology can be used in –

i. Robotics.

ii. Gaming.

iii. Systems which could understand human behavior based on their way of interaction.

iv. Enable disabled (handicapped) people to interact with computers with much ease.

6. REFERENCES:

1) Archana S. Ghotkar & Gajanan K. Kharate: "HAND SEGMENTATION TECHNIQUES TO HAND GESTURE RECOGNITION FOR NATURAL HUMAN COMPUTER INTERACTION" International Journal of Human Computer Interaction (IJHCI), Volume (3) : Issue (1) : 2012

2) Sagar Badve , Parik Soni, Sachin Handibag, Sharad Madane: "GESTURE AND VOICE BASED INTERACTIVE SYSTEMS" International Journal of Innovative Research and studies(IJIRS), Volume (2) : Issue (6) : 2013

3) International Journal of Computer Science & Engineering Survey (IJCSES) Vol.1, No.2, November 2010 "SURVEY ON VARIOUS GESTURE RECOGNITION TECHNIQUES FOR INTERFACING MACHINES BASED ON AMBIENT INTELLIGENCE" Harshith.C, Karthik.R.Shastry, Manoj Ravindran, M.V.V.N.S Srikanth, Naveen Lakshmikhanth

4) International Journal of Computer Applications (0975 – 8887) Volume 50 – No.7, July 2012 38 "Survey on Various Gesture Recognition Technologies and Techniques" by RafiqulZaman Khan and Noor Adnan Ibraheem

7. APPENDICES:

7.1 CODE FOR GESTURE RECOGNITION

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv2\opencv.hpp>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string>
using namespace cv;
int lowerH=50;
int lowerS=192;
int lowerV=91;
int upperH=180;
int upperS=256;
int upperV=250;
int erodecount=1;
int maxerode=20;
int dilatecount=1;
int maxdilation=20;
IplImage* GetThresholdedImage(IplImage* imgHSV){
       IplImage* imgThresh=cvCreateImage(cvGetSize(imgHSV),IPL_DEPTH_8U, 1);
       cvInRangeS(imgHSV, cvScalar(lowerH,lowerS,lowerV),
cvScalar(upperH,upperS,upperV), imgThresh);
       return imgThresh;
}

void setwindowSettings(){
```

```
        cvNamedWindow("Video");
        cvNamedWindow("Ball");
        cvCreateTrackbar("LowerH", "Ball", &lowerH, 180, NULL);
        cvCreateTrackbar("UpperH", "Ball", &upperH, 180, NULL);
        cvCreateTrackbar("LowerS", "Ball", &lowerS, 256, NULL);
        cvCreateTrackbar("UpperS", "Ball", &upperS, 256, NULL);
        cvCreateTrackbar("LowerV", "Ball", &lowerV, 256, NULL);
        cvCreateTrackbar("UpperV", "Ball", &upperV, 256, NULL);
        createTrackbar("erode","Ball", &erodecount, maxerode, NULL);
        createTrackbar("dilate","Ball", &dilatecount, maxdilation, NULL);
}
int main(){

        CvCapture* capture =0;
 capture = cvCaptureFromCAM(0);
 if(!capture){

         printf("Capture failure\n");
   return -1;
 }

 IplImage* frame=0;
 vector<vector<Point> > contours;
 vector<Vec4i> hierarchy;

 int j,framenum=0,framenum1=0,framenum2=0,framenum3=0,framenum4=0;
 String a;
 setwindowSettings();
 Mat canny_output;
 RNG rng(12345);
 while(true){
  frame = cvQueryFrame(capture);
  if(!frame)  break;
  frame=cvCloneImage(frame);
```

```
   IplImage* imgHSV = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 3);

   cvCvtColor(frame, imgHSV, CV_BGR2HSV);

   IplImage* imgThresh = GetThresholdedImage(imgHSV);

   cvErode(imgThresh, imgThresh, 0, erodecount);

   cvDilate(imgThresh, imgThresh, 0, dilatecount);

   cv::Mat mrix = cv::cvarrToMat(imgThresh);

   findContours( mrix,contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE,
Point(0, 0) );

   std::cout << contours.size();


   j=contours.size();

   if(j==0)
   {
   a="no gesture";framenum=0;framenum1=0;framenum2=0;framenum3=0;framenum4=0;
   }

   else if(j==1)
   {
   a="GES 1";framenum++;framenum1=0;framenum2=0;framenum3=0;framenum4=0;
   }

   else if(j==2)
   {
   a="GES 2";framenum=0;framenum1++;framenum2=0;framenum3=0;framenum4=0;
   }

   else if(j==3)
   {
   a="GES 3";framenum=0;framenum1=0;framenum2++;framenum3=0;framenum4=0;
   }

   else if(j==4)
   {
   a="GES 4";framenum=0;framenum1=0;framenum2=0;framenum3++;framenum4=0;
   }

   else if(j==5)

   {
   a="GES 5";framenum=0;framenum1=0;framenum2=0;framenum3=0;framenum4++;
   }

   else
   {
   a="TOO MANY";framenum=0;framenum1=0;framenum2=0;framenum3=0;framenum4=0;
   }

   putText(mrix,abhi,cvPoint(30,90),FONT_ITALIC,2,cvScalar(255,255,255),4, CV_AA);

   if(framenum>10)
  {
                    system("notepad");
```

```
                framenum=0;framenum1=0;framenum2=0;framenum3=0;framenum4=0;

   }
  else if(framenum1>10)
   {
                system("calc");

                framenum=0;framenum1=0;framenum2=0;framenum3=0;framenum4=0;
   }
   else if(framenum2>10)
   {
                system("mspaint");

                framenum=0;framenum1=0;framenum2=0;framenum3=0;framenum4=0;
   }
   else if(framenum3>10)
   {
                system("start wmplayer");

                framenum=0;framenum1=0;framenum2=0;framenum3=0;framenum4=0;
   }
    else if(framenum4>10)
   {
             system("write");

             framenum=0;framenum1=0;framenum2=0;framenum3=0;framenum4=0;
   }
   else{}

   imshow("Ball",mrix);

   cvShowImage("Video", frame);

   cvReleaseImage(&imgHSV);

   cvReleaseImage(&imgThresh);

   cvReleaseImage(&frame);


   int c = cvWaitKey(80);

   if((char)c==27 ) break;

 }


 cvDestroyAllWindows();

 cvReleaseCapture(&capture);

       return 0;
}
```