A Project Report on

Implementation of Image Processing Algorithm for Model Smart Home

Project submitted in partial fulfilment of
the requirement for the award of the degree of

Bachelor of Technology Hons. (B.Tech Hons.)
in
Electronics and Communication Engineering

by

Vivek Kumar
Roll: 555/10

Anish Agrawal
Roll: 070/10

Yogesh Kumar Gupta
Roll: 561/10

Department of Electronics and Communication Engineering

NATIONAL INSTITUTE OF TECHNOLOGY

JAMSHEDPUR

APRIL 2014

BONAFIDE CERTIFICATE

This is to certify that the project entitled "**IMPLEMENTATION OF IMAGE PROCESSING ALGORITHM FOR MODEL SMART HOME"** is the bonafide work done by:

Vivek Kumar (EL110555)

Anish Agrawal (EL110070)

Yogesh Kumar Gupta (EL110561)

that is submitted in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology (B.Tech)** in **Electronics and Communication Engineering(ECE)** by **NATIONAL INSTITUTE OF TECHNOLOGY, JAMSHEDPUR** during the academic year 2010-2014.

Prof. Amit Prakash	**Dr. S. N. Singh**
(Associate Professor)	(Professor and Head)
Project Guide	

Project viva-voce held on ________________________________

ACKNOWLEDGEMENTS

The real spirit of achieving a goal is through the way of excellence and austere discipline. We would have never succeeded in completing our task without the cooperation, encouragement and help provided to us by various personalities.

It is a matter of great pleasure to acknowledge the invaluable guidance, enormous assistance and excellent co-operation extended in completing the mini project by our mentor and guide Prof. Amit Prakash. We are very privileged to thank him for extending his continuous support.

We would also like to thank Prof. S. N. Singh (Prof. and Head, Dep't of ECE), for showing faith in our abilities and providing facilities and infrastructure in the department.

We would also like to place on record the contribution of our dear friends Vivek Kumar and Sujit Kumar Dutta, both of Department of Electrical and Electronics as well as Alok Kumar Jha, Department of Electronics and Communication for their valuable contribution in developing the final prototype.

We are also thankful to all the professors of the department for their support and encouragement. We must acknowledge the academic resources that we have got from NIT Jamshedpur.

We would conclude with my deepest gratitude to our parents and all our loved ones. Our full dedication to the work would have not been possible without their blessings and moral support.

Vivek Kumar
vkvivekkumar.2010@gmail.com

Anish Agrawal
anish818@gmail.com

Yogesh Kumar Gupta
ugeshgupta000@gmail.com

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-1
INTRODUCTION

1.1 Background:

Pattern analysis is one of the active research topics in digital image processing. The technology is leaping into so much advancement that image recognition will become part and parcel of our daily lives. Applications such as Ultra sound, MRI use image processing to detect broken bones, tissues, tumours and various kind of diseases and are used for various other industrial applications. Contemporary it is used for detecting airport luggage scanning and for detecting the quality of food grains to detect fungi and other micro diseases. So now in modern era image processing is used for security reasons through thumb print recognition, eye retina detection and then for crime detection it is also used for face recognition. Satellite imagery is used to detect crop growing patterns, their cultivated area; advance warning to farmers in Australia is given for pesticide and disease. One of the state of the art applications is the cruise missile guidance system developed by US Defence to Map the territory for possible accurate target selection by using GIS and image processing. Even in Pakistan Sugar mills are using satellite imagery for planning their sugar cane purchasing campaign and procurement planning. Imaging for medical reasons CT- Scan, MRI and Ultra Sound is now-a-days a common content in the patient's medical history, detection of fungi is good topic for research. Researchers have put lot of effort in image processing. Image processing also holds a lot of potential in changing the dynamics of how a human interacts with a machine. This is explored in further sections.

1.2 Literature Survey:

Different techniques have been used to detect and track hand gestures. Another common way to track hand gestures is to use two cameras. The cameras can be placed in front or on the side [14] to be able to triangulate and identify the exact 3D position of the hand. A stereo camera [13], which consists of two cameras side by side, can provide depth information to simplify hand recognition. Although these methods achieve good results, the need for multiple cameras may not be convenient. In addition, some desktop monitors and laptops now come embedded with a single webcam.

Another approach is to detect edges using either a Sobel filter or a Canny filter, and then extract the hand shape from these edges [11]. This method appears to be efficient on a clear background. Nonetheless, these filters are sensitive to detected edges and in particular, to edges located in the background. This is why this technique is often combined with background modelling, which does not fit the requirements of our application scenario.

1.3 Motivation:

A ubiquitous computing provides advanced dynamic environments where humans want to make assorted types of interface for interaction with media and information without any physical restrictions. Computers now a days are a fundamental part of any educational core curriculum hence the educational systems also provide a very large domain of applications for ubiquitous computing. The key board, mouse etc lack the sensitivity desired in required application. Eventually, the researchers are putting in their efforts in the field of Human Computer Interaction with a common emphasis to design and develop the user interfaces capable enough fulfil the intended performance criteria desired in the dynamic environment. Also learning for students could be made easier if the complex commands of the dynamic applications of computers are controlled through easier hand gestures, for example browsing images in an image browsers with help of hand gestures or controlling the power point presentations with the usage of hand gestures without need of learning various commands involved, from a distance without the need of any physical contact with the devices. The user may benefit from such means of interaction in communicating with computing devices used in dynamic environment. Most of the users abhor the complexities of the present interaction devices and methods because of the time and attempts needed for understanding the functionalities and the chances of failures arising out of the cognitive burden.

Hand gestures enable humans to communicate naturally and the recognition of these gestures has many applications. For example, hand gestures can be used to recognize and interpret sign language and musical conducting action.

1.4 Objectives of the work:

The objective of this work is to develop a simple set of applications that demonstrate the advancements made in image processing to enable detection and extraction of useful image features and use them for human computer interaction in which a user can control a few devices and applications using just hand-gestures.

The programs developed should be simple enough to track human gestures and control the devices. But, it should also be comprehensive enough to know when the human is making a gesture to the device and when he/she is not.

The objective will be to develop robustness against unwanted device control by making the machine understand that needs to receive instructions only when a specific gesture is made by the user.

The models developed will include control of PowerPoint applications, home lighting and Windows Media Player using hand gestures. We will also develop an application, more specifically an Image Processing GUI that demonstrates some simple operations in image processing carried out in spatial domain.

The performance will be tested in MATLAB computing environment and Arduino Open Source Development platform for prototype building.

Towards the end, we will try to ascertain the accuracy of the algorithms under various conditions by collecting statistics under different light conditions, out of focus images, and other forms of degraded image. We will also make an attempt to impart some intelligence to a few algorithms by incorporating fundamental fuzzy logic in them.

CHAPTER-2
PROPOSED GESTURE CONTROLLED SYSTEM

2.1 Introduction:

From our discussion on the importance of Human Machine Interaction to make our connection with them easier and make our lives smoother, it becomes apparent to develop systems based on it. In such a scenario image processing proves to be a powerful tool as has already been seen by the examples discussed in the introductory section. This project on image processing looks at a practical application of image processing where the user comfortably sits or rests in his/her room and only needs to gesture with the hand to control the appliances in that room or run slide shows or make a few basic edits to an image or control a media player. The project makes use of a variety of development tools, software and hardware platforms to develop a prototype of the same. The proposed system in further elaborated as discussed below.

2.2 Design of the proposed system:

In this system we make use of a digital camera to capture the movement of hands of the user. The captured image is then processed using a powerful computing tool, MATLAB which is fairly well known to the engineering community, and extract useful information from the image to make a prediction about what task the user intends to perform. The MATLAB environment is interfaced with the Open source hardware platform Arduino Duemilanove to execute the instructions sent by MATLAB in accordance with what the user intends to do with home appliances.

In this project the Arduino board is used to control a few lights connected to it using hand gestures. The circuit diagrams for the connections on the Arduino board are shown below. Also the circuit to control heavy appliances by using the same board is also shown. The circuit diagrams in this section have been designed and simulated in Fritzing and Multisim. Fritzing is an open-source initiative to support designers and artists in taking the step from physical prototyping to actual product. This software was created in the spirit of Processing and

Arduino. Fritzing's goal is to allow the designer / artist / researcher / hobbyist to document their breadboard-based prototype and create a PCB layout for manufacturing.

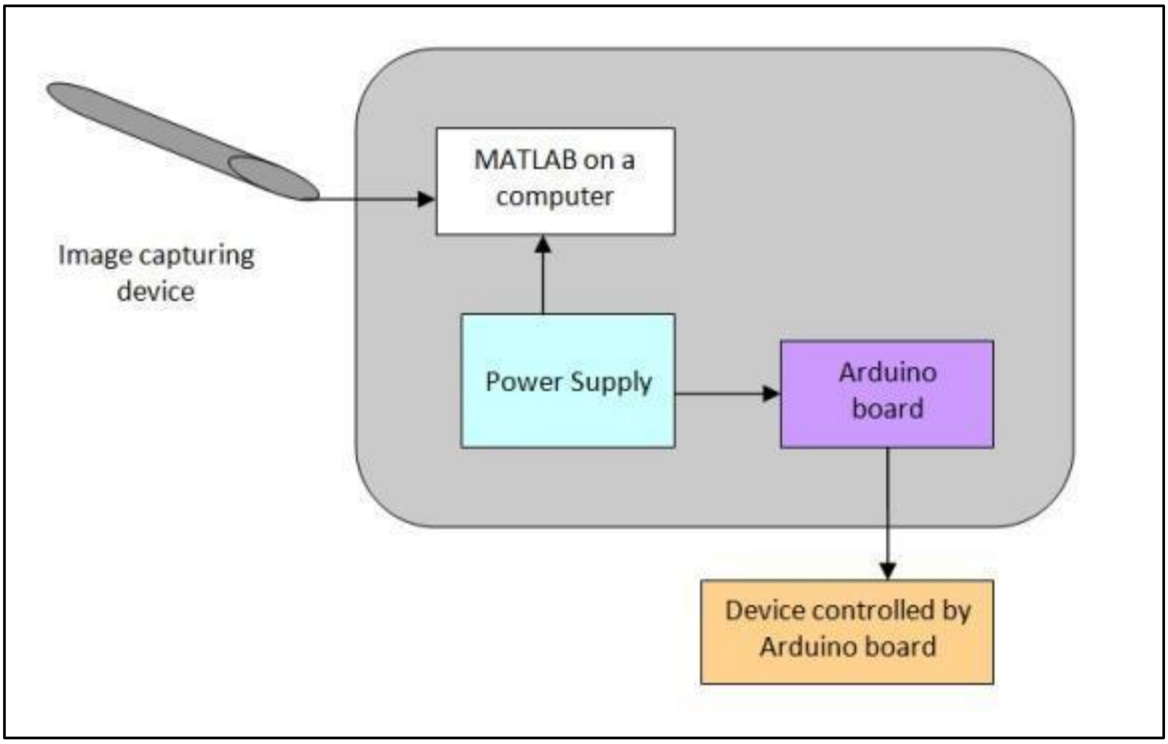


Fig 1. Block Diagram of the proposed system for light control showing major components

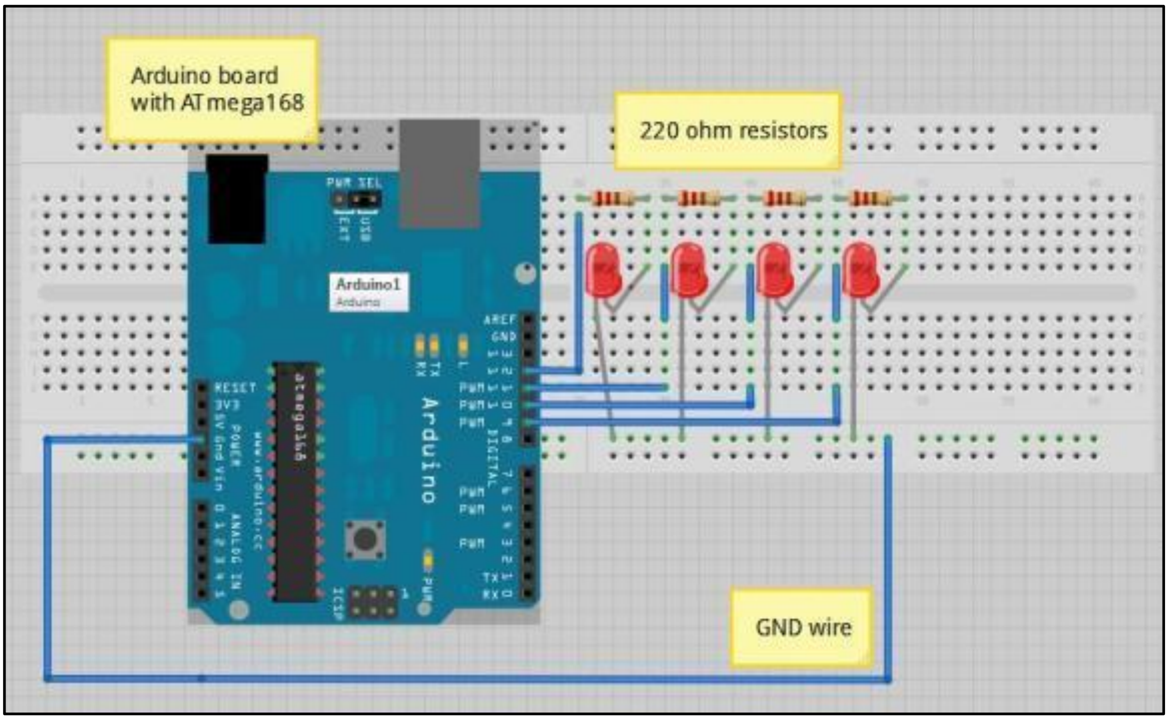


Fig 2. Fritzing image of the Arduino board showing the connections

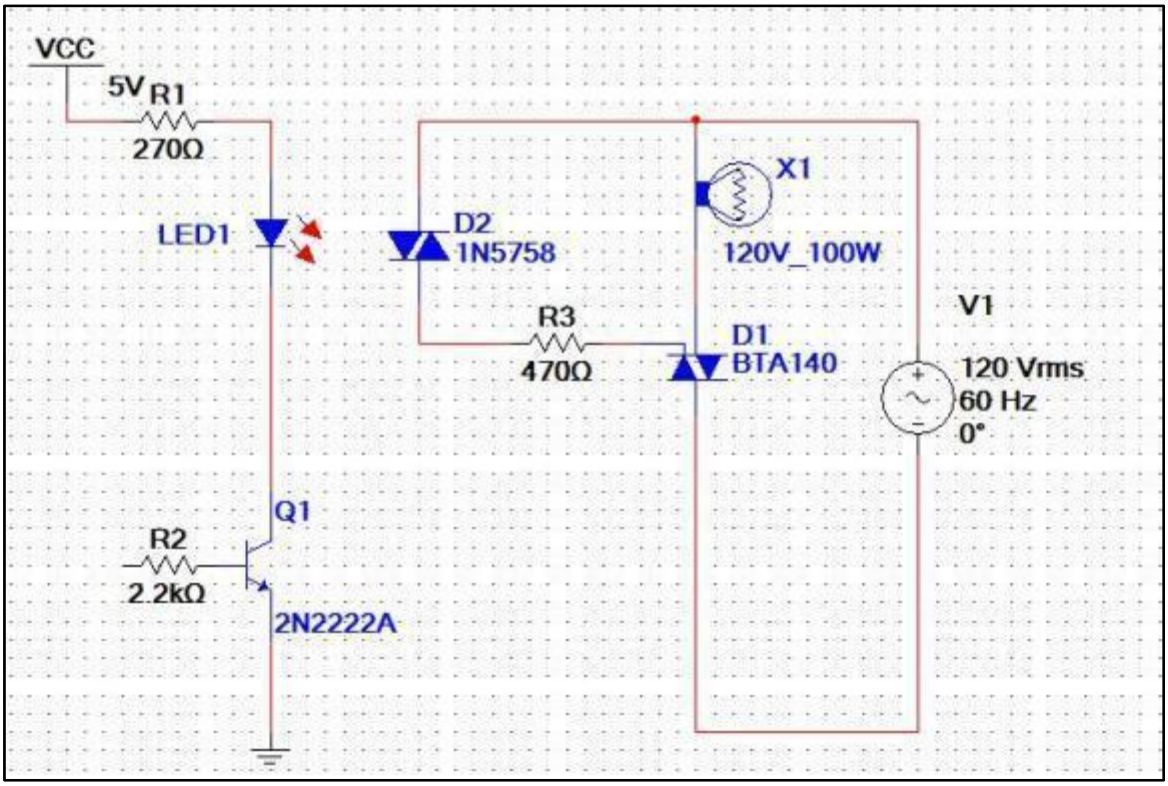


Fig 3. Circuit of a solid-state switch module developed in Multsim

The later sections of this project focus on a few distinct aspects as the MATLAB computing environment, the Arduino open source development board, and Image processing basics. Apart from that the source code developed to control the devices is also discussed. Throughout the project the goal remains to develop reliable systems that can make fairly accurate guess about what the user intends to perform, thus leading to overall high levels of consumer satisfaction.

As already elaborated in previous sections, the other models will also try to achieve a high degree of accuracy, albeit they will be implemented on the computer itself as they are applications that will be controlled by the hand gestures.

CHAPTER-3
IMAGE PROCESSING OVERVIEW

Image processing involves changing the nature of an image in order to either:

1) Improve its pictorial information for human interpretation,
2) Render it more suitable for autonomous machine perception.

E.g. enhance edges, remove noise, improve contrast, remove motion blur, detection of faces, counting of objects.

We shall be concerned with digital image processing, which involves using a computer to change the nature of a digital image.

3.1 Aspects Of Image Processing:

1) **Image enhancement:** Sharpening or de-blurring an out of focus image, highlighting edges, improving image contrast, or brightening an image.

2) **Image restoration:** Removing of blur caused by linear motion, removal of optical distortions.

3) **Image segmentation:** This involves subdividing an image into constituent parts, or isolating certain aspects of an image finding lines, circles, or particular shapes in an image, in an aerial photograph, identifying cars, trees, buildings, or roads.

3.2 Tasks In Image Processing:

1) **Acquiring the image:** First we need to produce a digital image from a paper envelope. This can be done using either a CCD camera, or a scanner.

2) **Preprocessing:** This is the step taken before the “major” image processing task. The problem here is to perform some basic tasks in order

to render the resulting image more suitable for the job to follow. In this case it may involve enhancing the contrast, removing noise, or identifying regions likely to contain the postcode.

3) **Segmentation:** Here is where we actually "get" the postcode; in other words we extract from the image that part of it which contains just the postcode.

4) **Representation and description:** These terms refer to extracting the particular features which allow us to differentiate between objects. Here we will be looking for curves, holes and corners which allow us to distinguish the different digits which constitute a postcode.

5) **Recognition and interpretation:** This means assigning labels to objects based on their descriptors (from the previous step), and assigning meanings to those labels. So we identify particular digits, and we interpret a string of four digits at the end of the address as the postcode.

A digital image differs from a photo in that the x, y and f(x, y) values are all discrete. Usually they take on only integer values, so the images will have x, y ranging from 1 to 256 each, and the brightness values also ranging from 0 (black) to 255 (white).

3.3 Types Of Digital Images:

In image processing, the following three formats of digital images are very popular:

1) **Binary:** Each pixel is just black or white. Since there are only two possible values for each pixel, we only need one bit per pixel. Such images can therefore be very efficient in terms of storage. Images for which a binary representation may be suitable include text (printed or handwriting), fingerprints, or architectural plans.

2) **Greyscale:** Each pixel is a shade of grey, normally from 0 (black) to 255 (white). This range means that each pixel can be represented by eight bits, or exactly one byte. This is a very natural range for image file handling. Other greyscale ranges are used, but generally they are a power of 2. Such images arise in medicine (X-rays), images of printed

works, and indeed 256 different grey levels are sufficient for the recognition of most natural objects.

3) **True colour or RGB:** Here each pixel has a particular colour; that colour being described by the amount of red, green and blue in it. If each of these components has a range 0-255, this gives a total of 255^3=16,777,216 different possible colours in the image. This is enough colours for any image. Since the total number of bits required for each pixel is 24, such images are also called 24-bit colour images. Such an image may be considered as consisting of a “stack” of three matrices; representing the red, green and blue values for each pixel. This means that for every pixel there correspond three values.

3.4 Image Sizes:

Suppose the image has 256 rows and 256 columns. Then the image has 256x256 pixels. Size of the image in three formats is:

1) **Binary:** Each pixel takes only 1 bit, so size is 256*256 bits.

2) **Greyscale:** Each pixel requires 8 bits, so size is 256*256*8 bits.

3) **True colour or RGB:** Each pixel requires 24 bits, 8 each for red, green and blue. So size is 256*256*8*3 bits

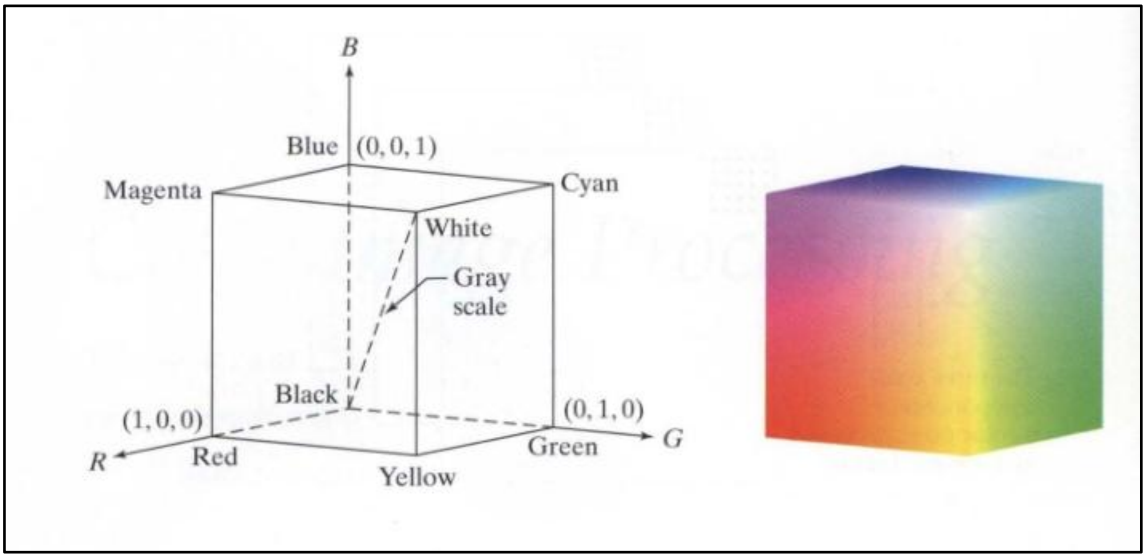


Fig 4. RGB colour model cube

CHAPTER-4
ARDUINO OVERVIEW

One of the most fantastic outcomes of allowing use of technology by artists, designers, hobbyists and almost anyone has been the ‘Arduino’ prototyping platform. It has emerged has a powerful new tool to allow us to create several DIY projects ranging from a simple array of blinking LED’s to gesture controlled robots.

4.1 Development and Origin:

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. Arduino not only stands for the single microcontroller board, it also refers to the development environment used for writing the program to the board.

The hardware consists of a simple open hardware design for the Arduino board with an Atmel AVR processor and on-board input/output support. Just like the ATmega series has several development boards to suite varying needs, Arduino too has several prototyping platforms. Some of them may offer more number of pins for I/O; other may have connections for stepper motors while still others may support high end sensors. Users have a variety to choose from, based on their needs. Some of the most popular Arduino boards are Arduino Uno, Arduino Duemilanove and LilyPad Arduino. Apart from the USB, most Arduino boards also have an external DC power supply connector to meet on-board power needs.

Looking into the software side, the open source IDE can be downloaded for free and needs no installation. The IDE has an editor to write the codes and several menus to set ports, drivers, load the program on the board as well as general functions like cut, copy and paste. The Arduino IDE comes with a C/C++ library called "Wiring" (from the project of the same name, Arduino was born from this project), which makes many common input/output operations much easier.

4.2 Building a code in Arduino:

Here we take up an example of blinking an LED which is built onto the Arduino board. We will be using the Arduino Uno board for this purpose.

- Once you have downloaded the Arduino IDE (available at http://www.arduino.cc), simply run the application without the need to install it.

- Then we need to select the type of Arduino board under 'Tools' option in the menu. Here our choice will be the Arduino Duemilanove board.

- Now we work on writing the code. On an Arduino Uno board, the in-built LED is connected to PIN number 13. So first we give a name to this PIN as follows:

```
    int ledPin = 13;
```

- Any Arduino code essentially consists of two functions:

setup() – a function run once at the start of a program that can initialize PIN's and other settings

loop() – a function called repeatedly until the board powers off.

Now we write the code for setup() function:

```
void setup()
{
    //initialize pins
    pinMode(ledPin, OUTPUT); //setting PIN 13 as output
}
```

The code for the loop() function looks like below:

```
void loop()
{
    digitalWrite(ledPin, HIGH); //drives the LED ON
    delay(1000); //1 sec wait before LED switches off
    digitalWrite(ledPin, LOW); // drives LED OFF
    delay(1000);
}
```

Once this is done, we connect the Arduino board to the PC via USB cable. On some systems we may require to install the drivers for Arduino which can be conveniently done in the device manager of the computer.

Next the code is compiled and burned onto the board.

Once the board is powered up by means of the USB or a 5V dc, the LED starts blinking. The setup () function initializes the necessary parameters and loop function is continuously executed till we switch off the supply.

This project makes use of the Arduino Duemilanove board with an on-board ATmega328 microcontroller for control of consumer appliances. Specifications for this board are given in Appendix 1.

4.3 Interfacing Arduino board with MATLAB

UPLOADING ADIOSRV.PDE TO THE ARDUINO BOARD (to be done only once):

The adiosrv.pde (or srv.pde) is the "server" program that will continuously run on the microcontroller. It listens for MATLAB commands arriving from the serial port, executes the commands, and, if needed, returns a result.

The instructions given below are for Windows Operating System Vista, 7 and above. Instructions for other OSs differ and can be found on the website of Mathworks.

From the Arduino IDE, go to File > Open, locate the file adiosrv.pde, (in the ArduinoIO/pde/adiosrv folder) and open it. If a dialog appears asking for the permission to create a sketch folder and move the file, press OK (this will create an adiosrv folder and move the adiosrv.pde file inside it).

Connect the Arduino, make sure that the right board and serial port are selected in the IDE, (Tools/Board and Tool/Serial Port) then select File -> Upload to I/O Board and wait for the "Done Uploading" message.

At this point the adiosrv.pde file is uploaded the IDE should be closed, which is not needed anymore for the purpose of this package. Actually closing the IDE is suggested, so that the serial connection to the Arduino board is not taken by the IDE when MATLAB needs to use it.

FINAL PRELIMINARY STEPS (to be done only once):

On Windows 7 or Vista, run MATLAB as administrator by right-clicking on the MATLAB icon and select "Run as Administrator". This will allow the updated path to be saved. From MATLAB, the "install_arduino" command is launched; this will simply add the relevant ArduinoIO folders to the MATLAB path and save the path.

CHAPTER-5
MATLAB OVERVIEW

MATLAB is a powerful computing environment for handling the calculations involved in scientific and engineering problems. The name MATLAB stands for MATrix LABoratory, because the system was designed to make matrix computations particularly easy.

One of the many things that is liked about MATLAB (and which distinguishes it from many other computer programming systems, such as C++ and Java) is that it can be used interactively. This means we type some commands at the special MATLAB prompt, and get the answers immediately. The problems solved in this way can be very simple, like finding a square root, or they can be much more complicated, like finding the solution to a system of differential equations. For many technical problems we have to enter only one or two commands, and we get the answers at once. MATLAB does most of the work for us.

5.1 Getting around MATLAB

The interface of MATLAB is very clean, appealing and can be modified according to our needs.

Next we look at some of the main windows to explore in MATLAB:

1) **The Command Window:** The command window is the primary code execution window. Here we enter codes, execute them and see the results. Each code is executed after a 'double greater than', '>>' symbol.
2) **The workspace:** The workspace is the window which shows us the list of all variables which we have declared while executing a code. These can include numbers, arrays, character strings and vectors and matrices.
3) **The command history window:** The command history window holds the list of all the list of all the commands that have been executed in the command window.

4) **The Help window:** MATLAB possibly has the most comprehensive help tool one will find in any software. As most users of MATLAB would say, MATLAB itself can be the best teacher. If there is have problem using a specific inbuilt function simply type 'help' followed by the function name in the command window and MATLAB opens for the user a comprehensive detail about the function.
5) **Toolboxes in MATLAB:** Toolboxes in MATLAB are modules that contain all the functions that one would need to work while working on a particular model. This is very similar to header files in C/C++. The only thing is that the toolboxes in MATLAB are much more intelligent, and then come with a Graphical User Interface of their own.

5.2 Image Processing in MATLAB:

In this project of ours, we will be making use of the commands and functions available in the Image Acquisition Toolbox and Image Processing Toolbox of MATLAB. The most common commands that we will come across are detailed below:

```
>>i=imread('abc.jpg');              %ABC in any format, such as jpg, gif etc
>>imshow(i);                        %show the image
>>j=rgb2gray(i);                    % j is grayscale of i
>>imhist(j);                        %see the histogram, i should be 2-dimensional.
                                    %so use grayscale image or i(:, :, 1)
>>imwrite( j, 'New.jpg', 'jpg');    %save to disk the grayscale image
>>i=imadjust(i);                    %improves contrast
>>i=immultiply(i, 2);               %lightens image, enter value less than 1
                                    %to darken the image, e.g. 0.5
>>imshow(i);                        % you will see lighter image
>>i=imdivide(i, 2);                 %similar to entering value less than 1 in immultiply
>>k=imcomplement(i);                %negative of image, accepts all image formats
>>l=im2bw(k);                       %converts to binary image
>>m=imfinfo('abc.jpg');             %displays all the information, m is a structure
```

CHAPTER-6
BASIC IMAGE PROCESSING STEPS

Before undertaking a study on various image segmentation and restoration techniques, it is imperative to look at how the image degradation is modelled mathematically, and what are some of the ways to obtain noisy images.

6.1 Model for image degradation and restoration

The degradation process is modeled in this chapter as a degradation function that, together with an additive noise term, operates on an input image f(x , y) to produce a degraded image g(x , y).

$$g(x, y) = H\left[f(x, y)\right] + \eta(x, y)$$

If H is a linear, spatially invariant process, it can be shown that the degraded image is given in the spatial domain by:

$$g(x, y) = h(x, y) \star f(x, y) + \eta(x, y)$$

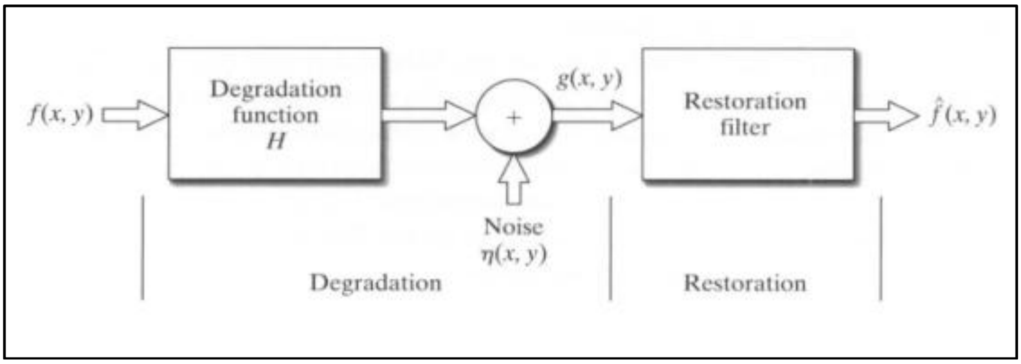


Fig 5. Image Degradation and Restoration model

Adding Noise to Images with Function imnoise

The Image Processing Toolbox uses function imnoise to corrupt an image with noise. This function has the basic syntax

g = imnoise(f, type, parameters)

where f is the input image, and type and parameters are as explained below. Function imnoise converts the input image to class double in the range [0, 1] before adding noise to it. This must be taken into account when specifying noise parameters. For example, to add Gaussian noise of mean 64 and variance 400 to a uint8 image, we scale the mean to 64/255 and the variance to $400/(255)^2$ for input into imnoise.

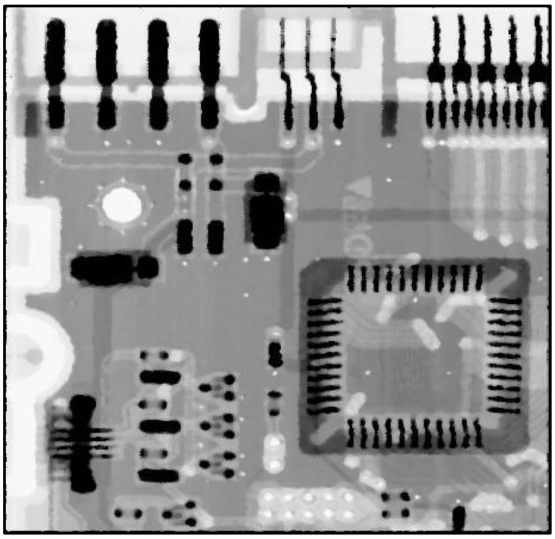


Fig 6. Original image of a printed circuit board

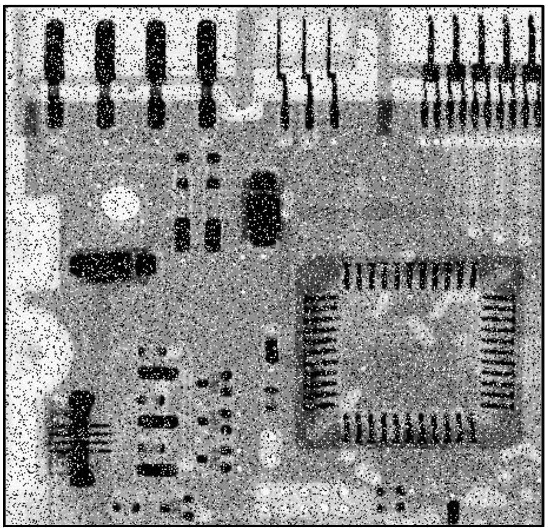


Fig 7. Same image corrupted with salt and pepper noise

6.2 Capturing single images in MATLAB:

```
v=videoinput('winvideo', 2);
i=getsnapshot(v);
imtool(i);
j=ycbcr2rgb(i);
imtool(j);
```

obj = videoinput(adaptorname, deviceid) constructs the video input object, v. A video input object represents the connection between MATLAB and a particular image acquisition device. ‘adaptorname’ is a text string that specifies the name of the adaptor used to communicate with the device. The ‘imaqhwinfo’ function is used to determine the adaptors available on the system.

i = getsnapshot(obj) immediately returns one single image frame, frame, from the video input object, obj. The frame of data returned is independent of the FramesPerTrigger property, and has no effect on the FramesAvailable or FramesAcquired property.

imtool(frame) is used to view the captured object in Image Frame. Alternatively, imshow may also be used.

YCBCR2RGB converts YCbCr colour values to RGB colour space. RGBMAP = YCBCR2RGB(YCBCRMAP) converts the YCbCr values in the colourmap. YCBCRMAP to the RGB colour space. If YCBCRMAP is M-by-3 and contains the YCbCr luminance (Y) and chrominance (Cb and Cr) colour values as columns, then RGBMAP is an M-by-3 matrix that contains the red, green, and blue values equivalent to those colours.

Alternatively, images may also be captured using the camera available in smart phones running on Android Operating System. For this, we need to install an application IP WebCam on the cell phone, the application can stream the recorded videos in the form of jpg images to a browser or streaming media player. The code to accomplish the same is given below:

```
url = 'http://27.62.220.3:8080/shot.jpg';
ss = imread(url);
fh = image(ss);
while(1)
ss = imread(url);
set(fh,'CData',ss);
drawnow;
pause(4);
end
```

The IP address may change according to the network connection. The code has been written in MATLAB.

6.3 Capturing live video feed in MATLAB:

As has already been discussed, the first step would be to capture the image of gesture of the user and then use MATLAB to make a prediction about what the user intends to do. To make that possible the images need to be captured continuously, or in other words we need a video to detect user gestures. The code for the same look like below:

```
v=videoinput('winvideo',2, 'YUY2_640x480');
s=serial('COM44');
fopen(s);
while 1
i=getsnapshot(v);
end
```

6.4 Feature extraction from captured images:

- The Sobel method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of I is maximum.
- The Prewitt method finds edges using the Prewitt approximation to the derivative. It returns edges at those points where the gradient of I is maximum.

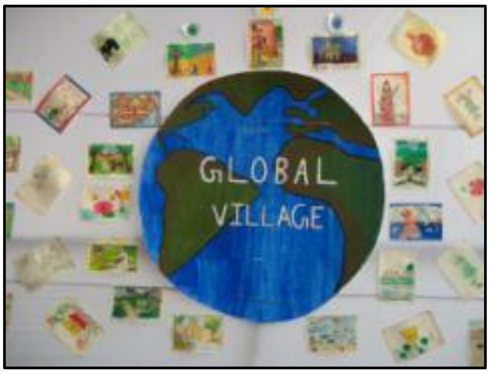


Fig 8. Actual captured image - I

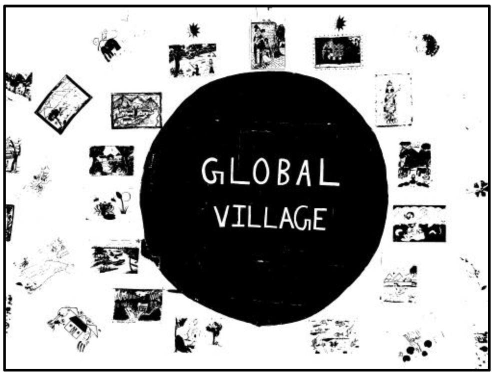


Fig 9. Separating the text from the image

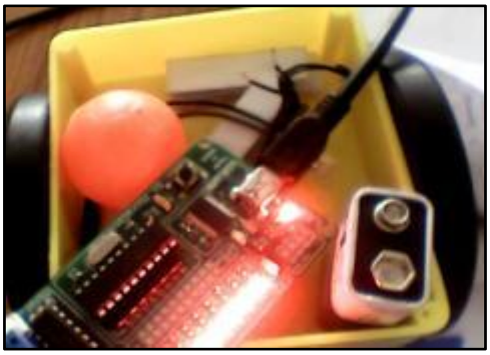


Fig 10. Actual captured image- II

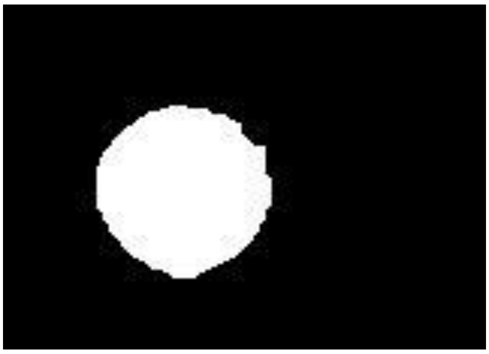


Fig 11. Separating the ball from the rest of image

To extract the useful portion from the image, the code first converts the RGB image captured into a binary image. The decision whether a pixel will be one or zero is taken as follows. The pixels of the region of interest(ROI) of the hand taken values in a specific range, say 120-150 for the Red segment of the image, similarly certain pixel values are taken by most of the pixels in Green and Blue region. The code below uses these ranges to decide whether a pixel belongs to the (ROI), a pixel in an image is part of the (ROI) only if its pixel value falls in the specified range. This is known as segmentation, as the algorithm is essentially dividing the pixel into two groups. Pixels that are part of (ROI) will have value one, while the rest will have value zero. However, it is imperative that certain pixels of (ROI) will have values outside this range. So these pixels will be ignored as being a part of the (ROI), since these pixels will have value zero. The binary image is not so smooth and has holes in it. These holes are then smoothed out using a function which is essentially either a median filter or an averaging filter in local neighbourhood.

```
%Separating the region of interest
k=y(:,:,1)<=255 & y(:,:,1)>=175 & y(:,:,2)<=129 & y(:,:,2)>=69 &
y(:,:,3)<=130 & y(:,:,3)>=23;

%applying an averaging filter
f=imfill(k, 'holes');

%smoothening the images
se=strel('disk', 5);
e=imerode(f, se);
```

6.5 Colour Image Sharpening

Sharpening an RGB colour image with a linear spatial filter follows the same procedure as that for a greyscale image, but using a sharpening filter instead. In this section we consider image sharpening using the Laplacian. From vector analysis, we know that the Laplacian of a vector is defined as a vector whose components are equal to the Laplacian of the individual scalar components of the input vector. In the RGB colour system, the Laplacian of vector c is

$$\nabla^2[c(x,y)] = \begin{bmatrix} \nabla^2 R(x,y) \\ \nabla^2 G(x,y) \\ \nabla^2 B(x,y) \end{bmatrix}$$

It tells us that we can compute the Laplacian of a full-colour image by computing the Laplacian of each component image separately.

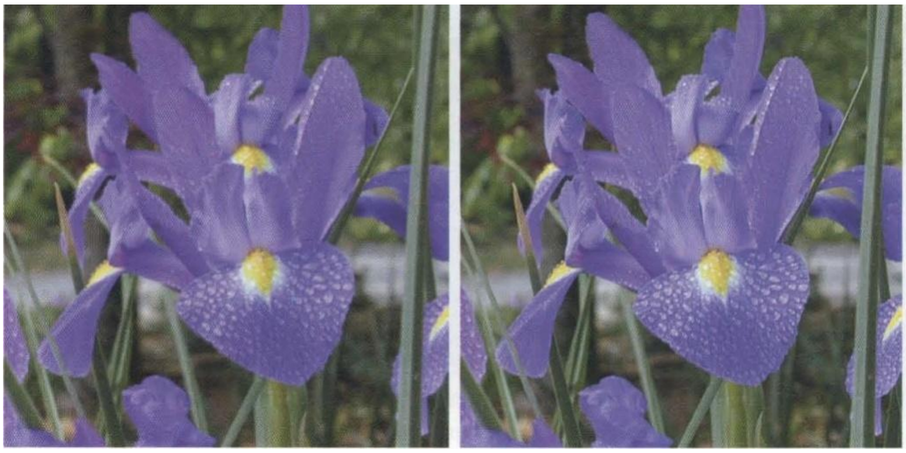


Fig 12. The original image and the processed image after applying sharpening filter.

6.6 Image Segmentation in RGB vector space

Colour region segmentation using RGB colour vectors is straightforward. Suppose that the objective is to segment objects of a specified colour range in an RGB image. Given a set of sample colour points representative of a colour (or range of colours) of interest, we obtain an estimate of the "average" or "mean" colour that we wish to segment. Let this average colour be denoted by the RGB vector m. The objective of segmentation is to classify each RGB pixel in a given image as having a colour in the specified range or not. In order to perform this comparison, it is necessary to have a measure of similarity. One of the simplest measures is the

Euclidean distance. Let z denote an arbitrary point in the 3-D RGB space. We say that z is similar to m if the distance between them is less than a specified threshold, T. The Euclidean distance between z and m is given by

$$D(\mathbf{z},\mathbf{m}) = \| \mathbf{z} - \mathbf{m} \| = \left[(\mathbf{z}-\mathbf{m})^T (\mathbf{z}-\mathbf{m}) \right]^{1/2}$$
$$= \left[(z_R - m_R)^2 + (z_G - m_G)^2 + (z_B - m_B)^2 \right]^{1/2}$$

1/2

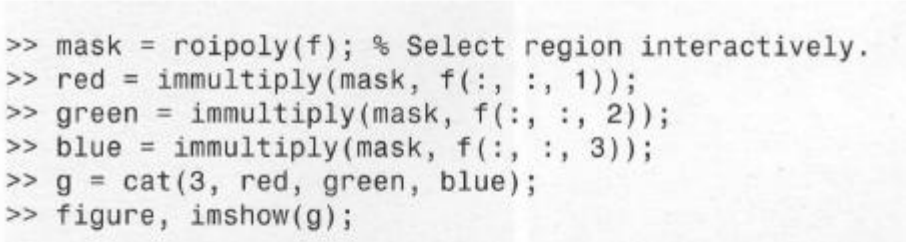
Where|| . || is the norm of the argument, and the subscripts R, G, and B, denote the RGB components of vectors z and m. The locus of points such that D (z, m) <= T is a solid sphere of radius T, as illustrated in Fig. 7.29(a). By definition, points contained within, or on the surface of, the sphere satisfy the specified colour criterion; points outside the sphere do not. Coding these two sets of points in the image with, say, black and white, produces a binary, segmented image.

Segmentation in the manner just described is implemented by custom function colorseg, which has the syntax

S = colorseg(method, f, T, parameters)

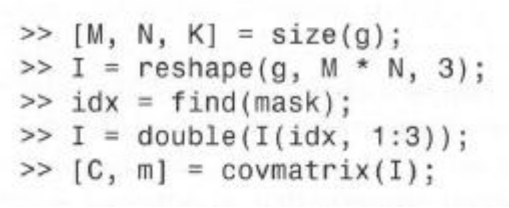
where method is either 'euclidean' or 'mahalanobis' , f is the RGB colour image to be segmented, and T is the threshold described above. The input parameters are either m if 'euclidean' is chosen, or m and C if 'mahalanobis ' is selected. Parameter m is the mean, m, and C is the covariance matrix, C. The output, S, is a two-level image (of the same size as the original) containing Os in the points failing the threshold test, and Is in the locations that passed the test. The 1s indicate the regions that were segmented from f based on colour content.

First we obtain samples representing the range of colours to be segmented. One simple way to obtain such a region of interest (ROI) is to use function roipoly, which produces a binary mask of a region selected interactively. Let, f denote the colour image.



```
>> mask = roipoly(f); % Select region interactively.
>> red = immultiply(mask, f(:, :, 1));
>> green = immultiply(mask, f(:, :, 2));
>> blue = immultiply(mask, f(:, :, 3));
>> g = cat(3, red, green, blue);
>> figure, imshow(g);
```

Here, mask is a binary image (the same size as f) generated using roipoly. Next, we compute the mean vector and covariance matrix of the points in the ROI, but first the coordinates of the points in the ROI must be extracted.



```
>> [M, N, K] = size(g);
>> I = reshape(g, M * N, 3);
>> idx = find(mask);
>> I = double(I(idx, 1:3));
>> [C, m] = covmatrix(I);
```

The final preliminary computation is to determine a value for T. A good starting point is to let T be a multiple of the standard deviation of one of the colour components. The main diagonal of e contains the variances of the RGB components, so all we have to do is extract these elements and compute their square roots. We now proceed to segment the image using values of T equal to multiples of SD, which is an approximation to the largest standard deviation.

Figures shown next illustrate the concepts just discussed.

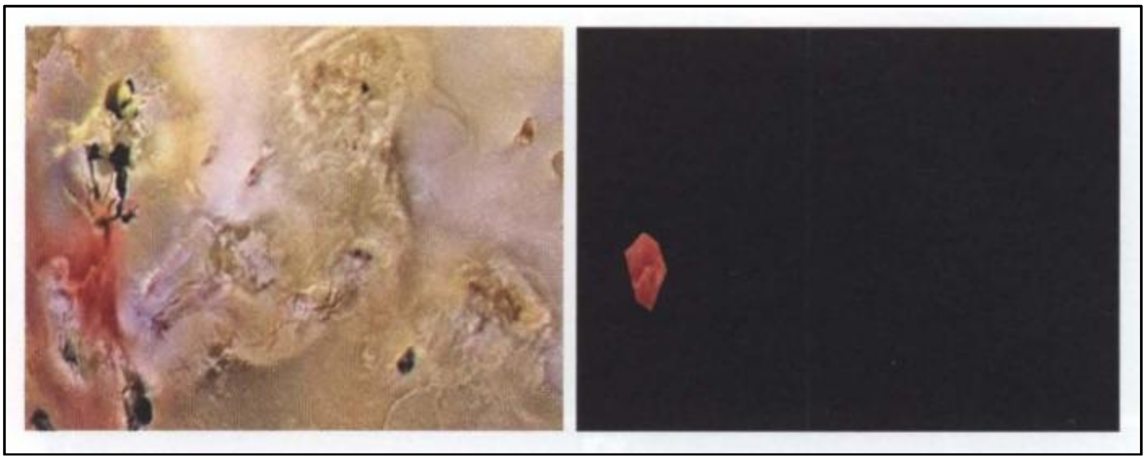


Fig 13. Pseudocolor of the surface of Jupiter's Moon Io and region of interest extracted interactively using function roipoly

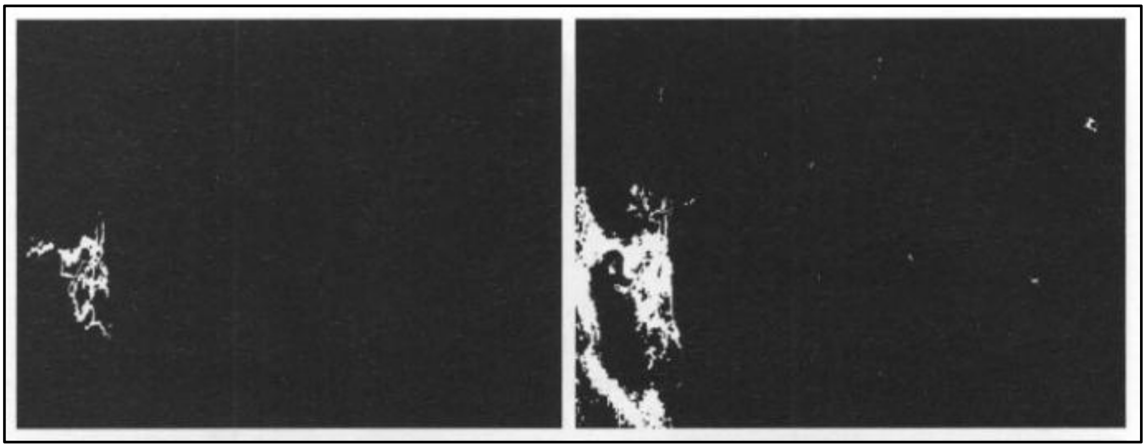


Fig 14.Segmentation result obtained using option 'Euclidean' for T=25 and T = 50

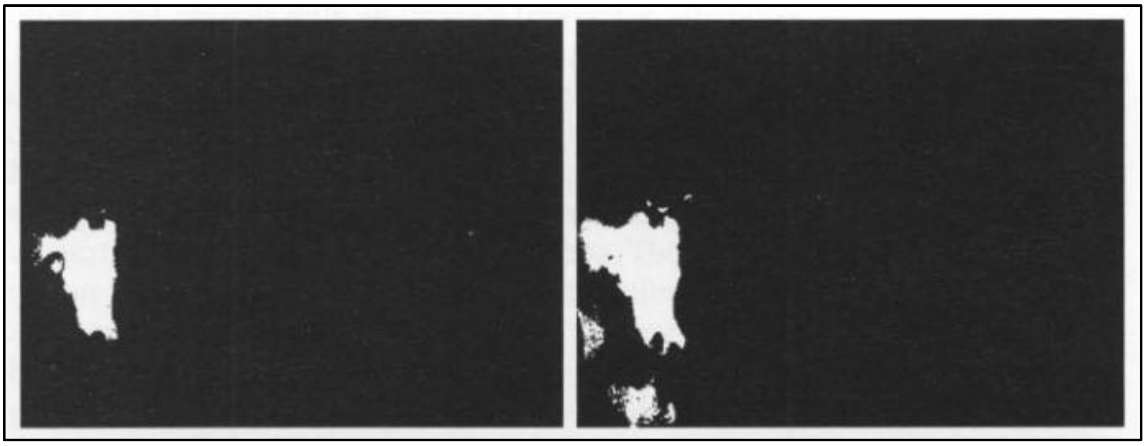


Fig 15.Segmentation result obtained using option 'Mahalanobis' for T=25 and T = 50

6.7 Dilation and Erosion

Before we undertake a discussion on Dilation and Erosion, a brief discussion of Set Theory is warranted.

The union of two sets, A and B, denoted by

$$C = A \cup B$$

is the set of all elements that belong to A, to B, or to both. Similarly, the intersection of sets A and B, denoted by

$$C = A \cap B$$

is the set of all elements that belong to both A and B.

The difference of sets A and B, denoted A - B, is the set of all elements that belong to A but not to B:

$$A - B = \{w \mid w \in A, w \notin B\}$$

Figure below illustrates the set operations defined thus far. The result of each operation is shown in gray.

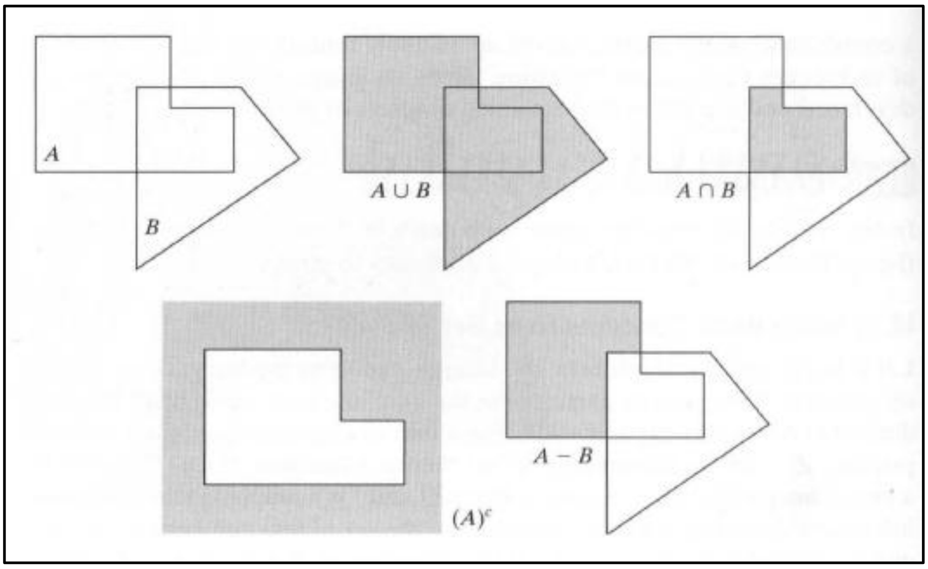


Fig 16. Basic Set Operations

Dilation

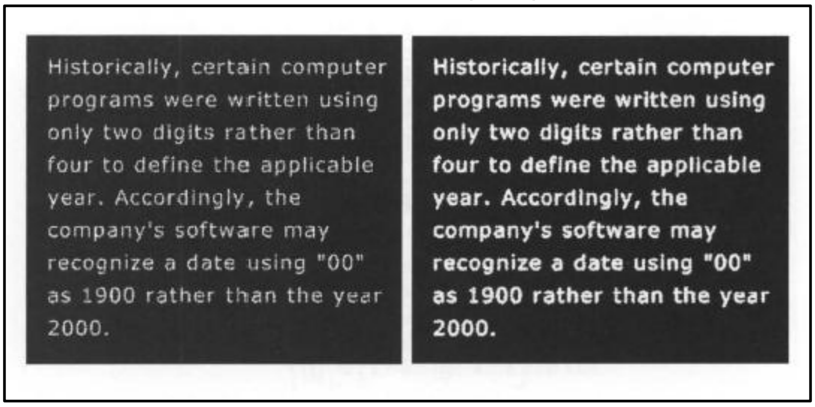
Dilation is an operation that "grows" or "thickens" objects in an image. The specific manner and extent of this thickening is controlled by a shape referred to as a structuring element. Graphically, structuring elements can be are represented either by a matrix of 0s and 1s or as a set of foreground (I-valued) pixel. Regardless of the representation, the origin of the structuring element must be clearly identified. Dilation as a process translates the origin of the structuring element throughout the domain of the image and checks to see where the element overlaps I-valued pixels. The output image is I at each location of the origin of the structuring element such that the structuring element overlaps at least one I-valued pixel in the input image. The dilation of A by B is defined as the set operation

$$A \oplus B \models \{z \mid (\hat{B})_z \cap A \neq \varnothing\}$$

where Ǿ is the empty set and B is the structuring element. In words, the dilation of A by B is the set consisting of all the structuring element origin locations where the reflected and translated B overlaps at least one element of A. It is a convention in image processing to let the first operand be the image and the second operand be the structuring element, which usually is much smaller than the image. We follow this convention from this point on. The translation of the structuring element in dilation is similar to the mechanics of spatial convolution.

Toolbox function imdilate performs dilation. Its basic calling syntax is

D = imdilate(A, B)



Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

Fig 17. Effectiveness of dilation in enhancing readability of text

Erosion

Erosion "shrinks" or "thins" objects in a binary image. As in dilation, the manner and extent of shrinking is controlled by a structuring element. Erosion is defined graphically as a process of translating the structuring element throughout the domain of the image and checking to see where it fits entirely within the foreground of the image. The output image has a value of I at each location of the origin of the structuring element , such that the element overlaps only I-valued pixels of the input image (i.e., it does not overlap any of the image background). The erosion of A by B is defined as:

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$

This equation says that the erosion of A by B is the set of all points z such that B, translated by z, is contained in A. Because the statement that B is contained in A is equivalent to B not sharing any elements with the background of A, we can write the following equivalent expression as the definition of erosion:

$$A \ominus B = \{z \mid (B)_z \cap A^c = \varnothing\}$$

Here, erosion of A by B is the set of all structuring element origin locations where no part of B overlaps the background of A.

In the figure next shown, we see an example of the effect of applying erosion on the image. Structuring elements of size 5, 10 and 20 were. The larger the disk size, more the erosion occurs.

Toolbox function imerode performs dilation. Its basic calling syntax is
D = imerode(A, B)

Where A is the input image and B is the structuring element.

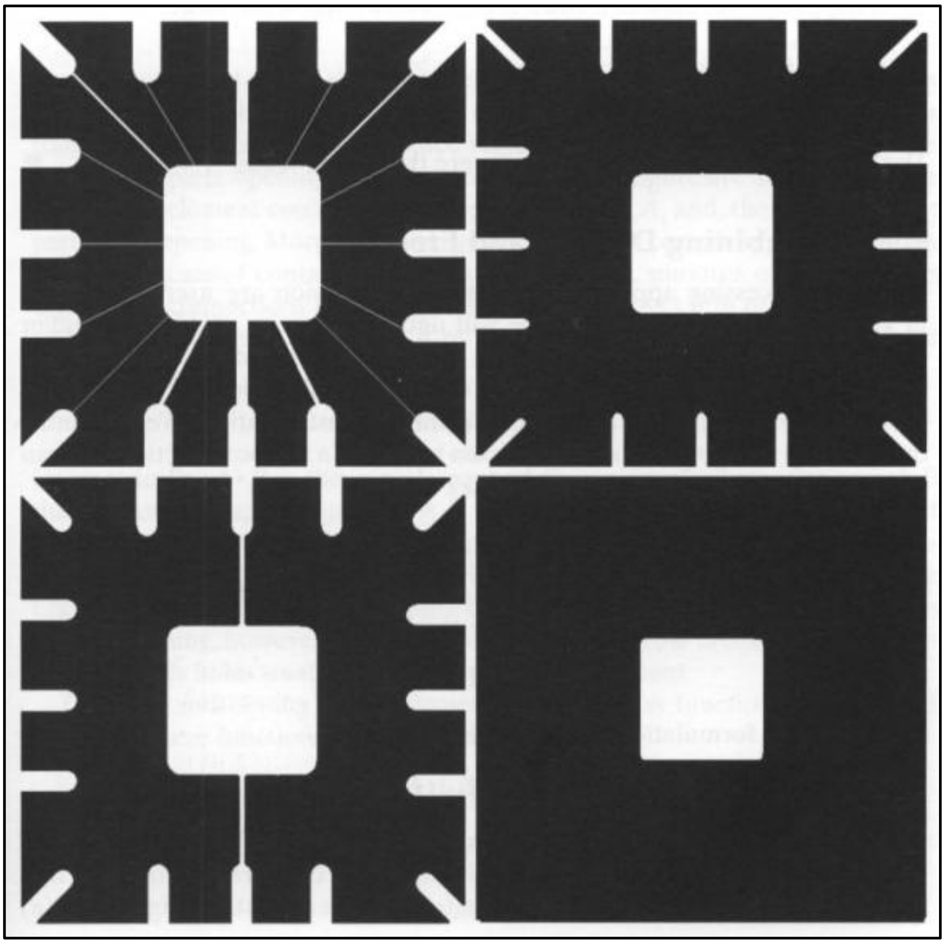


Fig 18. Erosion operation performed on a binary image

The strel Function

Toolbox function strel constructs structuring elements with a variety of shapes and sizes. Its basic syntax is

se = strel(shape, parameters)

where shape is a string specifying the desired shape, and parameters is a list of parameters that specify information about the shape, such as its size. For example, strel('diamond', 5) returns a diamond-shaped structuring element that extends ± 5 pixels along the horizontal and vertical axes. Other common parameters are disk, rectangle, square, line and octagon.

Labelling Connected Components

A pixel p at coordinates (x, y) has two horizontal and two vertical neighbours whose coordinates are (x + 1, y), (x - 1, y), (x, y + 1), and (x, y - 1). This set of neighbours of p is denoted N_4(p). The four diagonal neighbours of p have coordinates (x+1,y+1), (x+1,y-1), (x-1,y+1), and (x - 1, y -1). These neighbours are denoted N_D(P). The union of N_4(p) and N_D(P) are the 8-neighbors of P, denoted N_g(p)

.

Two pixels p and q are said to be 4-adjacent if q E N_4(p). Similarly, p and q are said to be 8-adjacent if q ε N_g(p). A path between pixels P_1 and p_n is a sequence of pixels P_1 , P_2, .. P_{N-1},P_n such that P_k is adjacent to P_{k+1} for 1 <= k < n. A path can be 4-connected or 8-connected, depending on the type of adjacency used.

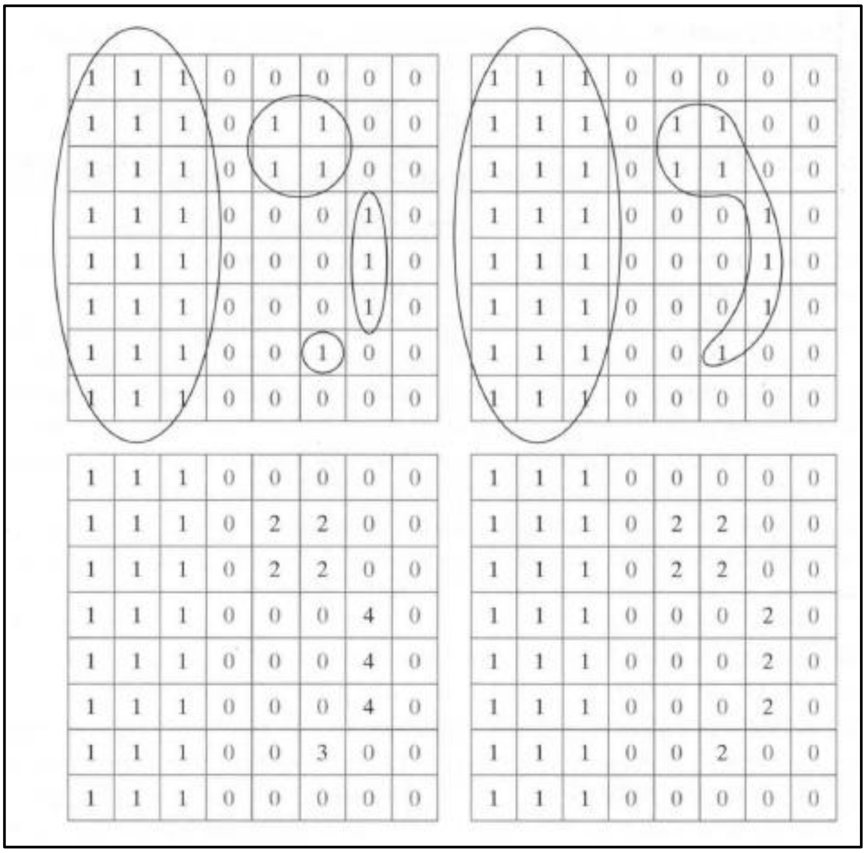
Two foreground pixels P and q are said to be 4-connected if there exists a 4-connected path between them, consisting entirely of foreground pixels. They are 8-connected if there exists an 8-connected path between them. For any foreground pixel, p, the set of all foreground pixels connected to it is called the connected component containing p.

A connected component was just defined in terms of a path, and the definition of a path in turn depends on the type of adjacency used. This implies that the nature of a connected component depends on which form of adjacency we choose, with 4- and 8-adjacency being the most common. Figure shown next illustrates the effect that adjacency can have on determining the number of connected components in an image.

Toolbox function bwlabel computes all the connected components in a binary image. The calling syntax is:

[L, numl = bwlabel(f, conn)

where f is an input binary image and conn specifies the desired connectivity (either 4 or 8). Output L is called a label matrix, and num (optional) gives the total number of connected components found. If parameter conn is omitted, its value defaults to 8.



1	1	1	0	0	0	0	0
1	1	1	0	1	1	0	0
1	1	1	0	1	1	0	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	1	0	0
1	1	1	0	0	0	0	0

1	1	1	0	0	0	0	0
1	1	1	0	1	1	0	0
1	1	1	0	1	1	0	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	1	0	0
1	1	1	0	0	0	0	0

1	1	1	0	0	0	0	0
1	1	1	0	2	2	0	0
1	1	1	0	2	2	0	0
1	1	1	0	0	0	4	0
1	1	1	0	0	0	4	0
1	1	1	0	0	0	4	0
1	1	1	0	0	3	0	0
1	1	1	0	0	0	0	0

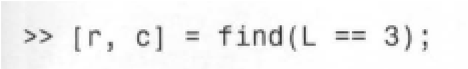
1	1	1	0	0	0	0	0
1	1	1	0	2	2	0	0
1	1	1	0	2	2	0	0
1	1	1	0	0	0	2	0
1	1	1	0	0	0	2	0
1	1	1	0	0	0	2	0
1	1	1	0	0	2	0	0
1	1	1	0	0	0	0	0

Fig 19. 4-connected and 8-connected adjacency

Next, we show how to compute and display the centre of mass of each connected component. First, we use bwlabel to compute the 8-connected components:

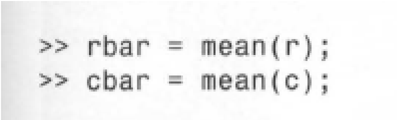
```
» f = imread('objects.tif');
» [L, n] = bwlabel(f);
```

Function ‘find’ is useful when working with label matrices. For example, the following call to find returns the row and column indices for all the pixels belonging to the third object:



```
>> [r, c] = find(L == 3);
```

Function ‘mean’ with r and c as inputs then computes the centre of mass of this object.



```
>> rbar = mean(r);
>> cbar = mean(c);
```

6.8 Contrast Stretching and Brightness change

$$g(x,y) = \frac{f(x,y)-fmin}{fmax-fmin} * 2^{bpp}$$

For increasing brightness:

%loop over the pixel data and add a value
p[i] = p[i]+100 < 255 ? p[i]+100 : 255;
p[i+1] = p[i+1]+100 < 255 ? p[i+1]+100 : 255;
p[i+2] = p[i+2]+100 < 255 ? p[i+2]+100 : 255;

And for reducing brightness:

%loop over the pixel data and add a value
p[i] = p[i]-100 >= 0 ? p[i]-100 : 0;
p[i+1] = p[i+1]-100 >= 0 ? p[i+1]-100 : 0;
p[i+2] = p[i+2]+100 >= 0 ? p[i+2]-100 : 0;

6.9 Displaying Histogram for Colour Images in MATLAB

imhist displays a histogram for the image if it is a greyscale or binary. Use rgb2gray on the image, or use imhist(input(:,:,1)) to see one of the channel at a time (red in this example).

Alternatively, the following option is available:

hist(reshape(input,[],3),1:max(input(:))); colormap([1 0 0; 0 1 0; 0 0 1]);
to show the 3 channels simultaneously

B = reshape(A,m,n) or B = reshape(A,[m n]) returns the m-by-n matrix B whose elements are taken column-wise from A. An error results if A does not have m*n elements.

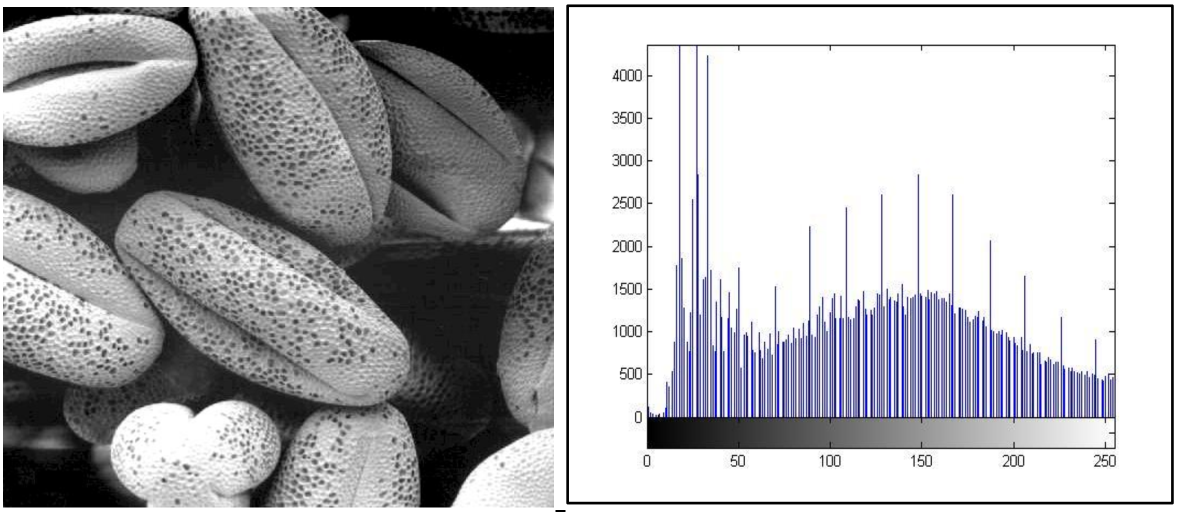


Fig 20. A grayscale image and its histogram

6.10 Linear spatial filtering, correlation and convolution

The concept of linear filtering has its roots in the use of the Fourier transform for signal processing in the frequency domain. Here, we are interested in filtering operations that are performed directly on the pixels of an image. Use of the term linear spatial filtering differentiates this type of process from frequency domain filtering. The linear operations of interest consist of multiplying each pixel in the neighborhood by a corresponding coefficient and summing the results to obtain the response at each point (x , y). If the neighborhood is of size m X n, mn coefficients are required. The coefficients are arranged as a matrix, called a filter, mask, filter mask, kernel, template, or window, with the first three terms being the most prevalent. For reasons that will become obvious shortly, the terms convolution filter, convolution mask, or convolution kernel, also are used.

Figure illustrates the mechanics of linear spatial filtering. The process consists of moving the centre of the filter mask, w, from point to point in an image, f. At each point (x, y), the response of the filter at that point is the sum of products of the filter coefficients and the corresponding neighbourhood pixels in the area spanned by the filter mask. For a mask of size m X n we assume typically that m = 2a + 1 and n = 2b + 1 where a and b are nonnegative integers. All this says is that our principal focus is on masks of odd sizes, with the smallest meaningful size being 3 X 3. Although it certainly is not a requirement, working with odd-size masks is more intuitive because they have an unambiguous center point. There are two closely related concepts that must be understood clearly when performing linear spatial filtering. One is correlation; the other is convolution.

Correlation is the process of passing the mask W by the image array f in the manner described in Figure. Mechanically, convolution is the same process, except that W is rotated by 180 degrees prior to passing it by f. As in the one-dimensional example discussed earlier, convolution yields the same result independently of the order of the functions. In correlation the order does matter, a fact that is made clear in the toolbox by assuming that the filter mask is always the function that undergoes translation. Note also, the important fact that the results of spatial correlation and convolution are rotated by 180 degrees with respect to each other. This, of course, is expected because convolution is nothing more than correlation with a rotated filter mask. Summarizing the preceding discussion in equation form, we have that the correlation of a filter mask w(x,y) of size m X n with a function f(x , y), is given by

$$w(x, y) \star f(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)$$

This equation is evaluated for all values of the displacement variables x and y so that all elements of w visit every pixel in f, which we assume has been padded appropriately. Constants a and b are given by

For notational convenience, we assume that m and n are odd integers. In a similar manner , the convolution of w(x, y) and f(x, y), denoted by

$$w(x,y)\star f(x,y)=\sum_{s=-a}^{a}\sum_{t=-b}^{b}w(s,t)f(x-s,y-t)$$

where the minus signs on the right of the equation flip f(i.e., rotate it by 180°). Rotating and shifting instead of w is done to simplify the notation. The result is the same. t The terms in the summation are the same as for correlation. The toolbox implements linear spatial filtering using function imfilter, which has the following syntax:

g = imfilter(f, w, filtering_mode, boundary_options, size_options)

where f is the input image, W is the filter mask, g is the filtered result. The filtering_mode is specified as 'corr' for correlation (this is the default) or as 'conv' for convolution. The boundary_options deal with the border - padding issue, with the size of the border being determined by the size of the filter. The size_options are either 'same' or 'full' . The most common syntax for imfilter is:

g = imfilter(f, w, 'replicate')

Next, we discuss a few popular filter masks used for image restoration:

Mean Filtering

Mean filtering is a simple, intuitive and easy to implement method of smoothing images, i.e. reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images. The idea of mean filtering is simply to replace each pixel value in an image with the mean (`average') value of its neighbours, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighbourhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, as shown in Figure 1, although larger kernels (*e.g.* 5×5 squares) can be used for more severe smoothing. (Note that a small kernel can be applied more than once in order to produce a similar but not identical effect as a single pass with a large kernel.) A 3x3 mean filter may look like as shown below:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Median Filtering

Median filtering is a nonlinear process useful in reducing impulsive, or salt-and-pepper noise. It is also useful in preserving edges in an image while reducing random noise. Impulsive or salt-and pepper noise can occur due to a random bit error in a communication channel. In a median filter, a window slides along the image, and the median intensity value of the pixels within the window becomes the output intensity of the pixel being processed. For example, suppose the pixel values within a window are 5, 6, 55, 10 and 15, and the pixel being processed has a value of 55. The output of the median filter is the current pixel location is 10, which is the median of the five values. Like low pass filtering, median filtering smoothes the image and is thus useful in reducing noise. Unlike low pass filtering, median filtering can preserve discontinuities in a step function and can smooth a few pixels whose values differ significantly from their surroundings without affecting the other pixels. Median filtering is not the only type of Non-Linear filtering. Other filtering techniques include the 0th percentile filtering also called the minimum filtering and 100th percentile filtering or the maximum filtering.

CHAPTER-7
DEVELOPMENT OF THE APPLICATION INTERFACE

The processed input is meant to generate some useful information that will be used to understand what the user intends to do. Based on the processed input, the requisite action will be taken by the software/application. Here, we will enlist a few applications that will be controlled using gesture inputs. The intention of this chapter is to look at the methods/functions/code-blocks in the program that actually perform the function. In other words, the master software will call these methods/functions/code-blocks to perform the operation/task based on user input. Here onwards, the terms methods, functions, library and code-blocks will be used synonymously and will denote a particular function to be executed. Similar holds true for operation and task.

Here, we will look at four different applications to be controlled using image processing:

1. Control of room light
2. Control of Windows Media Player
3. Control of PowerPoint presentation
4. Control of a basic image processing Graphical User Interface

Now, one by one we take a look at the development each of these applications:

Application 1: Control of room light

Application 1 will be controlled via the Arduino Development board that will form the interface between MATLAB and the light bulb. A brief introduction of the Arduino board and its features has been given in a preceding chapter and the appendices. As per the instruction received by Arduino from MATLAB to either light the bulb or turn it off, the Arduino board will excite/de-excite a relay that will control the light bulb. We will use a standard incandescent light bulb, although nothing restricts us from using the same circuit to control any of the electrical appliances to turn them ON/OFF.

Application 2: Control of Windows Media Player

Control of Windows Media Player uses two important technologies – The MATLAB's GUIDE that is used to develop Graphical User Interface in MATLAB and Microsoft's ActiveX control. Combining these two powerful technologies, we can create a Windows Media Player inside MATLAB that can be controlled using hand gestures. We first take a basic look at MATLAB's GUIDE.

Any GUI in MATLAB is in essence a collection of objects. Every object in the GUI has a unique handle (name). For instance, let's consider the following GUI. It is made up of three distinct objects, which are the following. The frame or (figure) which is labelled 'Untitled'. The second object is the edit box. And the third object is the 'Button' which is labelled 'Push Button'. Every object must have a unique handle. There are a number of ways the user can obtain the handle of an object in MATLAB. In MATLAB 7 it is almost too easy. In previous versions of MATLAB it's just a little bit more work. First we see how MATLAB 7 manages handles. MATLAB 7 has a function, which collects every handle in the GUI and places it in a convenient data structure. This makes life very easy as the user does not have to poll every object for it's handle.

MATLAB has two very important functions, which allow the user to alter an object's properties at their discretion. These functions are listed below.

GET Get object properties.
GET(H,'PropertyName') returns the value of the specified property for the graphics object with handle H.

SET Set object properties.
SET(H,'PropertyName',PropertyValue) sets the value of the specified property for the graphics object with handle H.

MATLAB is able to automatically generate a lot of code that is needed for GUI. When working under time constraints this feature of MATLAB comes in very handy. For beginners, it's ok to rely on the automatically generated code. MATLAB has a program called 'guide' which allows users to setup their GUI. The guide tool is very intuitive to use and can be accessed from the command line by simply typing 'guide' and pressing *Enter*.

Here, we developed a basic GUI using various input types available in GUIDE that has an Axis, a few buttons, and text area.

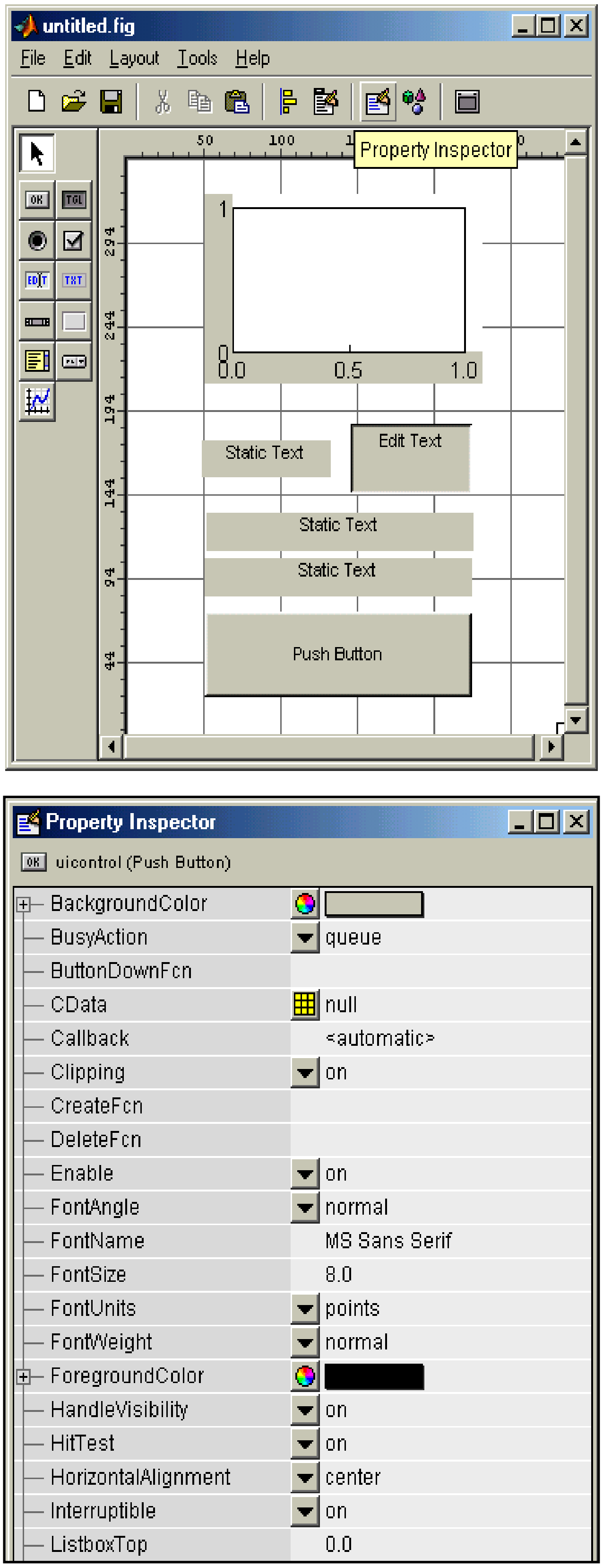
As soon as we save it(let the name be mygui), MATLAB should generate the skeleton source code and the source code should automatically open in an editor.

The m-file associated with the GUI contains all the skeletal code to get the GUI up and running, but we need to add our own code to do something useful. Example, in the event_listener for Push Button, we may add a function that plots the cosine function on the axis. An event_listener for an object calls the function associated with that object when we activate that object by clicking on it.

To run the program or the GUI, we simply go to the MATLAB main window and call the program by its name.

```
>> mygui
```

Doing so fires up the GUI we created, and is ready to respond to us.



BackgroundColor	□
BusyAction	queue
ButtonDownFcn	
CData	null
Callback	<automatic>
Clipping	on
CreateFcn	
DeleteFcn	
Enable	on
FontAngle	normal
FontName	MS Sans Serif
FontSize	8.0
FontUnits	points
FontWeight	normal
ForegroundColor	■
HandleVisibility	on
HitTest	on
HorizontalAlignment	center
Interruptible	on
ListboxTop	0.0

Fig 21. MATLAB's GUIDE tool to create GUI applications

Microsoft's ActiveX Controls

MATLAB has capabilities to host an ActiveX control. MATLAB has many useful features that developers can benefit from. Here is an introduction on steps to use an ActiveX control within MATLAB environment.

With the following MATLAB commands, we can host any ActiveX control in desired figures and set/get any property, invoke any method and handle events.

In the first step, we create an ActiveX control in a figure window. actxcontrol takes PROGID of ActiveX, position and figure handle of a window to create control on figure. The syntax is as below:

```
h = actxcontrol (progid [, position [, fig_handle [, callback |
{event1 eventhandler1; event2 eventhandler2; ...} [, filename]]]])
```

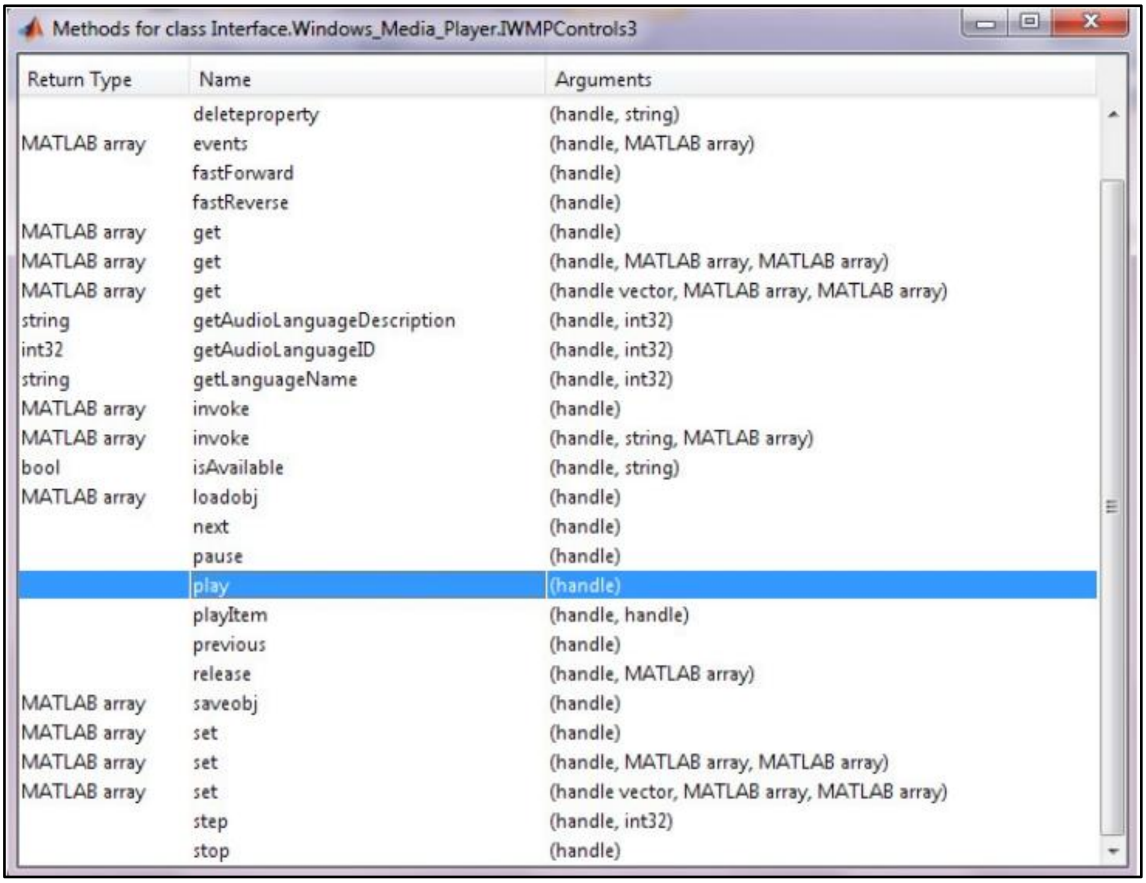
progid is a string that is the name of the control to create. The control vendor provides this string. position is position vector containing the x and y location and the width and height of the control, expressed in pixel units as [x y width height]. Defaults to [20 20 60 60]. fig_handle
is the handle of the figure window in which the control is to be created. callback is name of an M-function that accepts a variable number of arguments. event is specified by either number or name. eventhandler is name of a M-function that accepts a variable number of arguments. This function will be called whenever the control triggers the event associated with it. filename is the name of a file to which a previously created control has been saved. When you specify filename, MATLAB creates a new control using the position, handle and event/eventhandler arguments, and then initializes the control from the specified file. The progid argument in actxcontrol must match the PROGID of the saved control.
For example, for creating Media Player ActiveX in a figure, we must call actxcontrol as follows:

```
m=figure;
h=actxcontrol('MediaPlayer.MediaPlayer.1', [0 0 300 300], m);

v = invoke(h, ['methodname' [, arg1, arg2, ...]])
```

methodname is a string that is the name of the method to be invoked. arg1 , ..., argn are arguments, if any, required by the method being invoked. For example, for playing a movie in Media Player, we should do:

invoke(h, 'play')

If you don't remember the method name, just call methods or methodview commands. For example, methods(h) shows all of the methods that an ActiveX control supports. methodsview shows methods and their arguments. Figure below lists some of the controls supported by ActiveX for Windows Media Player



Methods for class Interface.Windows_Media_Player.IWMPControls3

Return Type	Name	Arguments
	deleteproperty	(handle, string)
MATLAB array	events	(handle, MATLAB array)
	fastForward	(handle)
	fastReverse	(handle)
MATLAB array	get	(handle)
MATLAB array	get	(handle, MATLAB array, MATLAB array)
MATLAB array	get	(handle vector, MATLAB array, MATLAB array)
string	getAudioLanguageDescription	(handle, int32)
int32	getAudioLanguageID	(handle, int32)
string	getLanguageName	(handle, int32)
MATLAB array	invoke	(handle)
MATLAB array	invoke	(handle, string, MATLAB array)
bool	isAvailable	(handle, string)
MATLAB array	loadobj	(handle)
	next	(handle)
	pause	(handle)
	play	(handle)
	playItem	(handle, handle)
	previous	(handle)
	release	(handle, MATLAB array)
MATLAB array	saveobj	(handle)
MATLAB array	set	(handle)
MATLAB array	set	(handle, MATLAB array, MATLAB array)
MATLAB array	set	(handle vector, MATLAB array, MATLAB array)
	step	(handle, int32)
	stop	(handle)

Fig 22. Viewing the methods that can be called on a handle

We once again show a slightly different technique to perform the same ActiveX controls as we just discussed.

fig = figure('Name', 'Windows Media Player', 'NumberTitle', 'off',...
'Menubar', 'none');
wmpM = actxcontrol ('WMPlayer.OCX.7', [0 0.4004 million], fig);

Then we use the object handles wmpM as its object.

Establish the location of the audio file
Syntax:
wmpM.url = [pathname filename]

For example, the action triggered by that button presses to play the audio file
Syntax:
wmpM.controls.play

For example, want to take action to Pause
Syntax:
wmpM.controls.pause

For example, want to take action to stop
Syntax:
wmpM.controls.stop

Thus, to control the Windows Media Player using gestures, we will need to assign a unique gesture to each of the controls such as stop, play, play, fast forward and fast reverse. Based on the unique gesture registered by MATLAB, the appropriate ActiveX control will be called.

In this Media Player, we will perform a few basic operations using hand gestures, namely – Play, Pause, Fast Forward, Fast Reverse, Stop and Volume Control.

The interface that was developed using MATLAB GUIDE and ActiveX is shown next.

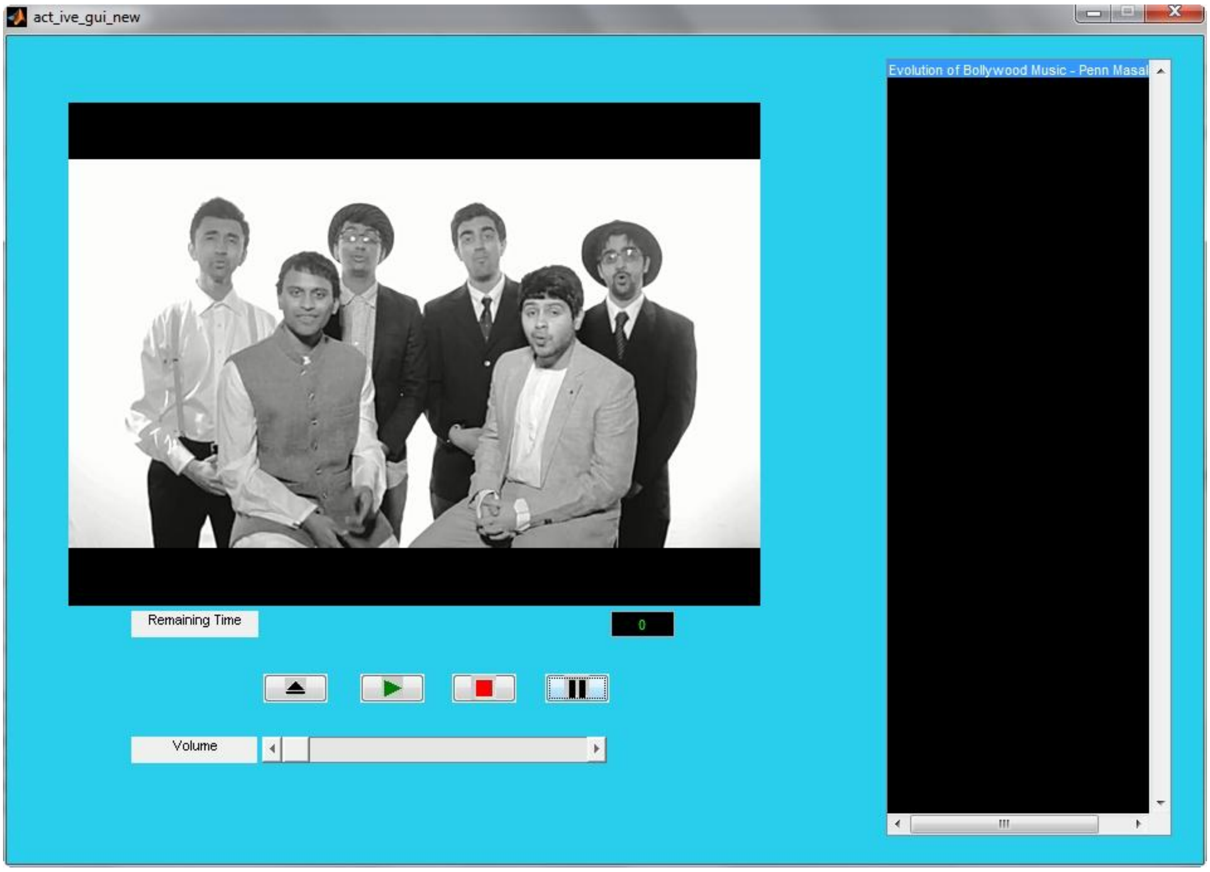


Fig 23. Windows Media Player created in GUIDE using ActiveX

Application 3: Control of PowerPoint Presentation

Controlling PowerPoint Presentation from MATLAB also uses ActiveX, but we won't need to create any GUI for this. We just need to use the methods for PowerPoint Presentation in the ActiveX control. For this we create an ActiveX Server in MATLAB using the function actxserver that creates an object handle to control the PowerPoint presentation. Next we see some the common ActiveX controls for PowerPoint.

The ActiveX controls described below are meant for MS-PowerPoint 2007. Other versions of MS-PowerPoint may use slightly different control. The corresponding documentation and object model for the same can be found on Microsoft website. We can create a new presentation in Powerpoint 2007 and add slides to it using MATLAB with the following code (note the commented sections for modifying an existing presentation):

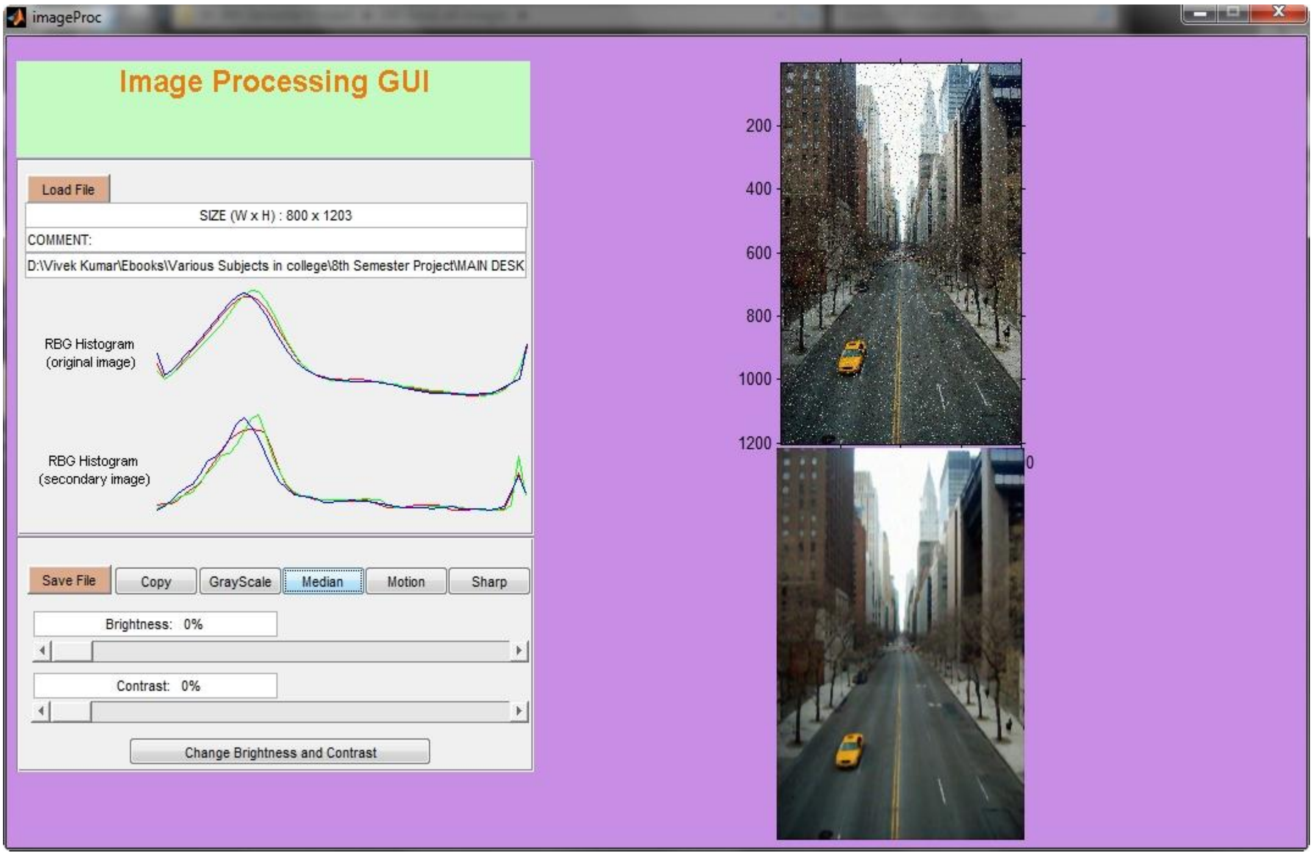
```
% Create an ActiveX object

ppt = actxserver('powerpoint.application');

ppt.Visible = 1;


% Open an existing presentation by supplying the fullpath
to the file

%ppt.Presentations.Open('C:\MyFolder\test.ppt');


% Create a new presentation

ppt.Presentations.Add()


% Add a blank slide to the presentation

layout=
ppt.ActivePresentation.SlideMaster.CustomLayouts.Item(1);

ppt.ActivePresentation.Slides.AddSlide(1, layout);


% Add a custom slide and add an image to that slide

Layout=
ppt.ActiveWindow.Selection.SlideRange(1).CustomLayout;

slides = ppt.ActivePresentation.Slides;

newSlide = slides.AddSlide(1,layout);


%Add the image using the fullpath of the image file

pic = ...

ppt.ActiveWindow.Selection.SlideRange(1).Shapes.AddPicture(
...

fullfile(matlabroot,'toolbox','images','imdemos','football.
jpg'), 'msoFalse','msoTrue',100,20,500,500);


% To add another image

pic2 =...

ppt.ActiveWindow.Selection.SlideRange(1).Shapes.AddPicture(
fullfile(matlabroot,'toolbox','images','imdemos','football.
jpg'), 'msoFalse','msoTrue',200,220,500,500);
```

```
% If you opened an already existing presentation, save it
using
%ppt.ActivePresentation.Save;


% Else save a newly created presentation at a particular
location
ppt.ActivePresentation.SaveAs(fullfile(pwd,'test.ppt'));


% Close Powerpoint and delete the object
ppt.ActivePresentation.Close;
ppt.Quit;
ppt.delete;
```

In this gesture controlled PowerPoint Presentation, we will use hand gestures to control the slide show, such as move to next slide or previous slide, save the presentation, close the presentation, add a new slide, and delete a slide.

Application 4: GUI implementing simple Image Processing Techniques

There are many tools such as Picasa, Adobe Photoshop, Irfan Editor, Pixlr and many other such tools in the market that can enhance the quality of images. But, these tools with their hundreds of functionality will many a times have a complicated interface. Navigating through the menu bars to find the appropriate option can be tiresome. Therefore we try to develop a simple image processing GUI that enhances and restores images, but the input is taken from user by means of gesture. For example, the user may slide his hand to increase brightness or make a pattern with hands to remove noise from the image. Thus, this greatly increases the ease and comfort for the user. Below is a snapshot of the image processing GUI.



SIZE (W x H) : 800 x 1203

COMMENT:

D:\Vivek Kumar\Ebooks\Various Subjects in college\8th Semester Project\MAIN DESK

Brightness: 0%

Contrast: 0%

Fig 24. Image Processing GUI created in GUIDE

As can be clearly seen from the Image Processing GUI, we have implemented some basic tasks such as converting to GrayScale, Median Filtering, Motion Filtering, Sharpening of image, brightness and contrast control. The filtered image can also be saved using the Save button. The filtered image and original image are shown on the right of the GUI.

CHAPTER-8
RESULTS

The applications developed were tested for reliable operations using following parameters:

1) Samples were taken under three different light conditions – low light conditions, natural light conditions and artificial light conditions. Natural light means the models were tested in sunlight and a standard fluorescent tube of output about 2700 lumens in a 7' by 7' closed room was used for artificial lighting.

2) In all cases the camera was kept at a distance ranging from 30cm to 50cm.

3) Three users tested the models against different light conditions.

4) The number of samples taken was successively increased from 20 to 100 in steps of 20. When it is said that 20 samples were taken then it means that in once the image processing application was started, 20 gestures were made. Next time when again the application was started, the number of gestures tested was increased to 40.

5) The Camera models used are –

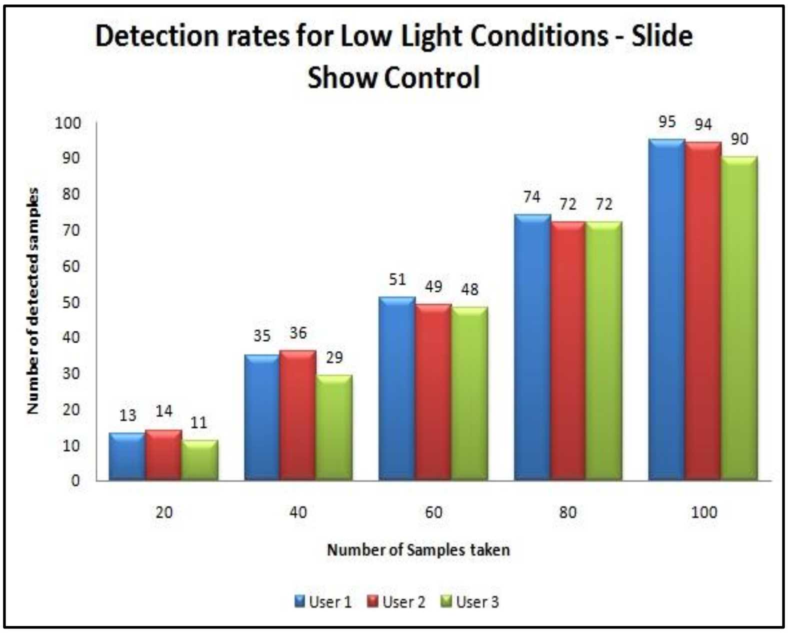
   (i) in-built 3.1 MP camera in Samsung GT-S6102 with Android Operating System.

   (ii) Beetel Eye Webcam.

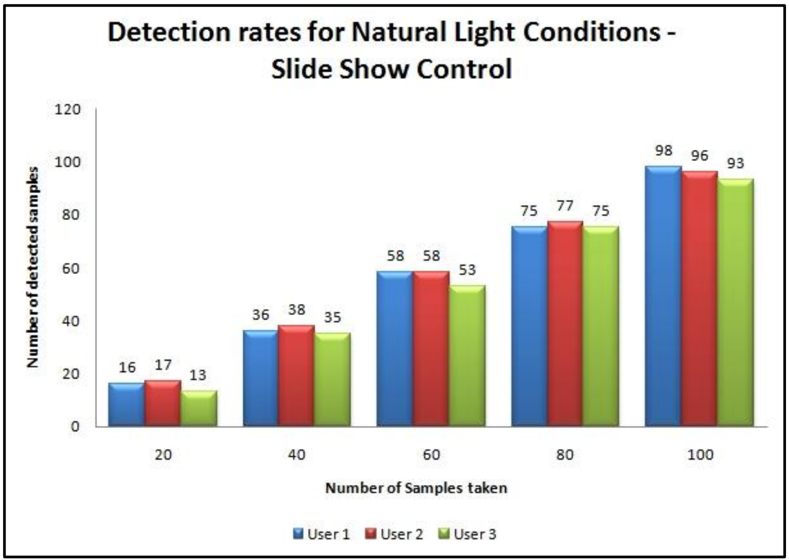
In both cases, the images were recorded at a resolution of 320x240.

6) The tests were conducted using colour bands worn on fingers and without using colour bands worn on fingers.

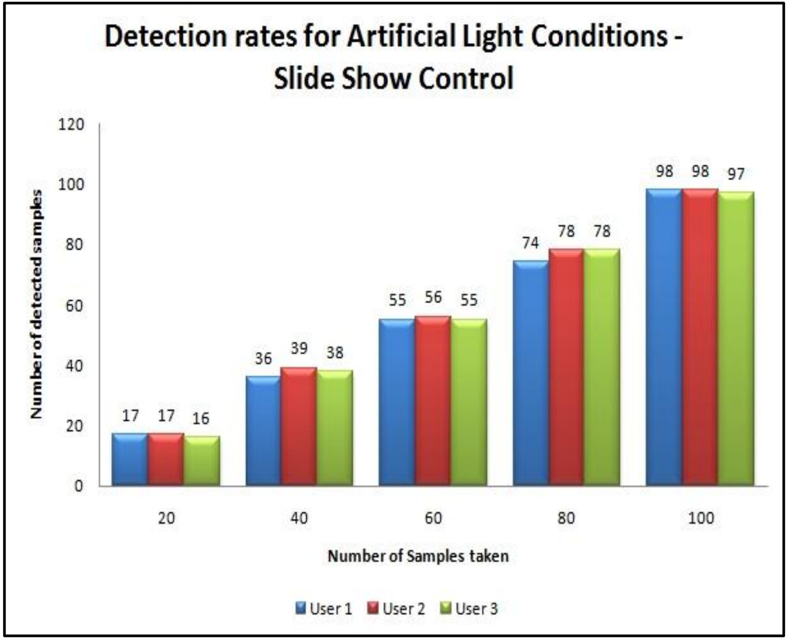
The results are shown on the next two pages.



Detection rates for Low Light Conditions - Slide Show Control



Detection rates for Natural Light Conditions - Slide Show Control



Detection rates for Artificial Light Conditions - Slide Show Control

Detection Rates Obtained with Colour Bands

Figure 25 (a), (b) and (c) show the detection rates obtained under different light conditions for the various gesture recognition applications taken together.

The results are shown for low light conditions, natural light conditions and artificial light conditions.

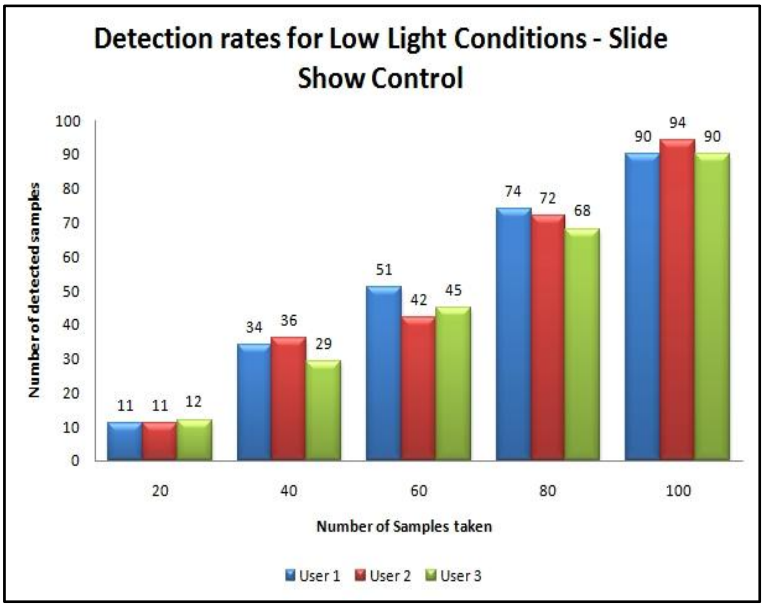
The number of samples tested was increased successively from 20 to 100.

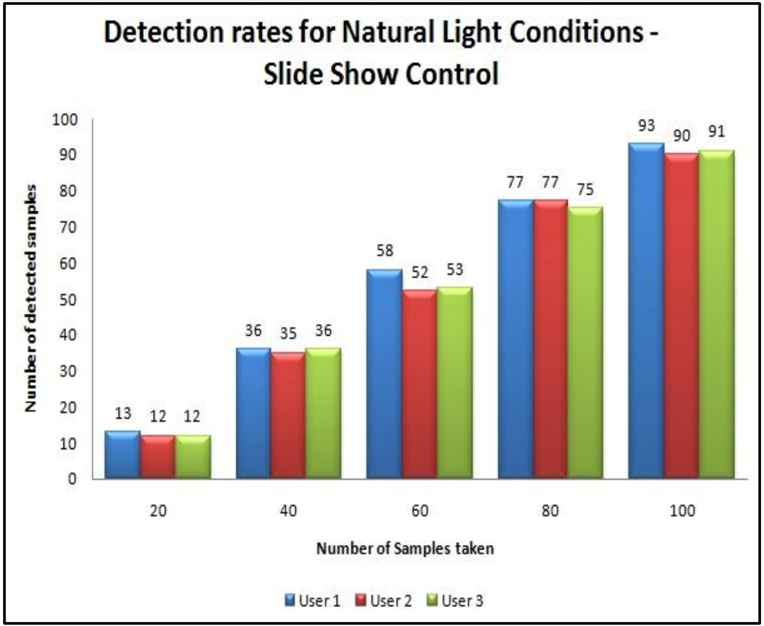
Detection Rates Obtained without Colour Bands

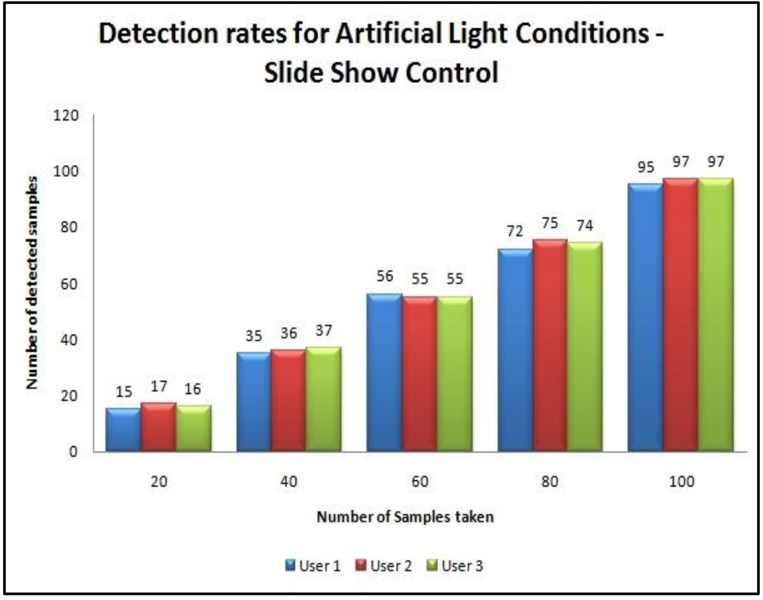
Figure 26 (a), (b) and (c) show the detection rates obtained under different light conditions for the various gesture recognition applications taken together.

The results are shown for low light conditions, natural light conditions and artificial light conditions.

The number of samples tested was increased successively from 20 to 100.







CHAPTER-9
SOURCE CODES AND ALGORITHMS

MATLAB PROGRAM FOR IMAGE PROCESSING GUI

```
function varargout = imageProc(varargin)

% IMAGEPROC M-file for imageProc.fig
%      IMAGEPROC, by itself, creates a new IMAGEPROC or raises the
existing
%      singleton*.
%
%      H = IMAGEPROC returns the handle to a new IMAGEPROC or the
handle to
%      the existing singleton*.
%
%      IMAGEPROC('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in IMAGEPROC.M with the given input
arguments.
%
%      IMAGEPROC('Property','Value',...) creates a new IMAGEPROC or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before imageProc_OpeningFunction gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to imageProc_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help imageProc

% Last Modified by GUIDE v2.5 14-Apr-2014 17:14:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @imageProc_OpeningFcn, ...
                   'gui_OutputFcn',  @imageProc_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before imageProc is made visible.
function imageProc_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to imageProc (see VARARGIN)

% Choose default command line output for imageProc

clc;

handles.fileLoaded = 0;
handles.fileLoaded2 = 0;
set(handles.axes1,'Visible','off');
set(handles.axes2,'Visible','off');
set(handles.axesHist1,'Visible','off');
set(handles.axesHist2,'Visible','off');
set(handles.editPath, 'Visible', 'off');
set(handles.editSize, 'Visible', 'off');
set(handles.editComment, 'Visible', 'off');
set(handles.textHist1, 'Visible', 'off');
set(handles.textHist2, 'Visible', 'off');
set(handles.sliderBright, 'Enable', 'off');
set(handles.sliderContrast, 'Enable', 'off');
set(handles.editBright,'String', sprintf('%10s:%4.0f%%',
'Brightness', 100*get(handles.sliderBright,'Value')));
set(handles.editContrast,'String', sprintf('%10s:%4.0f%%',
'Contrast', 100*get(handles.sliderContrast,'Value')));
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = imageProc_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in LoadButton.
function LoadButton_Callback(hObject, eventdata, handles)
% hObject    handle to LoadButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[FileName,PathName] = uigetfile({'*.*'},'Load Image File');

if (FileName==0) % cancel pressed
    return;
end

handles.fullPath = [PathName FileName];
[a, b, Ext] = fileparts(FileName); %returns the path, filename,
extension
availableExt = {'.bmp','.jpg','.jpeg','.tiff','.png','.gif'};
%supported image formats in MATLAB
FOUND = 0;
for i=1:length(availableExt)
    if (strcmpi(Ext, availableExt{i}))
        FOUND=1;
        break;
    end
end

if (FOUND==0)
    h = msgbox('File type not supported!','Error','error');
    return;
end

set(handles.editPath, 'Visible', 'on');
set(handles.editSize, 'Visible', 'on');
set(handles.editComment, 'Visible', 'on');

info = imfinfo(handles.fullPath);
if (~isempty(info.Comment))
    % save current image comment (to be used later in image save)
    handles.currentImageComment = info.Comment{1};
else
```

```
    handles.currentImageComment = '';
end

set(handles.editSize, 'String', sprintf('SIZE (W x H) : %d x %d',
info.Width, info.Height));
set(handles.editComment, 'String', sprintf('COMMENT: %s',
handles.currentImageComment));
set(handles.editPath, 'String', handles.fullPath);
set(handles.sliderBright, 'Enable', 'on');
set(handles.sliderContrast, 'Enable', 'on');

RGB = imread(handles.fullPath);

handles.RGB = RGB;
handles.RGB2 = RGB;
handles.RGBtmp = RGB;
handles.fileLoaded = 1;
handles.fileLoaded2 = 0;

set(handles.axes1,'Visible','off');
set(handles.axes2,'Visible','off');
set(handles.axesHist1,'Visible','off');
set(handles.axesHist2,'Visible','off');
set(handles.textHist1, 'Visible', 'off');
axes(handles.axesHist2); cla; %cla clears current axis
set(handles.textHist2, 'Visible', 'off');

axes(handles.axes1); cla; imshow(RGB);
axes(handles.axes2); cla;
set(handles.axes1,'Visible','on');

handles = updateHistograms(handles);
% Update handles structure
guidata(hObject, handles);

% --- Executes on button press in CopyButton.
function CopyButton_Callback(hObject, eventdata, handles)
% hObject    handle to CopyButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (handles.fileLoaded==1)
    handles.RGB2 = handles.RGB;
    axes(handles.axes2); imshow(handles.RGB2);
    handles.fileLoaded2 = 1;
    handles = updateHistograms(handles);
    guidata(hObject, handles);
else
    h = msgbox('No primary file has been loaded!','Error','error');
end
```

```
% --- Executes on button press in MedianButton.
function MedianButton_Callback(hObject, eventdata, handles)
% hObject    handle to MedianButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if (handles.fileLoaded2==1)
    [M,N,ttt] = size(handles.RGB2);
    handles.RGBtmp = handles.RGB2;

elseif(handles.fileLoaded==1)
    [M,N,ttt] = size(handles.RGB);
    handles.RGBtmp = handles.RGB;

else
    h = msgbox('No primary file has been loaded!','Error','error');
        return;
end

    RUN = 1;

    while (RUN==1)

        prompt = {'Enter Median Row Factor (0-5%):','Enter Median 
Column Factor (0-5%):'};
        dlg_title = 'Enter Median Parameters:';
        num_lines = 1; %number of lines for text box
        def = {'2','2'}; %default input values
        answer = inputdlg(prompt,dlg_title,num_lines,def); %answer 
taken as a cell-array
        if (isempty(answer))
            return;
        end

        M1 = str2num(answer{1})/100;
        M2 = str2num(answer{2})/100;

        if ((str2num(answer{1})>=0) && (str2num(answer{1})<=5)) && 
((str2num(answer{2})>=0) && (str2num(answer{2})<=5))
            RUN = 0;
        end
    end

    M1 = round(M1 * M);
    M2 = round(M2 * N);

    w = waitbar(0, 'Median filtering ... Please wait ...');
    handles.RGB2(:,:,1) = medfilt2(handles.RGBtmp(:,:,1),[M1 M2]); % 
Median filtering for Red component
    waitbar(1/3, w);
    handles.RGB2(:,:,2) = medfilt2(handles.RGBtmp(:,:,2),[M1 M2]); %
Median filtering for Green component
    waitbar(2/3, w);
    handles.RGB2(:,:,3) = medfilt2(handles.RGBtmp(:,:,3),[M1 M2]); %
Median filtering for Blue component
    close(w);
    axes(handles.axes2); imshow(handles.RGB2);
    handles.fileLoaded2 = 1;
    handles = updateHistograms(handles);

    guidata(hObject, handles);

% --- Executes on button press in SharpButton.
function SharpButton_Callback(hObject, eventdata, handles)
% hObject    handle to SharpButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if (handles.fileLoaded2==1)
 handles.RGBtmp = handles.RGB2;
elseif (handles.fileLoaded==1)
 handles.RGBtmp = handles.RGB;
else
    h = msgbox('No primary file has been loaded!','Error','error');
    return
end

    H = fspecial('unsharp');
    handles.RGB2(:,:,1) =
imfilter(handles.RGBtmp(:,:,1),H,'replicate');
    handles.RGB2(:,:,2) =
imfilter(handles.RGBtmp(:,:,2),H,'replicate');
    handles.RGB2(:,:,3) =
imfilter(handles.RGBtmp(:,:,3),H,'replicate');
    axes(handles.axes2); imshow(handles.RGB2);
    handles.fileLoaded2 = 1;
    handles = updateHistograms(handles);
    guidata(hObject, handles);

% --- Executes on button press in GrayButton.
function GrayButton_Callback(hObject, eventdata, handles)
% hObject    handle to GrayButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if (handles.fileLoaded2==1)
 handles.RGBtmp = handles.RGB2;
elseif (handles.fileLoaded==1)
 handles.RGBtmp = handles.RGB;
else
```

```
    h = msgbox('No primary file has been loaded!','Error','error');
    return
end
    Gray = rgb2gray(handles.RGBtmp);
    handles.RGB2(:,:,1) = Gray;
    handles.RGB2(:,:,2) = Gray;
    handles.RGB2(:,:,3) = Gray;
    axes(handles.axes2); imshow(handles.RGB2);
    handles.fileLoaded2 = 1;
    handles = updateHistograms(handles);
    guidata(hObject, handles);

% --- Executes on button press in SaveButton.
function SaveButton_Callback(hObject, eventdata, handles)
% hObject    handle to SaveButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


if (handles.fileLoaded2==1)
    [file,path] = uiputfile('*.jpg','Save Secondary Image As');
    imwrite(handles.RGB2,[path file],'jpg');
else
    h = msgbox('No secondary file has been loaded!','Save
Error','error');
end

function editSize_Callback(hObject, eventdata, handles)
% hObject    handle to editSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editSize as text
%        str2double(get(hObject,'String')) returns contents of
editSize as a double

% --- Executes during object creation, after setting all properties.
function editSize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editSize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function editComment_Callback(hObject, eventdata, handles)
% hObject    handle to editComment (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editComment as
text
%        str2double(get(hObject,'String')) returns contents of
editComment as a double

% --- Executes during object creation, after setting all properties.
function editComment_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editComment (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function editPath_Callback(hObject, eventdata, handles)
% hObject    handle to editPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editPath as text
%        str2double(get(hObject,'String')) returns contents of
editPath as a double

% --- Executes during object creation, after setting all properties.
function editPath_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function sliderBright_Callback(hObject, eventdata, handles)
% hObject    handle to sliderBright (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
range of slider
set(handles.editBright,'String', sprintf('%10s:%4.0f%%',
'Brightness', 100*get(handles.sliderBright,'Value')));

% --- Executes during object creation, after setting all properties.
function sliderBright_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderBright (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function sliderContrast_Callback(hObject, eventdata, handles)
% hObject    handle to sliderContrast (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine
range of slider
set(handles.editContrast,'String', sprintf('%10s:%4.0f%%',
'Contrast', 100*get(handles.sliderContrast,'Value')));

% --- Executes during object creation, after setting all properties.
function sliderContrast_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderContrast (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on button press in pushbuttonContrastBrightness.
function pushbuttonContrastBrightness_Callback(hObject, eventdata,
handles)
```

```
% hObject    handle to pushbuttonContrastBrightness (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (handles.fileLoaded2==1)
 handles.RGBtmp = handles.RGB2;
elseif (handles.fileLoaded==1)
 handles.RGBtmp = handles.RGB;
else
    h = msgbox('No primary file has been loaded!','Error','error');
    return
end
    handles.RGB2 = changeBrightness(handles.RGBtmp,
get(handles.sliderBright, 'Value'), get(handles.sliderContrast,
'Value'));
    axes(handles.axes2); imshow(handles.RGB2);
    handles.fileLoaded2 = 1;
    handles = updateHistograms(handles);
    guidata(hObject, handles);

function handlesNew = updateHistograms(handles)
handlesNew = handles;
if (handles.fileLoaded == 1)
    set(handles.textHist1, 'Visible', 'on');
    axes(handlesNew.axesHist1);
    cla;
    ImageData1 = reshape(handlesNew.RGB(:,:,1),
[size(handlesNew.RGB, 1) * size(handlesNew.RGB, 2) 1]);
    ImageData2 = reshape(handlesNew.RGB(:,:,2),
[size(handlesNew.RGB, 1) * size(handlesNew.RGB, 2) 1]);
    ImageData3 = reshape(handlesNew.RGB(:,:,3),
[size(handlesNew.RGB, 1) * size(handlesNew.RGB, 2) 1]);
    [H1, X1] = hist(ImageData1, 1:5:256);
    [H2, X2] = hist(ImageData2, 1:5:256);
    [H3, X3] = hist(ImageData3, 1:5:256);
    hold on;
    plot(X1, H1, 'r');
    plot(X2, H2, 'g');
    plot(X3, H3, 'b');
    axis([0 256 0 max([H1 H2 H3])]);
end
if (handlesNew.fileLoaded2 == 1)
    set(handles.textHist2, 'Visible', 'on');
    axes(handlesNew.axesHist2);
    cla;
    ImageData1 = reshape(handlesNew.RGB2(:,:,1),
[size(handlesNew.RGB2, 1) * size(handlesNew.RGB2, 2) 1]);
    ImageData2 = reshape(handlesNew.RGB2(:,:,2),
[size(handlesNew.RGB2, 1) * size(handlesNew.RGB2, 2) 1]);
    ImageData3 = reshape(handlesNew.RGB2(:,:,3),
[size(handlesNew.RGB2, 1) * size(handlesNew.RGB2, 2) 1]);
```

```
    [H1, X1] = hist(ImageData1, 1:5:256);
    [H2, X2] = hist(ImageData2, 1:5:256);
    [H3, X3] = hist(ImageData3, 1:5:256);
    hold on;
    plot(X1, H1, 'r');
    plot(X2, H2, 'g');
    plot(X3, H3, 'b');
    axis([0 256 0 max([H1 H2 H3])]);
end

function editBright_Callback(hObject, eventdata, handles)
% hObject    handle to editBright (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editBright as
text
% str2double(get(hObject,'String')) returns contents of editBright
as a double


% --- Executes during object creation, after setting all properties.
function editBright_CreateFcn(hObject, eventdata, handles)
% hObject handle to editBright (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editContrast_Callback(hObject, eventdata, handles)
% hObject handle to editContrast (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editContrast as
text
% str2double(get(hObject,'String')) returns contents of editContrast
as a double


% --- Executes during object creation, after setting all properties.
function editContrast_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to editContrast (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

MATLAB PROGRAM FOR SLIDE SHOW CONTROL

```
clear; clc;

%image url for capture may change according to connection

url = 'http://100.72.61.138:8080/shot.jpg';

%start ActiveX server and connect with PowerPoint Application
ppt = actxserver('powerpoint.application');
ppt.Visible = 1;
h = ppt.Presentations.Open('C:\Users\SAMSUNG\Desktop\abc.ppt');

pause(5);

while 1

%Read image from camera
ss = imread(url);

%Segmentation
k=ss(:,:,1)<=135 & ss(:,:,1)>=100 & ss(:,:,2)<=40 &
ss(:,:,2)>=10 & ss(:,:,3)<=40 & ss(:,:,3)>=10;

%Dilation followed by erosion and filling
se=strel('disk', 5);
e=imclose(k, se);
f=imfill(e, 'holes');

%count number of connected components
[stat num]=bwlabel(f);
disp(num);

pause(0.5);

if num==1
```

```
%move to next slide
ppt.ActivePresentation.SlideShowWindow.View.Next;

elseif num==2
%move to previous slide
ppt.ActivePresentation.SlideShowWindow.View.Previous;

elseif num==3
%exit the slideshow
ppt.ActivePresentation.SlideShowWindow.View.Exit;

elseif num==4
break;

else

end

pause(1);

end

%Save and close the Presentation
ppt.ActivePresentation.Save;
ppt.ActivePresentation.Close;

%disconnect from server and delete the object
ppt.Quit;
ppt.delete;
```

MATLAB PROGRAM FOR CONTROL OF ROOM LIGHT AND/OR LED LIGHTS OR OTHER APPLIANCES

```
clc; clear all;

last_state=[0 0 0 0];        %Stores last state of four LEDs

a=arduino ('COM44');         %object constructor for Arduino control

pin=[12 11 10 9];            %Pin number on Arduino board

%declare pin mode as output for the Digital Pins on Arduino board
a.pinMode(pin(1),'output');
a.pinMode(pin(2),'output');
a.pinMode(pin(3),'output');
a.pinMode(pin(4),'output');
```

```
%initialising the video object with image capturing device
v=videoinput('winvideo', 2);


while 1

i=getsnapshot(v);
y=ycbcr2rgb(i);

%thresolding or feature extraction

k=y(:,:,1)<=255 & y(:,:,1)>=59 & y(:,:,2)<=255 & y(:,:,2)>=53 &
y(:,:,3)<=255 & y(:,:,3)>=32;

f=imfill(k,'holes');
se=strel('disk',10);
e=imerode(f,se);

%count number of Region of Interest

[stat num]=bwlabel(e);

%Take a decision based on the calculated Region of Interest

if num==1
     disp('1');

if last_state(1)==0
     a.digitalWrite(pin(1),1);
     last_state(1)=1;

elseif last_state(1)==1
     a.digitalWrite(pin(1),0);
     last_state(1)=0;
end

elseif num==2

    disp('2');

    if last_state(2)==0
        a.digitalWrite(pin(2),1);
        last_state(2)=1;

    elseif last_state(2)==1
        a.digitalWrite(pin(2),0);
        last_state(2)=0;

    end
```

```
elseif num==3

    disp('3');

    if last_state(3)==0
        a.digitalWrite(pin(3),1);
        last_state(3)=1;

    elseif last_state(3)==1
        a.digitalWrite(pin(3),0);
        last_state(3)=0;

    end

elseif num==4

    disp('4');

    if last_state(4)==0
        a.digitalWrite(pin(4),1);
        last_state(4)=1;

    elseif last_state(4)==1
        a.digitalWrite(pin(4),0);
        last_state(4)=0;

    end

elseif num==5

    a.digitalWrite(pin(1),0);
    a.digitalWrite(pin(2),0);
    a.digitalWrite(pin(3),0);
    a.digitalWrite(pin(4),0);
    last_state(1:4)=0;

    break;

else
    disp(0);
end                     %end of if

end                     % end of while

delete(a);          % delete the object constructor
```

MATLAB function ‘COLORSEG’ for performing colour segmentation of RGB images

```
function I = colorseg(varargin)
%COLORSEG Performs segmentation of a color image.
%   S = COLORSEG('EUCLIDEAN ' , F, T, M) performs segmentation of
color image F %   using a Euclidean measure of similarity. M is a 1-
by-3 vector representing the %    average color used for
segmentation (this is the center of the sphere in Fig. 6.26 %    of
DIPUM). T is the threshold against which the distances are compared.
%
%   S = COLORSEG( ' MAHALANOBIS ' , F, T, M, C) performs
segmentation ofcolor % image F using the Mahalanobis distance as a
measure ofsimilarity. C is the 3-by -%  3 covariance matrix of the
sample color vectors of the class of interest. See % function
covmatrix for the computation of C and M.
%
%   S is the segmented image (a binary matrix) in which 0s denote
the background.

% Preliminaries.
% Recall that varargin is a cell array.
f = varargin{2};
if(ndims(f) ~= 3) || (size(f, 3) ~= 3)
error ( ' Input image must be RGB . ' ) ;
end

M = size(f, 1); N = size(f, 2);
% Convert f to vector format using function imstack2vectors.
f = imstack2vectors(f);
f = double(f);
% Initialize I as a column vector. It will be reshaped later into an
image.

I = zeros(M*N, 1);
T = varargin{3};
m = varargin{4};
m = m(:)'; % Make sure that m is a row vector.
if length(varargin) == 4
method = ' euclidean ' ;
elseif length(varargin) == 5
method = 'mahalanobis';
else
error( 'Wrong number of inputs. ');
end

switch method
case 'euclidean'
    % Compute the Euclidean distance between all rows of X and m.
See Section 12.2  % of DIPUM for an explanation of the following
expression. D(i) is the Euclidean    % distance between vector
X(i,:) and vector m.
```

```
    p = length(f);
    D = sqrt(sum(abs(f - repmat(m, p, 1)).^2, 2));
case 'mahalanobis'
C = varargin{5};
D = mahalanobis(f, C, m);
otherwise
error( ' Unknown segmentation method. ')
end

% D is a vector of size MN-by-1 containing the distance
computationsfrom all the color % pixels to vector m. Find the
distances <= T.

J = D <= T;

% Set the values of I(J) to 1. These are the segmented color pixels.

I (J) = 1;
% Reshape I into an M-by-N image.
I = reshape(I, M, N);
```

MATLAB function 'mahalanobis' for computing the mahalanobis distance when using colour segmentation

```
function D = mahalanobis(varargin)
% MAHALANOBIS computes the mahalanobis distance.
%   D = MAHALANOBIS (Y, X) computes the mahalanobis Distance between
each vector %  in Y to the mean (centroid) of the vector in X, and
outputs the result in vector D, %   whose length is size(Y,1). The
vectors in X and Y are assumed to be organized as %  rows. The input
data can be real or complex. The outputs are real quantities.
%
%   D = MAHALANOBIS (Y, CX, MX) computes the mahalanobis Distance
between % each vector in Y and the given mean vector MX. The result
are output in vector D%    whose length is Size(Y, 1). The vectors
in Y are assumed to be organized as the%    rows of this array. The
input data can be real or complex. The outputs are real%
quantities. In addition to the mean vector MX, the covariance matrix
CX of a%   population of vectors X must be provided also. Use
function COVMATRIX(Section%  11.5) to compute MX and CX.

param = varargin; % keep in mind that param is a cell array.
Y = param(1);

if length(param) == 2
    X = param{2};
    %compute the mean vector and covariance matrix of the vector in
X.
    [Cx, mx] = covmatrix(X);

elseif length(param) == 3 % cov. Matrix and mean vector provided.
    Cx = param{2};
    mx=param{3};

else
    error('Wrong number of inputs')

end

mx=mx(:)'; % Make sure that mx is a row vector for the next step.

% subtract the mean vector from each vector in Y.
Yc = bsxfun(@minus, Y, mx);

% compute the mahalanobis distances.
D= real(sum(Yc/Cx.*conj(Yc), 2));
```

MATLAB function for Adaptive Median Filtering of Input Image

```
function  f  =  adpmedian(g,  Smax)
% ADPMEDIAN  Perform  adaptive  median  filtering.
%  F  =  ADPMEDIAN(G,  SMAX) performs adaptive median filtering of
%  image  G.  The median filter starts at size 3-by-3 and iterates
%  up  to  size  SMAX - by - SMAX. SMAX must be an odd integer
greater
%  than  1.
%  SMAX  must  be  an  odd,  positive  integer  greater  than  1.

if  (Smax  <=  1)  ||  (Smax/2  ==  round(Smax/2))  ||  (Smax  ~=
round(Smax))
error(' SMAX  must  be  an  odd  integer > 1.')
end
%  Initial  setup.
f  =  g;
f(:)  =  0;

alreadyProcessed  =  false(size(g));

%  Begin  filtering.

for  k  =  3:2:Smax
zmin = ordfilt2(g,  1,  ones(k,  k),  'symmetric');
zmax = ordfilt2(g,  k  *  k,  ones(k,  k),  'symmetric');
zmed = medfilt2(g,  [k k],  'symmetric');
```

```
processUsingLevelB = (zmed > zmin) & (zmax > zmed) &
~alreadyProcessed;
zB = (g > zmin) & (zmax > g);
outputZxy = processUsingLevelB & zB;
outputZmed = processUsingLevelB & ~zB;
f(outputZxy) = g(outputZxy);
f(outputZmed) = zmed(outputZmed);
alreadyProcessed = alreadyProcessed | processUsingLevelB;
if all(alreadyProcessed(:))
break;
end
end
% Output zmed for any remaining unprocessed pixels. Note that
%this zmed was computed using a window of size Smax-by- Smax,
%which is the final value of k in the loop.
f( ~alreadyProcessed) = zmed(~alreadyProcessed);
```

CHAPTER-10
CONCLUSION AND FUTURE WORK

It is noteworthy to mention that an accuracy rate in excess of 80% were obtained by the algorithm in making a prediction about what the user intends to perform. The results were affected by how fast the user changed the gestures as well as the light conditions of the surrounding environment. Also, when colour bands were used, the detection rates were marginally higher because of the ease of extracting region of interest.

This project on development of gesture-controlled appliances proved to be a great learning experience in terms of assessing the power of image processing to develop simple systems that add a new dimension to human-machine interaction as well as other potential uses of this system. Image processing has wider applications in medical imaging, remote sensing, identity verification, estimating count of materials in bulk-but discrete quantities, surface deformities and defects in manufactured materials, and abnormalities in fruits and vegetables grown in farmlands. Potential methods to use image processing in these areas were studied and would form the basis for later projects to be undertaken. However, above all this, the importance of image processing in Human Computer Interaction cannot be undermined, as already demonstrated by this project.

This project is not the end but means to develop better and more refined algorithms, more efficient models that continuously raise the levels of how well the machine can understand the human.

The future projects will maybe focus on incorporating advanced techniques of segmentation and analysis coupled with self-learning systems that will be much more useful and much more powerful.

For now this project on using a variety of tools together was a great learning step in that direction.

CHAPTER-11
APPENDICES

The Arduino Duemilanove ("2009") is a microcontroller board based on the ATmega168 or ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

Summary

Microcontroller	ATmega168 or ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	16KB (ATmega168) or 32KB (ATmega328) of which 2KB used by boot loader
SRAM	1 KB (ATmega168) or 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1 KB (ATmega328)
Clock Speed	16 MHz

Memory

The ATmega168 has 16 KB of flash memory for storing code (of which 2 KB is used for the boot loader); the ATmega328has 32 KB, (also with 2 KB used for the boot loader). The ATmega168 has 1 KB of SRAM and 512 bytes of EEPROM (which can be read and written with the EEPROM library); the ATmega328 has 2 KB of SRAM and 1 KB of EEPROM.

Input and Output

Each of the 14 digital pins on the Duemilanove can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

CHAPTER-12
REFERENCES

[1] The Mathworks. Image Processing Toolbox User’s Guide Version 2.1. MA: The Mathworks, Inc, January 1998.

[2] R. C. Gonzalez and R. E. Woods, “Object recognition,” in Digital Image Processing, Second Edition. Upper Saddler River, NJ: Prentice Hall Inc, 2002.

[3] R. C. Gonzalez and R. E. Woods, “Image Segmentation,” in Digital Image Processing, Second Edition. Upper Saddler River, NJ: Prentice Hall Inc, 2002.

[4] ZHAO Ping; LI Yongkui, “Grain Counting Method Based On Image Processing”, 2009 IEEE.

[5] Nur Badariah Ahmad Mustafa, Nurashikin Ahmad Fuad, Syed Khaleel Ahmed, Aidil Azwin Zainul Abidin, Zaipatimah Ali, Wong Bing Yit, and Zainul Abidin Md Sharrif, “Image Processing of an Agriculture Produce: Determination of Size and Ripeness of a Banana”, 2008 IEEE.

[6] Tien Dzung Nguyen, Quyet Hoang Manh, Phuong Bui Minh, Long Nguyen Thanh, Thang Manh Hoang, “Efficient and reliable camera based multiple-choice test grading system”, 2011 International Conference on Advanced Technologies for Communications (ATC 2011), 2011 IEEE, Pg- 268-271.

[7] Woodford, K. & Bancroft, P. (2004). Using Multiple choice questions effectively in Information Technology education. In R. Atkinson, C. McBeath, D. Jonas-Dwyer & R. Phillips (Eds).

[8] Hui Deng, Feng Wang, Bo Liang, "A Low-Cost OMR Solution for Educational Applications," IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA, pp.967-970, 2008.

[9] S. N. Srihari and V. Govindaraju. Analysis of textual images using the Hough transform. Machine Vision and Applications, vol. 2 pp: 141-153, 1989.

[10] Arnaud Bernard, Benny Bing, "Hand Gesture Video Browsing for Broadband-Enabled HDTVs".

[11] Chen, F., Fu, C., Huang, C., "Hand Gesture recognition using a real-time tracking method and hidden Markov models", Image and Vision Computing, Vol. 21, 2003, pp. 745–758.

[12] M.K. Bhuyan, D. Ghosh and P.K. Bora "Feature Extraction from 2D Gesture Trajectory in Dynamic Hand Gesture Recognition", IEEE-2006

[13] Hongmo, J., Jiman, K. and Daijin, K. "Hand Gesture Recognition To Understand Musical Conducting Action". in Robot and Human Interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on. 2007.

[14] Just, A., and Marcel, S., "A comparative study of two state-of-the-art sequence processing techniques for hand gesture recognition". Comput. Vis. Image Underst., 2009. 113(4): p. 532-543.

CHAPTER-13
BIBLIOGRAPHY

[1] Website of Arduino: http://www.arduino.cc/en

[2] Website of Matrix Laboratory: http://www.mathworks.in

[3] Course on Image processing: http://www.coursera.org

[4] MATLAB Image Processing Toolbox: http://www.mathworks.in/help/images/index.html

[5] Arduino Duemilanove board specifications: http://arduino.cc/en/Main/arduinoBoardDuemilanove

[6] Arduino tutorial: http://www.ladyada.net/learn/arduino/

[7] Arduino software download: http://arduino.cc/en/Main/Software

[8] An Introduction to Digital Image Processing with MATLAB: Notes for SCM2511, Semester 1, 2004, Alasdair McAndrew, Victoria University of Technology

[9] A Guide to MATLAB for Beginners and Experienced Users: Brian R. Hunt, Ronald L. Lipsman, Jonathan M. Rosenberg. Cambridge University Press.

[10] Image Processing Toolbox for use with MATLAB: Version 5

[11] EECE/CS 253 Image Processing Lecture Notes, Richard Alan Peters II, Fall Semester 2006, Vanderbilt University School of Engineering (licensed under the Creative Commons Attribution-Non-commercial 2.5 License)

[12] Rafael Gonzalez and Richard E. Woods. Digital Image Processing. Addison-Wesley, second edition, 2002.

[13] Web link for Arduino Toolbox in MATLAB: http://www.mathworks.in/academia/arduino-software/arduino-MATLAB.html

[14] ATmega 328 datasheet: http://www.atmel.com/dyn/resources/prod_documents/doc8161.pdf

Copyrights information:

[1] MATLAB® is registered trademark of The MathWorks, Inc.

[2] The Image Processing GUI developed is based on a similar GUI developed by Theodoros Giannakopoulos, Department of Informatics and Telecommunications. University of Athens, Greece
Copyright (c) 2011, Theodoros Giannakopoulos.

[3] Windows Media Player, Microsoft PowerPoint 2007 (part of MS-Office 2007 package) and ActiveX are products of Microsoft Corporation.

[4] IP Webcam App used for image capture is developed by Pavel Khlebovich. Email – pas.anddev@gmail.com

[5] Android OS is the product of Google Inc.