Compression Techniques vs Huffman Coding

Vikas Kumar
Departement of Electronics, Instrumentation and control Engineering, Azad Institute of Engineering & Technology, Lucknow (Uttar Pradesh Technical University, Lucknow)

Article Info

Article history:

Received Nov 12 2014
Revised Feb 20, 2015
Accepted Mar 26, 2015

Keyword:

Dct
Huffman codes
Huffman decoding
Huffman encoding
Lzw
Rle
Source reduction
Symbol

ABSTRACT

The technique for compressioning the Images has been increasing because the fresh images need large amounts of disk space. It is seems to be a big disadvantage during transmission & storage of image. Even though there are so many compression technique already presents and have better technique which is faster, memory efficient and simple, and friendly with the requirements of the user. In this paper we proposed the method for image compression and decompression using a simple coding technique called Huffman coding and show why this is more efficient then other technique. This technique is simple in implementation and utilizes less memory compression to other. A software algorithm has been developed and implemented to compress and decompress the given image using Huffman coding techniques in a MATLAB platform.

Corresponding Author:

Vikas Kumar,
Departement of Electronics, Instrumentation and control Engineering,
Azad Institute Of Engineering & Technology, Lucknow,
(Uttar Pradesh Technical University, Lucknow).
Email: mr.vikaskum@gmail.com

1. INTRODUCTION

When we save a digital image as a file on a camera or a web server, we are basically saving it as a long string of bits (zeros and ones). Each pixel in the image is comprised of one byte and each byte is made of 8 bits. If the image has dimensions M x N pixels, then we need MN bytes or 8MN bits to store the image. There are 256 different bytes and each has its own base 2 (bit) representation. Each of these representations are 8 bits long. So for a web browser or camera to display a stored image, it must have the string of bits in addition to the dictionary that describes the bit representation of each byte.

A digital image obtained by sampling and quantizing a continuous tone picture requires an enormous storage. For instance, a 24 bit colour image with 512x512 pixels will occupy 768 Kbyte storage on a memory disk, and a picture twice of this size will not fit in a single floppy disk. To transmit such an image over a 28.8 Kbps modem would take almost 4 minutes. The purpose for image compression is to reduce the amount of data required for representing sampled digital images and therefore reduce the cost for storage and transmission. Image compression plays a key role in many important applications, including image database, image communications, remote sensing (the use of satellite imagery for weather and other earth-resource application). The image (s) to be compressed are gray scale with pixel values between 0 to 255. The idea of compression is very simple - we simply change the bit representation of each byte with the hope that the new dictionary will have a shorter string of bits needed to store the image.

There are different techniques for compressing images. They are broadly classified into two classes called lossless and lossy compression techniques. As the name suggests in lossless compression techniques, no information regarding the image is lost. In other words, the reconstructed image from the compressed image is identical to the original image in every sense. Whereas in lossy compression, some image information is lost, i.e. the reconstructed image from the compressed image is similar to the original image but not identical to it. In this work we will use a lossless compression and decompression through a technique called Huffman coding (i.e. Huffman encoding and decoding). It is well known that the Huffman's algorithm is generating minimum redundancy codes compared to other algorithms. The Huffman coding has effectively used in text, image, video compression, and conferencing system such as, JPEG, MPEG-2, MPEG-4, and H.263 etc.

The Huffman coding technique collects unique symbols from the source image and calculates its probability value for each symbol and sorts the symbols based on its probability value. Further, from the lowest probability value symbol to the highest probability value symbol, two symbols combined at a time to form a binary tree. Moreover, allocates zero to the left node and one to the right node starting from the root of the tree. To obtain Huffman code for a particular symbol, all zero and one collected from the root to that particular node in the same order.

The main objective of this paper is to compress images by reducing number of bits per pixel required to represent it and to decrease the transmission time for transmission of images and then reconstructing back by decoding the Huffman codes. The entire paper is organized in the following sequence. Firstly we need for the compression is stated, second discribes various types of data redundancies, then the Methods\techology of compressions are explained, furtherly the implementation of lossless method of compression and decompression (i.e. Huffman Coding & Decoding) is done and finally the algorithm is developed and in the results were presented with explanation and paper concludes with References.

2. NEED FOR COMPRESSION

The following example illustrates the need for compression of digital images [3].
1. To store a colour image of a moderate size, e.g. 512×512 pixels, one needs 0.75 MB of disk space.
2. A 35mm digital slide with a resolution of 12μm requires 18 MB.
3. One second of digital PAL (Phase Alternation Line) video requires 27 MB.

A high-quality image may require 10 to 100 million bits for representation. For example, a clean photographic image requires approximately 1,280 rows of 800 pixels each, with 24 bits of color information per pixel; that is, a total of 24,576,000 bits, or 3,072,000 bytes. The large data files associated with images thus drive the need for extremely high compression ratios to make storage practical. to store about 200 pictures like that above, or, at the 24 frames per second rate of a motion picture, about 8 seconds Without compression, a CD with a storage capacity of approximately 600 million bytes would only be able of a movie.

To store these images, and make them available over network (e.g. the internet), compression techniques are needed. Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. According to mathematical point of view, this amounts to transforming a two-dimensional pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At receiver, the compressed image is decompressed to reconstruct the original image or an approximation to it.

The example below clearly shows the importance of compression. An image, 1024 pixel×1024 pixel×24 bit, without compression, would require 3 MB of storage and 7 minutes for transmission, utilizing a high speed, 64 Kbits/s, ISDN line. If the image is compressed at a 10:1 compression ratio, the storage requirement is reduced to 300 KB and the transmission time drop to less than 6 seconds.

A common characteristic of most images is that the neighboring pixels are correlated and therefore contain redundant information. The altimate task is then, is to find less correlated representation of the image. Two fundamental components of compression are redundancy and irrelevancy reduction.

Redundancy reduction aims are removing duplication from the signal source (image/video). Irrelevancy reduction omits parts of the signal that will not be noticed by the signal receiver, namely the Human Visual System (HVS). In general, three types of redundancy can be identified.

3. VARIOUS TYPES OF REDUNDANCY

In digital image compression, three basic data redundancies can be identified and exploited:
1. Coding redundancy
2. Spatial Redundancy and Temporal Redundancy
3. Irrelevant Information

Data compression is achieved when one or more of these redundancies are reduced or eliminated.

3.1. Coding Redundancy

A code is a system of symbols (letters, numbers, bits, and the like) used to represent a body of information or set of events. Each piece of information or events is assigned a sequence of code symbols, called a code word. The number of symbols in each code word is its length. The 8-bit codes that are used to represent the intensities in the most 2-D intensity arrays contain more bits than are needed to represent the intensities.

3.2. Spatial Redundancy and Temporal Redundancy

Because the pixels of most 2-D intensity arrays are correlated spatially, the information is unnecessarily replicated in the representations of the correlated pixels. In video sequence, temporally correlated pixels are also duplicate information.

3.3. Irrelevant Information

Most 2-D intensity arrays contain information that is ignored by the human visual system and extraneous to the intended use of the image. It is redundant in the sense that it is not used. Image compression research aims at reducing the number of bits needed to represent an image by removing the spatial and spectral redundancies as much as possible.

4. WHY DO WE NEED COMPRESSION?

The following chart is show the qualitative transition from simple text to full-motion video data and the disk space transmission bandwidth, and transmission time needed to store and transmit such uncompressed data. Chart contain: Multimedia data types and uncompressed storage pace, transmission bandwidth, and transmission time required. The prefix kilo-denotes a factor of 1000 rather than 1024.

Multimedia data	Size/Duration	Bits/Pixel Or Bits/Sample	Uncompressed Size (B for bytes)	Transmission Bandwidth (b for bits)	Transmission Time
A page of text	11” x 8.5”	Varying resolution	4 – 8 KB	32-64 Kb/page	1.1-2.2 sec
Telephone quality speech	10 sec	8 bps	80 KB	64 Kb/sec	22.2 sec
Grayscale image	512 x 512	8 bpp	262 KB	2.1 Mb/image	1 min 13 sec
Color image	512 x 512	24 bpp	786 KB	6.29 Mb/image	3 min 39 sec
Medical image	2048 x 2048	12 bpp	5.16 MB	41.3 Mb/image	23 min 54 sec
SHD image	2048 x 2048	24 bpp	12.58 MB	100 Mb/image	58 min 15 sec

The examples given in the above chart clearly defined the need for sufficient storage space, large transmission bandwidth, and long transmission time for image, audio, and video data.

At the present state of technology, the only solution is to compress multimedia data before its storage and transmission, and decompress it at the receiver for play back. For example, with a compression ratio of 32:1, the space, bandwidth, and transmission time requirements can be reduced by a factor of 32, with acceptable quality.

5. WHAT ARE THE DIFFERENT CLASSES OF COMPRESSION TECHNIQUES?

Compression can be divided into two categories, as Lossless and Lossy compression.

1. **Lossless coding (entropy coding)**

a. Data can be decoded to form exactly the same bits.
b. Used in .zip.
c. Can only achieve moderate compression (e.g. 2:1 - 3:1) for natural images.
d. Can be important in certain applications such as medical imaging.

The reconstructed image after compression is numerically identical to the original image. In lossy compression scheme, the reconstructed image contains degradation relative to the original.

2. Lossly source coding

a. Decompressed image is visually similar, but has been changed.

b. Used in .JPEG. and .MPEG.

c. Can achieve much greater compression (e.g. 20:1 - 40:1) for natural images.

5.1. Simple Repetition

In this If in a sequence a series on *n* successive tokens appears we can replace these with a token and a count number of occurrences. We usually need to have a special flag to denote when the repeated token appears.

For Example
8000079000000000000000000000000000000000000000
We can replace with 8f479f40
Where f is the flag for zero.
In it Compression savings depend on the content of the data.

Applications:

1. Suppression of zero's in a file (Zero Length Suppression) .
2. Silence in audio data, or pauses in conversation etc.
3. Bitmaps.
4. Blanks in text or program source files.
5. Backgrounds in images.
6. Other regular image or data tokens.

5.2. RLE

This encoding method [34] is frequently applied to images (or pixels in a scan line). In this instance, sequences of image elements (X1,X2……….,Xn)are mapped to pairs (c1,l1),(c2,l2),……..,(cn,ln) where *ci* represent image intensity or color and *li* the length of the *i*th run of pixels (Not dissimilar to zero length suppression above). The savings are dependent on the data. In the worst case (Random Noise) encoding is greater than original file.
Applications:
It is a small compression component used in JPEG compression.

5.3. Pattern Substitution

This is a simple form of statistical encoding. Here we substitute a frequently repeating pattern(s) with a code. The code is shorter than pattern giving us compression. More typically tokens are assigned according to frequency of occurrence of patterns:

1. Count occurrence of tokens
2. Sort in Descending order
3. Assign some symbols to highest count tokens

5.4. Entropy Encoding

Lossless compression frequently involves some form of entropy encoding and is based on information theoretic techniques. Shannon is father of information theory.

5.5. Huffman Coding

Huffman coding [24] is based on the frequency of occurrence of a data item (pixel in images). The principle is to use a lower number of bits to encode the data that occurs more frequently. Codes are stored in a Code Book which may be constructed for each image or a set of images. In all cases the code book plus encoded data must be transmitted to enable decoding.

5.6. Adaptive Huffman Coding

The key is to have both encoder and decoder to use exactly the same initialization and update model routines. Update model does two things:

1. increment the count
2. update the Huffman tree [25]

During the updates, the Huffman tree will be maintained its sibling property, i.e. the nodes (internal and leaf) are arranged in order of increasing weights, When swapping is necessary, the farthest node with weight W is swapped with the node whose weight has just been increased to W+1.**Note:** If the node with weight W has a subtree beneath it, then the subtree will go with it. The Huffman tree could look very different after node swapping.

5.7. Arithmetic Coding

Huffman coding and the like use an integer number (k) of bits for each symbol, hence k is never less than 1. Sometimes, e.g., when sending a 1-bit image, compression becomes impossible. [15]

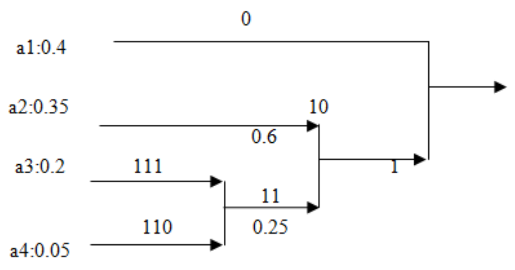
5.8. LZW

LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters. The code that the LZW algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds. The sample program runs as shown with 12 bit codes. This means codes 0-255 refer to individual bytes, while codes 256-4095 refer to substrings.

LZW compression works best for files containing lots of repetitive data. This is often the case with text and monochrome images. Files that are compressed but that do not contain any repetitive information at all can even grow bigger.
Advantages and Disadvantages:
LZW compression is fast.
Applications:
LZW compression can be used in a variety of file formats [30].
1. TIFF files
2. GIF files

6. HUFFMAN ALGORITHM

Huffman coding is an entropy encoding algorithm used for lossless data compression in computer science and information theory. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix-free code (that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol) that expresses the most common characters using shorter strings of bits than are used for less common source symbols. Huffman was able to design the most efficient compression method of this type: no other mapping of individual source symbols to unique strings of bits will produce a smaller average output size when the actual symbol frequencies agree with those used to create the code. A method was later found to do this in linear time if input probabilities (also known as weights) are sorted. For a set of symbols with a uniform probability distribution and a number of members which is a power of two, Huffman coding is equivalent to simple binary block encoding [40]e.g., ASCII coding.



6.1. Basic Technique

Assume you have a source generating 4 different symbols {$a1$,$a2$,$a3$,$a4$} with probability{0.45;0.35;0.25;0.05}. Generate a binary tree from left to right taking the two less probable symbols, putting them together to form another equivalent symbol having a probability that equals the sum of

the two symbols. Keep on doing it until you have just one symbol. Then read the tree backwards, from right to left, assigning different bits to different branches.
The final huffman code is:

6.2. Symbol Code

The technique works by creating a binary tree of nodes. These can be stored in a regular array, the size of which depends on the number of symbols (N). A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the symbol itself, the weight (frequency of appearance) of the symbol and optionally, a link to a parent node which makes it easy to read the code (in reverse) starting from a leaf node. Internal nodes contain symbol weight, links to two child nodes and the optional link to a parent node. As a common convention, bit '0' represents following the left child and bit '1' represents following the right child. A finished tree has N leaf nodes and N−1 internal nodes. A linear-time method to create a Huffman tree is to use two queues, the first one containing the initial weights (along with pointers to the associated leaves), and combined weights (along with pointers to the trees) being put in the back of the second queue. This assures that the lowest weight is always kept at the front of one of the two queues.

a1	0
a2	10
a3	111
a4	110

7. DEVELOPMENT OF HUFFMAN CODING AND DECODING ALGORITHM

Step1- Read the image on to the workspace of the matlab.
Step2- Convert the given colour image into grey level image.
Step3- Call a function which will find the symbols (i.e. pixel value which is non-repeated).
Step4- Call a function which will calculate the probability of each symbol.
Step5- Probability of symbols are arranged in decreasing order and lower probabilities are merged and this step is continued until only two probabilities are left and codes are assigned according to rule that :the highest probable symbol will have a shorter length code.
Step6- Further Huffman encoding is performed i.e. mapping of the code words to the corresponding symbols will result in a compressed data.
Step7- The original image is reconstructed i.e. decompression is done by using Huffman decoding.
Step8- Generate a tree equivalent to the encoding tree.
Step9- Read input character wise and left to the table until last element is reached in the table .
Step10-Output the character encode in the leaf and return to the root, and continue the step9 until all the codes of corresponding symbols are known.

8. RESULTS

The input image shown in Figure 1 to which the above Huffman coding algorithm is applied for the generation of codes and then decompression algorithm(i.e. Huffman decoding) is applied to get the original image back from the generated codes, which is shown in the Figure 2. The number of saved bits is the difference between the no of bits required to represent the input image i.e. shown in the Figure 1 by considering each symbol can take a maximum code length of 8 bits and the no of bits taken by the Huffman code to represent the compressed image i.e. Saved bits = (8*(r*c)-(l1*l2))=3212, r and c represents size of the input matrix, l1 and l2 represents the size of Huffman code. The compression ratio is the ratio of number of bits required to represent the image using Huffman code to the no of bits used to represent the input image. i.e. Compression ratio = (l1*l2)/ (8*r*c) = 0.8456, The output image is the decompressed image i.e. from the Figure 2 it is clear that the decompressed image is approximately equal to the input image.

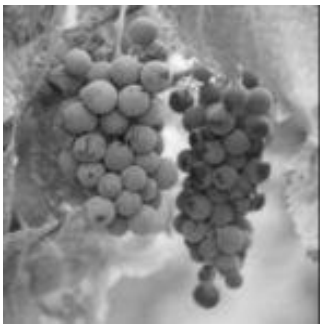


Figure 1. Input image

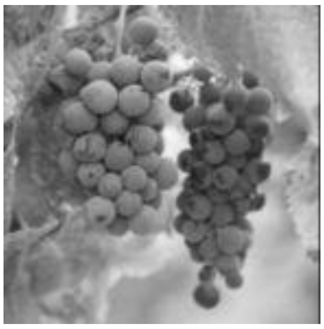


Figure 2. Decompress image

9. MAIN PROPERTIES

1. Unique Prefix Property: no code is a prefix to any othercode (all symbols are at the leaf nodes) great for decoder, unambiguous.
2. If prior statistics are available and accurate, then Huffman coding is very good.
3. The frequencies used can be generic ones for the application domain that are based on average experience, or they can be the actual frequencies found in the text being compressed.
4. Huffman coding is optimal when the probability of each input symbol is a negative power of two.
5. The worst case for Huffman coding can happen when the probability of a symbol 6 ceeds 2-1 = 0.5, making the upper limit of inefficiency unbounded. These situations often respond well to a form of blocking called run-length encoding.

10. ADVANTAGES

a. Algorithm is easy to implement
b. Produce a lossless compression of images

11. DISADVANTAGES

a. Efficiency depends on the accuracy of the statistical model used and type of image.
b. Algorithm varies with different formats, but few get any better than 8:1 compression.
c. Compression of image files that contain long runs of identical pixels by Huffman is not as efficient when compared to RLE.
d. The Huffman encoding process is usually done in two passes. During the first pass, a statistical model is built, and then in the second pass the image data is encoded based on the generated model. From here we can see that Huffman encoding is a relatively slow process as time is required to build the statistical model in order to archive an efficient compression rate.

12. CONCLUSION

The experiment shows that the higher data redundancy helps to achieve more compression. The above presented a new compression and decompression technique based on Huffman coding and decoding for scan testing to reduce test data volume, test application time. Experimental results show that up to a 0.8456compression ratio for the above image is obtained .hence we conclude that Huffman coding is efficient technique for image compression and decompression to some extent. As the future work on compression of images for storing and transmitting images can be done by other lossless methods of image compression because as we have concluded above the result the decompressed image is almost same as that of the input image so that indicates that there is no loss of information during transmission. So other methods of image compression can be carried out as namely JPEG method, Entropy coding, etc.

REFERENCES

[1] Ternary Tree, F.G.K Huffman Coding Technique, Dr. Pushpa R.Suri, Madhu Goel, “Department of Computer Science & Applications”, Kurukshetra University, Kurukshetra, India
[2] Massachusetts Institute of Technology Department of Electrical Engineering and Computer Science.

[3] Compression Using Fractional Fourier Transform A Thesis Submitted in the partial fulfillment of requirement for the award of the degree of Master of Engineering In Electronics and Communication, By Parvinder Kaur.

[4] “RL-Huffman Encoding for Test Compression and Power Reduction in Scan Applications-mehrdad nourani and Mohammadh tehranipour”, The University of Texas at Dallas.

[5] Hemasundara Rao, “A Novel VLSI Architecture of Hybrid Image Compression Model based on Reversible Blockade Transform C”, Student Member, *IEEE*, M. Madhavi Latha, Member, IEEE

[6] D.A.Huffman, “A Method for the construction of Minimum-redundancy Codes”, *Proc. IRE*, vol.40, no.10, pp.1098-1101,1952.

[7] A.B.Watson,“Image Compression using the DCT”, *Mathematica Journal*, 1995,pp.81-88.

[8] David A. Huffman, Profile Background Story: Scientific American, pp. 54-58, 1991

[9] Manoj Aggrawal, Ajai Narayan, “Efficient Huffman Decoding”

[10] C. Saravanan Assistant Professor, “Computer Centre, National Institute of Technology”, Durgapur, West Bengal, India, Pin – 713209.R. PONALAGUSAMY Professor, Department of Mathematics, National Institute of Technology, Tiruchirappalli, Tamilnadu, India, Pin – 620015.

[11] http://www.rz.go.dlr.de:8081/info/faqs/graphics/jpeg1.html

[12] http://www.amara.com/IEEEwave/IEEEwavelet.html

[13] http://cm.belllabs.com/who/jelena/Wavelet/w_applets.html

[14] http://en.wikipedia.org/wiki/Lossless_JPEG

[15] http://en.wikipedia.org/wiki/Arithmetic_coding

[16] http://fly.cc.fer.hr/~eddie/Sem_DPCM.htm

[17] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression", *IEEE* Transactions on Information Theory, May 1977

[18] Rudy Rucker, "Mind Tools", Houghton Mifflin Company, 1987

[19] http://www.dogma.net/markn/index.html

[20] “Huffman's Original Article”*:* D.A. Huffman, "[3]" (PDF), Proceedings of the I.R.E., sept 1952, pp 1098-1102

[21] Background story: Profile: David A. Huffman, Scientific American, Sept. 1991, pp. 54-58

[22] Thomas H., Cormen, Charles E., Leiserson, Ronald L., Rivest, Clifford Stein, “Introduction to Algorithms”, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 16.3, pp.385–392.

[23] http://www.Huffman coding - Wikipedia, the free encyclopedia.htm

[24] http://en.wikipedia.org/wiki/Huffman_coding

[25] http://en.wikipedia.org/wiki/Adaptive_Huffman_coding

[26] Compressed Image File Formats: JPEG, PNG, GIF,XBM,BMP, John Miano, August 1999

[27] Khalid Sayood, “Introduction to Data Compression”, Ed Fox (Editor), *March 2000 IJCSNS International Journal of Computer Science and Network Security*, VOL.10 No.5, May 2010 141

[28] Managing Gigabytes: Compressing and Indexing Documents and Images, Ian H H. Witten, Alistair Moffat,Timothy C. Bell , May 1999

[29] Digital Image Processing, Rafael C. Gonzalez, Richard E.Woods, November 2001

[30] http://en.wikipedia.org/wiki/Comparison_of_graphics_file_formats

[31] Dzung Tien Hoang, Jeffery Scott Vitter, “Fast and Efficient Algorithms for Video Compression and Rate Control”, June 20,1998.

[32] V., Bhaskaranand, K., Konstantinides, “Image and video compression standards”, Kluwer Academic Publishers, Boston, MA,1995.

[33] http://en.wikipedia.org/wiki/Huffman_coding

[34] http://en.wikipedia.org/wiki/Information_theory

[35] http://en.wikipedia.org/wiki/Entropy_encoding.

[36] http://en.wikipedia.org/wiki/Lossless_data_compression.

[37] http://en.wikipedia.org/wiki/Variable-length_code.

[38] http://www.pha.jhu.edu/~sundar/intermediate/history.html

[39] http://myhome.hanafos.com/~soonjp/vchx.html

[40] http://en.wikipedia.org/wiki/Block_code

BIOGRAPHY OF AUTHOR

Vikas Kumar completed his B.tech (electronics and communication) in 2011 at Uttar Pradesh Technical University and is pursuing his M.Tech. in Advanced Electronics and Instrumentation with Specialization in control systems at Uttar Pradesh Technical University. His research interest includes image processing ,communication, control and instrumentation.
Email: mr.vikaskum@gmail.com